RTMaps

"Real Time, Multisensor, Advanced Prototyping Software"

Nicolas du Lac, Claire Delaunay, Gilles Michel

Intempora
2, place Jules Gévelot
92130 Issy les Moulineaux
http://www.intempora.com

Abstract

RTMaps Bruno Steux – « RTMaps, a software environment for real-time applications development » – PhD thesis at Ecole des Mines de Paris– Dec, 2001. (Real-Time Multisensor Advanced Prototyping Software) is a rapid development environment oriented for applications handling multiple asynchronous sensors and actuators of various types like videos, radars, Lidars, GPS, gyrometers, accelerometers, ultrasonic belts, etc. It is designed for efficiency and ease of use: it provides functionalities like modular development, graphical programming, associated with helpful functionalities for the prototyping phase like synchronized recording and replay of sensors data thanks to advanced timestamping ability.

The RTMaps kernel architecture and runtime engine has been initially developed at the "Centre de Robotique of Ecole des Mines de Paris" and is now edited by Intempora since year 2000. It is now widely used in fields like automotive and mobile robotics for perception and data-fusion applications.

Keywords

Real-Time; Multi-sensor; Modularity; Data fusion; Data logging; Architecture; Robot

1 INTRODUCTION

State-of-the-art automotive and robotics systems must embed many sensors. In order to manage the variety of embedded sensors and applications, the need for a modular software framework appears very clearly. We'll describe at first our requirements for such a framework, which might be yours too, and then we'll show how RTMaps can fulfil them.

First of all, is there a need to remind why modularity is necessary in any development framework?

- capitalization and re-use of the developments
- step by step developments with incremental upgrades
- quick and easy evolution of the applications (easy switching of sensors, algorithms, without having to break and rebuild the whole software)
- cooperative team work

Apart from that, it is often difficult to develop and test applications directly on a real system (an equipped vehicle or a robot): it is costly and time consuming for hardware maintenance

reasons, material immobilization, and for comfort issues as well (who likes developing software standing up next to the system with the keyboard in one hand and a screen somewhere far away…). This makes it necessary to use tools that can bring a hardware abstraction layer simulating the system on a desktop environment, and that can also facilitates porting desktop developments to the real-system (back and forth).

This software environment must finally enable very quick and easy integration and testing of algorithms, easy management of multi-tasking and distribution of the processing.

RTMaps has been designed to meet such requirements in that its modular architecture allows easy development of independent building blocks (also called RTMaps components) and provides integrated functionalities for recording timestamped sensors data flows and replaying them in a desktop environment in order to simulate the real-system. It was built upon experience of researchers and engineers in multi-sensor algorithms for perception and meets a lot of our needs.

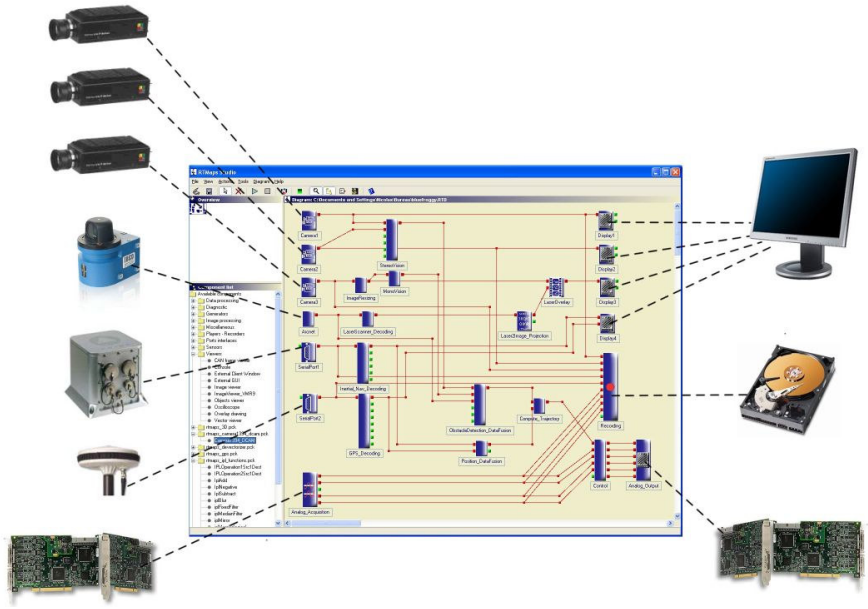

Figure 1 - The RTMaps Studio

2 THE RTMAPS ARCHITECTURE

2.1 What makes the software

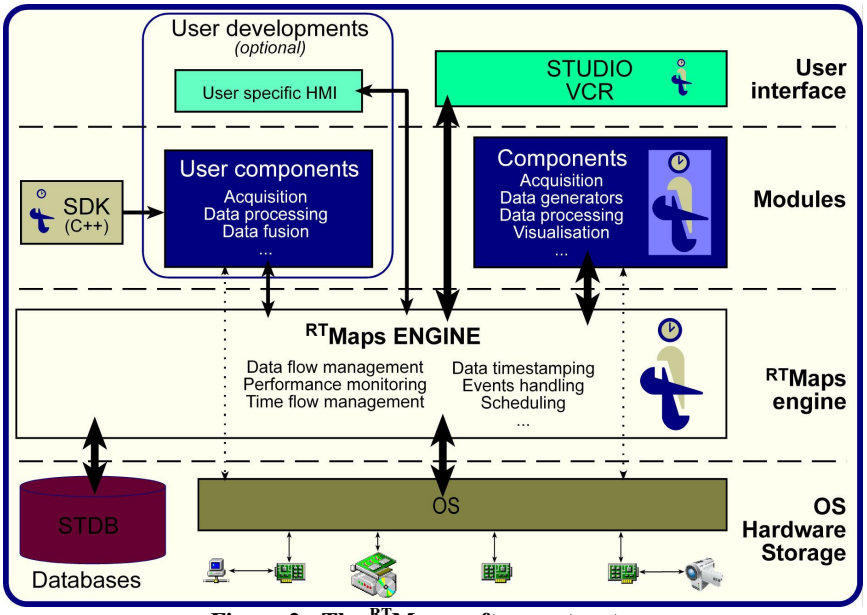

Figure 2 - The RTMaps software structure

The RTMaps software is composed of several modules (see Figure 2 - The RTMaps software structure):

- *The execution engine*:
  This is the central part of the software. It handles services such as real-time clock, inter component buffers management, threads control, dynamic loading of the RTMaps components. For optimization considerations, this part has been written in C and C++.

- *The components library*:
  It is a set of shared libraries (.dll files under Windows, .so files under Linux platforms), containing RTMaps components that can be dynamically loaded in the execution engine, and used in RTMaps applications. An RTMaps component is basically like a plug-in. See Figure 1 - The RTMaps Studio: components are represented as the blue functional blocks. A component is written in C++ and can contain any functionality like performing sensor data acquisition from a given sensor, running a given algorithm for data-processing or data fusion, provide data display functionalities, etc. An RTMaps application will be made by using different components at will, then connecting them together and configuring them.

- *The RTMaps Studio*:
  It is the graphical programming layer. It provides a graphical interface for easy development and configuration of RTMaps based applications. It has been written in Java for portability constraints and is not required for embedded use of RTMaps.

- *The RTMaps SDK*:

The Software Development Kit is made of the RTMaps component C++ API (headers and libraries), and also several additional tools like makefiles (or "projects") and wizards for code generation. It allows to write and compile your own RTMaps components (including your own algorithms for example) and integrating them easily in your RTMaps applications.

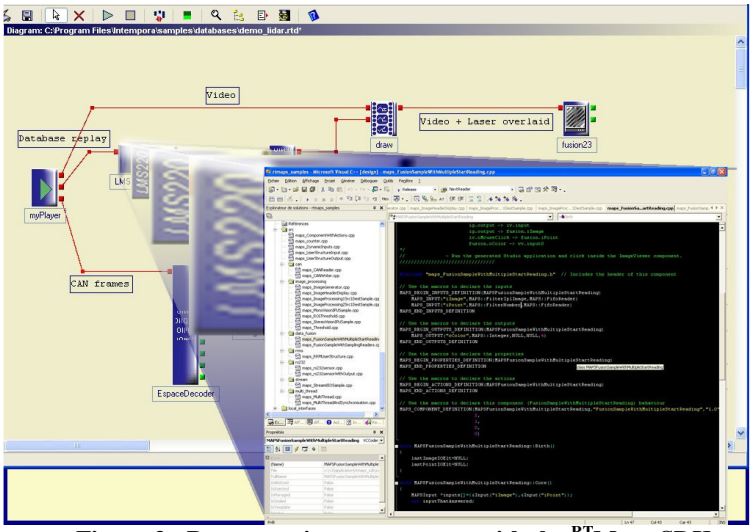

Figure 3 - Programming components with the RTMaps SDK

2.2 The RTMaps component model

Let's get inside the RTMaps architecture and study what an RTMaps component is made of and how it behaves.

As mentioned above, an RTMaps component is a building-block that can be placed (or "instantiated") on an RTMaps application (also called a "diagram") at will, a little bit like in other well-known software tools handle building blocks such as Simulink® or LabView®.

An RTMaps component is characterized by:

- *inputs*: inputs are ports that receive data. Since data is typed, a filter is specified on the inputs so that inputs accept connections only if the connected output data type passed the filter.
- *outputs*: outputs produce data. An output produces data of a given type (images, CAN frames, scalars, vectors, matrices, etc.)
- *properties*: they allow to configure the component dynamically at runtime (specifying algorithms parameters, threshold, or devices to address, etc.)
- *actions*: actions can be triggered at any time by the user and result in a specific function call in the component's code.

An output can be connected to several inputs, and input can only be connected to a single output (1 – N producer/consumer model).

So what makes the RTMaps execution model different from tools like Simulink® or LabView®?

2.2.1 Multi-threading

First of all, it can be considered that each component in RTMaps will run in its own thread, independently from the surrounding other components (note that sequential behaviour is also

available if needed). Multi-threading plays a key role in RTMaps since there is a need to handle sensors of different types, with different output frequencies (like 30Hz videos + 100Hz gyrometers + asynchronous CAN bus frames), or no output frequency at all but just asynchronous events.

Basically an RTMaps component life is most often like the following:

1. Wait for data on its input(s) and retrieve data when available (most often thanks to an event-based mechanism)
2. Process the data
3. Output the results on its output(s)
4. Loop back to step 1.

Additionally, for a threaded component, a property called "*priority*" is always available and allows adjusting the real-time priority of the component thread. This way the user can setup critical priority processing chains and lower priority ones that may stop running in case there is a lack of computing resources at some time.

2.2.2 How components synchronize and exchange data?

First of all, consider that a circular buffer (of configurable size) is associated to each component output. This buffer can contain samples of data produced on its output.

When connecting an output to another component input, the input registers on the circular buffer and can then read data

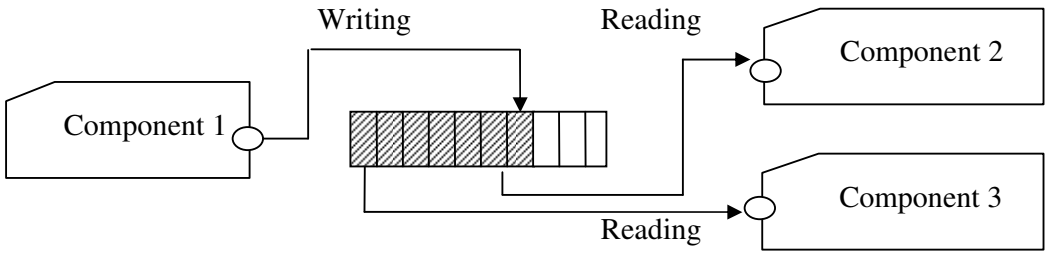

Figure 4 - Inter-components data exchange via output buffers

In order to deal with as many types of applications as possible such as *data acquisition, data logging, real-time control, real-time processing, post-processing, data display*, etc… RTMaps provides different possible behaviours for readers (inputs) like the following:

- FIFO
- Last or Next
- Wait for Next
- Sampling
- Never skipping

Let's have a closer look at each of them:

The FIFO reader:

This behaviour is the most widely used in RTMaps: in such a mode, an input will use the output buffer as a FIFO buffer. It constitutes an implementation of *asynchronous Kahn networks* [G.] Kahn, "*The semantics of a simple language for parallel programming*", in

Information Processing 74: Proceedings of IFIP Congress 74, (Stockholm, Sweden) – 1974., defined for inter-process communication, with the difference that in RTMaps only inter-thread communication is setup, and the buffer is constrained in size.

The data request on a FIFO reader input will behave the following way:

- If all the samples in the circular buffer have already been processed, the data request function will block until a new sample is generated. During this time, the component thread enters an efficient sleep state where it consumes no CPU. When a new sample data is written in the buffer by the upstream component, an event is triggered, and the data request function returns the newly arrived sample. This way, establish asynchronous inter-thread optimized communication and threads synchronization is performed transparently by the arrival of new data samples.
- If, for some reason, the consumer component cannot handle data as fast as the producer sends them, the buffer is used as a FIFO, and any data request function called on the consumer input will return immediately the oldest data in the buffer that has not been processed yet. This way, the buffer can absorb temporary lack of resources in the downstream component, which will catch up when possible.
- *Problem*: in case over-production (or under-consumption) in a durable situation, the output buffer will get filled with unprocessed data until the producer needs to overwrite samples that have not been processed yet by the consumer. Such a case can be very harmful for a real-time application for 2 reasons:
    - Data samples lost randomly
    - Latency is introduced between the 2 components since, when the downstream component can retrieve and process a sample, it will retrieve the oldest sample from the output buffer and:

      latency = circular buffer length / output frequency

  Of course, the user can be warned of such malfunction, and alternatives have to be found: a first approach can be to reduce the frame rate entering the component. The "*subsampling*" property, available on any input and output can be used for this.

The LastOrNext reader:

This behaviour is suited for example for component handling data display. The behaviour is the following:

- If all the samples in the circular buffer have already been processed, the data request function will block until a new sample is generated (the behaviour is then identical to the FIFO reader one).
- If for some reason, unprocessed samples have been stacked in the buffer, the data request function will retrieve immediately the most recent one, discarding the previous ones automatically.

The Wait for Next reader:

A data request on an input with such behaviour will simply ignore any unprocessed pending data in the output buffer and enter a wait state for new data. This model is most of the time identical to the 2 previous ones (when everyone has time to do its job), and can be used to ensure minimal latency between components even if data is missed in case of lack of resources.

The Sampling reader:

This mode can be used to re-sample a data stream in time independently of the data arrival frequency on an input.

The data request function will return immediately the most recent sample available in the output buffer, no matter whether is discards samples, or returns several times the same sample.

It can be used in conjunction with a timer or for reading or in multiple input components contexts.

The Neverskipping reader:
This mode is the only one where a downstream component can block an upstream one (without concerns about thread scheduling and priorities): a *Neverskipping* input will behave the same way as a FIFO one, but if the output buffer comes to be filled with samples, the upstream component will be blocked until some room is released.
Of course, such behaviour shall never be used in a real-time system. It is suited for data-conversion and post-processing modes where data can be replayed as fast as possible and needs to go through a processing chain without risking to be discarded.

Such approaches can be mixed into a single application depending on the jobs performed by each component. They can also be mixed for components with more than one input (usually data fusion components) for which several policies can be setup:

- asynchronous readings on multiple inputs (with FIFO reader inputs for example, and a single data request function on several inputs).
- synchronized readings on several inputs (see following chapter)
- triggered readings (1 FIFO input, and several Sampling inputs following)
- …

2.2.3 Data timestamping and application to data-fusion

Data timestamping is one of the major focuses when developing data fusion applications: knowing when such piece of data has been acquired, when such event has occurred, etc… is particularly critical when exploiting several sources of data within a same data-fusion algorithm.

- ability to estimate latencies in the application
- ability to re-synchronize data streams after processing chains with different latencies
- ability to record and playback the different sensors data tracks in a synchronized manner.

To ensure such functionalities, the RTMaps model associates 2 different timestamps with each and every sample of data.

The "Timestamp":
The timestamp associated to any piece of data represents the date of origin of the sample. It shall be defined by the most upstream producer of the samples (usually the acquisition component), and transferred "as is" by any intermediate processing component.

The "Time of Issue":
The Time of Issue of an RTMaps sample of data represents the last output date of the sample, which means the date at which the sample was output from the last component it went through. For a piece of data going through a chain of several processing components, this "time of issue" information will then increase as long as the sample traverses each stage.

At any point, it is then possible to estimate the latency of the processing chain by simply substracting the Timestamp from the Time of Issue.
The Timestamp information can also be used to resynchronize data-streams that went through different processing chains which induced different latencies: the Timestamps are then

preserved whatever the latency of each chain and the RTMaps API provides the necessary function to re-match data samples with corresponding timestamps from several outputs.

2.3 Distributing RTMaps applications [3]

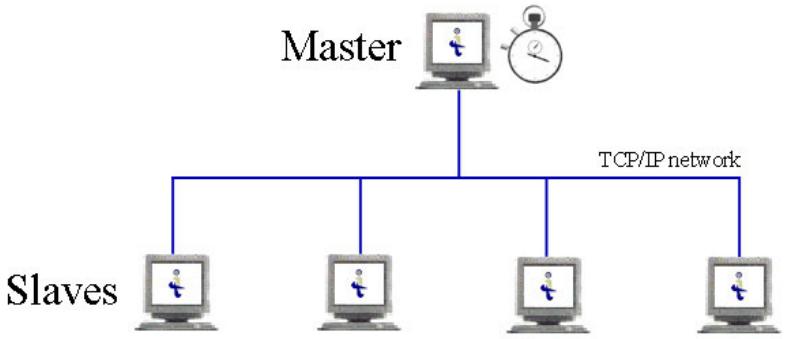

Figure 5 - Distributed RTMaps instances over a TCP/IP network

In order to allow quick and easy development of distributed applications, version 3 of RTMaps introduced functionalities aimed at:

- Synchronizing the clocks of distant RTMaps instances. This functionality enables to perform synchronized recordings on several computers,
- Exchanging data between different RTMaps instances.

These functionalities will be helpful for:

- Increasing the available computing resources by distributing the different parts of an application on several computers: indeed, when developing complex applications, a single computer can be limited in resources. These limitations can come from connectivity (number of USB or PCI slots for example), CPU computation power (several complex algorithms such as image processing, neural networks algorithms etc… may not have enough time to run), bandwidth (when exploiting several high speed data sources such as cameras, a standard PCI bus can be overloaded, standard hard disks may not be able to record more than 50 MB/s also).
- Allowing distant cooperative applications that may exchange data and information between them: different sensors may be placed at distant locations and may not have the ability to be connected to a single PC due to cabling issues for example, or moving agents may need to establish wireless connections between them or with the infrastructure.

2.3.1 Synchronizing RTMaps clocks distantly

The RTMaps IP based clocks synchronization architecture is based on a 1 Master – N slaves model. The slaves can connect to an RTMaps master, then start exchanging synchronization frames with the master to synchronize their own clock on the master clock. The protocol used is similar to NTP (Network Time Protocol).

In such a mode, the Master also sends commands to the Slave so that execution starts and stops simultaneously on the different systems, or a Master may also configure distantly the slave systems.

Independently from that, when no IP connection is available between different systems, it is possible to synchronize the clocks thanks to an external time source like a GPS sensor for example. Indeed, any RTMaps component implementing the necessary interface can behave as a time source for the entire RTMaps application which runs it. This allows performing synchronized data acquisition sessions on distant systems such as on-board systems and infrastructure based sensors for instance.

2.3.2 Data flows communications

Data exchange between several RTMaps instances can be performed independently of the Master-Slave(s) functionalities via specific Socket components.

Such components have the ability to serialize any RTMaps piece of data (images, CAN frames, stream data, numerical data, etc…) and send it to other instances of RTMaps via TCP, UDP or Multicast in order for it to be used within a distant application.

Socket components also have the ability to transmit the timestamps along with the pieces of data, so that if the RTMaps clocks are synchronized, the data timestamps are still coherent from one computer to another. Data fusion operations can then be performed correctly on the entire RTMaps network.

3 SOME APPLICATIONS USING RTMAPS

3.1 A robotized vehicle by Induct

For the DARPA Urban Challenge, the Dotmobilteam (http://www.dotmobilteam.com) equipped a Renault Scenic vehicle as an independent robot. The most advanced technologies have been used in the car like:

- 2 IBEO Alasca XT laser scanners
- an RTK GPS receiver
- a high accuracy IMU from IXSEA
- a high resolution front video
- an omni-directional video sensor developed by researchers from Riverside University (California)

The Dotmobil team chose RTMaps to perform data acquisition, prototype and integrate perception and data-fusion algorithms, and control the vehicle (road keeping with lateral and longitudinal control, obstacle detection and avoidance, road marking detections, etc.)

Figure 6 - The Dotmobilteam vehicle participating to the Darpa Urban Challenge

3.2 Connection with the Marilou Robotics simulator

Replaying pre-recorded data is interesting for prototyping perception and data-fusion algorithms, or for studying a system behaviour in post-processing replay. However, it remains difficult to work on control algorithms since it is not possible to react on the recorded data. To overcome this issue, we need to connect the development platform to a complete simulator which will simulate not only the sensor data streams but also the dynamics of the system, and which can receive commands and react to them accordingly. The use of RTMaps interconnected with a simulator offers a complete on-desktop development environment, and facilitates the porting of developments to the real system by just requiring the replacement of simulator interface components by sensors and actuators interface components.

An example has been developed with the Marilou Robotics Studio (http://www.anykode.com/index.php) which provides a complete robots simulator. RTMaps components have been developed to establish inter-process communication with the simulator and retrieve sensors data in real-time (like video streams from simulated cameras, ultrasonic belts, wheel coders, …) and send commands to robots actuators.

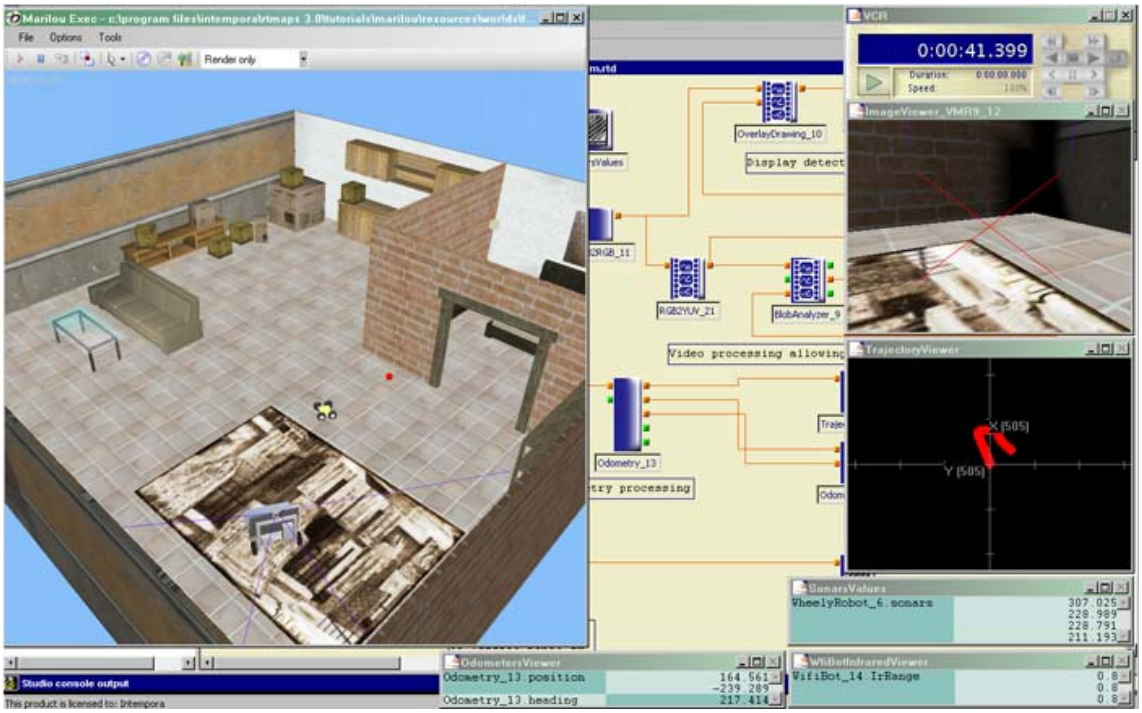

Figure 7 - Interface between RTMaps and the Marilou Robotics Studio simulator

3.3 The PUVAME project

The PUVAME project [4] (http://emotion.inrialpes.fr/puvame/) objective is to detect dangerous situations that may lead to collisions between pedestrians and public transport vehicles such as buses or tramways.

The developed system has to be able to detect dangerous situations thanks to the on-board sensors system and, if available, to match the information with infrastructure sensors systems such as infrastructure cameras that would be installed in dangerous areas such as crossroads or bus stops.

In order to be able to develop such a cooperative system between on-board systems and infrastructure systems, a Cycab from INRIA has been equipped with several sensors:

- a FireWire DCAM camera,
- a Sick LMS lidar,
- ultrasonic sensors (Microsonic Trans'o Prox)

whereas the experimentation site was equipped with 6 analog cameras installed on masts that could cover the entire site.

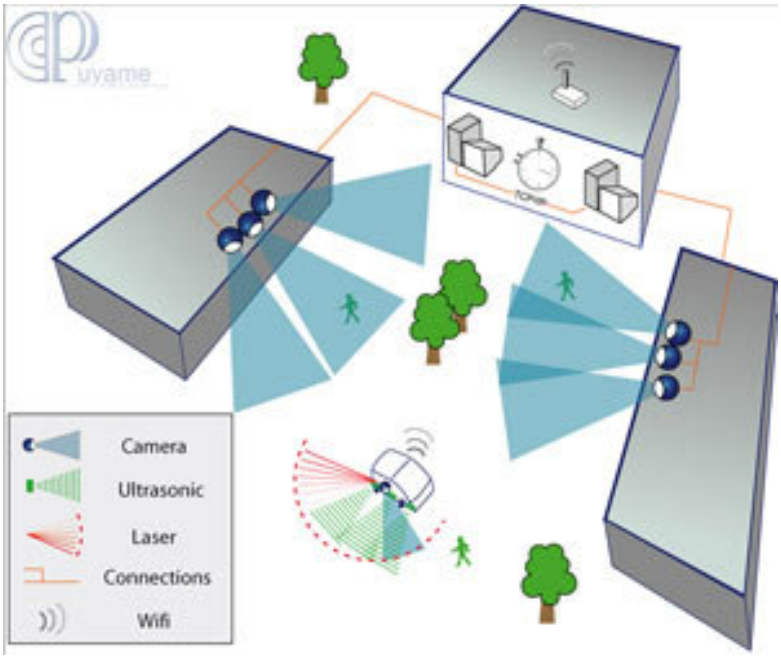

Figure 8 - The PUVAME experimentation system

Infrastructure system had to perform pedestrian detection, vehicle detection, and trajectory prevision of all the actors in order to compute a risk estimation.

The on-board system also performed its own risk estimation based on the on-board sensors informations. Based on the result of data fusion between the on-board system results and the infrastructure system results, reliable alerts could be delivered to the vehicle driver whenever dangerous situations occurred.

In order to be able to develop such a system in cooperation with different partners, there was a need to record the raw data in a synchronized way (both in vehicle and on the infrastructure) and to be able to playback the scenarios in order to develop the detection algorithms and the communication protocol.

In order to quickly setup such a distributed system, an RTMaps software has been installed in the 3 PCs (1 on-board, and 2 on the infrastructure). A Wifi connection has been setup between the vehicle and the infrastructure LAN, and the vehicle software has been set as Master whereas the infrastructure systems were setup as Slaves.

Figure 9 - The PUVAME playback diagram - synchronized playback of 6 infrastructure cameras and on-board sensorspresents the RTMaps diagram which performs the synchronized playback of a PUVAME scenario: 3 Player components are placed on a single diagram, each of which replays a databases that has been recorded on one of the 3 datalogger PCs.

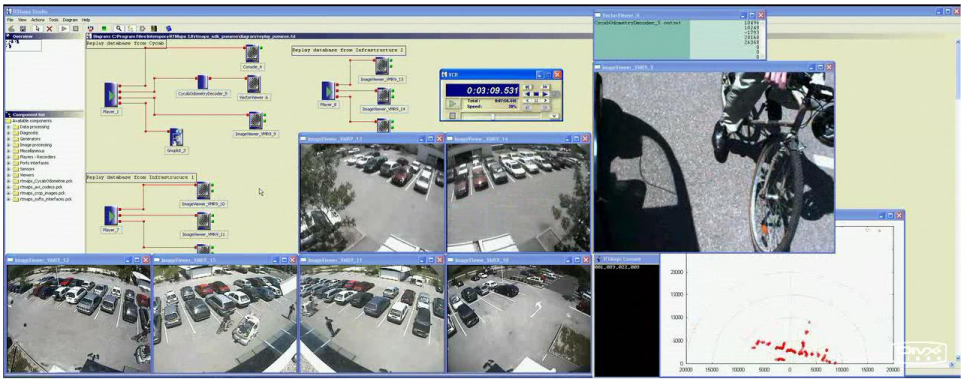

Figure 9 - The PUVAME playback diagram - synchronized playback of 6 infrastructure cameras and on-board sensors

4 CONCLUSION

In this paper, we have shortly presented RTMaps, a framework to be used for the development of real-time applications involving multiple sensors, as it is often the case in modern automotive and robotics applications. RTMaps is currently operational in many labs and industries, and is also often used as a cooperative tool to exchange and capitalize developments and data sets between teams.

It has been made with concerns of ease of use and modularity, but also performance and powerful unconstrained programming with in-depth configuration availability in order to meet the real-time constraints of applications from embedded use on small targets (such as autonomous vehicles and robots) to clusters of computers running distributed applications.

Its modular architecture also allows interfacing third party software tools (like simulators, or other control architectures) in order to take advantage of the best of each tool.

References

[1] Bruno Steux – « *RTMaps, a software environment for real-time applications development* » – PhD thesis at Ecole des Mines de Paris– Dec, 2001.

[2] G. Kahn, "*The semantics of a simple language for parallel programming*", in *Information Processing 74*: Proceedings of IFIP Congress 74, (Stockholm, Sweden) – 1974.

[3] CAR'07 – « *RTMaps V3.2 applied to distributed applications development* » - June 2007, G. Michel, N. du Lac, C. Delaunay.

[4] PUVAME – "*New French Approach for Vulnerable Road Users Safety*", Intelligent Vehicles Symposium, 2006 IEEE, Aycard, O. Spalanzani, A. Yguel, M. Burlet, J. Du Lac, N. De La Fortelle, A. Fraichard, T. Ghorayeb, H. Kais, M. Laugier, C. Laurgeau, C. Michel, G. Raulo, D. Steux, B.