Chapter Number

Task Management for Soft Real-Time Applications Based on General Purpose Operating Systems

Paulo Pedreiras and Luís Almeida
LSE-IEETA/DETI, University of Aveiro, Aveiro
Portugal

1. Introduction

Many embedded systems, such as those found in planes, trains, cars, robots and machine tools, exhibit specific timeliness, predictability and precedence constraints that must be respected. In many cases they are built on top of COTS microprocessor boards, possibly PCcompatibles, e.g., PC104, SBCs and Mini-ITX, and using multi-tasking operating systems or kernels, both real-time (RTOS) and general purpose (GPOS) despite the profound architectural and functional differences exhibited by these two classes of software infrastructures (Gopalan, 2001).
In fact, GPOS are typically time-shared multi-processing systems, optimized to manage heterogeneous classes of resources such as CPU, memory, disk, network interface, etc. The performance criteria for these systems are mostly associated with average throughput and fairness, the typical applications are not strictly time-constrained and may exhibit unpredictable blocking and execution times and activation latencies. Conversely, RTOS favor the timely execution of the activities they support. However the predictability delivered by this class of OSs comes at a price, which usually takes the form of additional information and constraints on the application tasks, e.g., bounded worst-case execution time, minimum inter-arrival time or activation period, relative phases and precedence constraints, and on operating system primitives, e.g., bounded blocking times, predictable synchronization primitives, suitable schedulers and admission control.
The architectural dichotomy presented by RTOS and GPOS leads to significant differences at the application implementation level, and thus the choice of using an RTOS or a GPOS may not be completely dictated by the timeliness and predictability aspects, only. For example, in soft real-time applications, which tolerate occasional failures in the time domain, GPOS may be preferred since they deliver sufficient real-time performance and generally outperform RTOS in practical aspects like hardware support, price, availability and diversity and quality of development tools.
Nevertheless, GPOS lack some features that are commonly required by embedded applications, e.g., support for automatic activation of recurrent tasks with enough precision, phase control and precedence constraints. These difficulties have been perceived in the scope of the CAMBADA middle-size robotic soccer team (Cambada), developed at the

This work was partially supported by the European Commission (NoE ARTIST2, IST-2 004527) and by IEETA - Universidade de Aveiro, Portugal

University of Aveiro, Portugal, and led to the development of a user-space simple process manager library, called PMan, which extends the native services provided by the underlying GPOS, Linux in this case. The PMan services are currently used to automatically trigger processes, adapt the Quality of Service (QoS) delivered to each process according to the operational environment, and to enforce phase and precedence constraints between distinct but related processes.

This chapter describes the PMan library implemented in Linux and presents results that confirm the usefulness of the services provided. It is organized in the following way: Section 2 discusses related work; Section 3 addresses the internals of the process manager layer (PMan); Section 4 presents a case study including practical experiments and Section 5 concludes the chapter.

2. Related work

Since the mid 90s that several attempts were made to achieve real-time performance with time-sharing general purpose operating systems. One approach that received substantial attention from the research community was the use of CPU reservations (Lee *et al*, 1996) (Jones *et al*, 1997) according to which a task could establish a contract with the CPU, reserving a given amount of time units every given period. These reservations would have priority over the time-sharing tasks. However, this model was not adequate to provide low jitter execution, could lead to large blocking unless a consistent system-wide reservation scheme was applied to all resources, and did not account for variable execution time. This aspect caused particular inefficiency in multimedia applications, thus (Chu and Nahrstedt, 1997) proposed supporting a new class of periodic but variable execution time tasks, which was completely built at user-level, thus highly portable, but required a kernel that could provide reserves.

This same idea of providing real-time support to applications running in user space was also the motivation for the development of the LXRT module in RTAI/Linux (LXRT). This module permits executing hard real-time tasks in user-space context, with a positive impact in the application development effort and in the system integrity, since programming errors caused by the real-time tasks do not jeopardize the overall Linux kernel sanity. This feature comes at a cost of degraded real-time performance, namely latency and jitter, particularly when Linux system calls are used by the tasks. The Xenomai project (Xenomai) presents several resemblances with RTAI/LXRT, however has a greater focus in facilitating the developers migration from RTOS to GNU/Linux based environments, by providing an emulation layer that supports diverse RT-APIs via skins, e.g., for VxWorks, POSIX and μITRON. Both of these approaches provide good timing performance but require adequate kernel level support that is not provided by the Linux kernel alone.

A different path was followed by Chu and Nahrstedt (1997) in which soft real-time operation was achieved with a simple user-space scheduler based on the fixed priorities defined within POSIX.4. We also follow this approach, providing a user-space scheduler that can simply be executed as a normal application in a Linux GPOS, without need for kernel patches or specialized kernels. With respect to (Chu and Nahrstedt, 1997), we take a step further adding support for off-sets and precedence constraints.

3. The process manager layer - PMan

The core of our proposal is the processor manager layer, called PMan, which aims at facilitating the development of soft real-time applications, extending the native services provided by the underlying GPOS in the following aspects:

- automatic activation of recurrent tasks;
- settling of relative phase control, allowing to establish temporal offsets among tasks;
- precedence constraints, conditioning the release of processes to the conclusion of a set of predecessors;
- on-line process management and QoS adaptation, allowing adding and removing processes at run-time as well as changing dynamically the temporal properties of the executing ones, without service disruption.

The time management within PMan is associated to a periodic tick whose source is userconfigurable and can be generated with a timer or an external event. For example, in the CAMABADA project the PMan tick is associated with the arrival of image frames, in order to minimize the latency between the image acquisition and the activation of the related processing tasks.

The PMan operation relies on certain data concerning the processes, which is kept within the PMan table. A process record in this table is shown in Table 1. The process name and process pid fields allow a proper process identification, which is used to associate a table entry with a particular process and to send OS signals to the processes, respectively. The period and phase fields are used to trigger the processes at adequate instants. The period is expressed in number of **PMan ticks**, allowing each process to be triggered every n ticks. The phase and delay fields permit de-phasing the processes activation, for example to balance the CPU load over time, with potential benefits in terms of process activation jitter. The deadline field supports a basic reflection mechanism permitting the process, when necessary, to carry out sanity checks or recovery actions in case of process misbehavior. A user specified process is automatically activated upon occurrence of a deadline miss event. The following section of the PMan process record is devoted to the recollection of statistical data, which can be useful for profiling purposes. Finally, the status field keeps track of the current process state.

The library of services associated to the PMan layer is summarized in Table 2. The layer is initialized via the **PMAN_init** service and terminated with **PMAN_close**. The process registration in the PMan table is carried out with **PMAN_procadd**. After registering it is necessary to bind the process OS pid using **PMAN_attach**. This separation allows having a process registering itself autonomously or having a third party managing the registration and properties of other processes. Process entries may be removed from the PMan table by calling **PMAN_procdel**. **PMan_detach** dissociates a process from a PMan table entry.

Process identification		
	PROC_name	Process ID string
	PROC_pid	OS process ID
Generic temporal properties		
	PROC_period	Period (ticks)
	PROC_phase	Phase (ticks)
	PROC_delay	Execution delay (ms)
	PROC_deadline	Deadline (μs)
Precedence constraints		
	PROC_pred_name	Predecessor list
QoS management		
	PROC_qosdata	QoS attributes
	PROC_qosupdateflag	QoS change flag
Statistical data		
	PROC_laststart	Activation instant of last instance
	PROC_lastfinish	Finish instant of last instance
	PROC_nact	Number of activations
	PROC_ndm	Number of deadline misses
Process status		
	PROC_status	Process status

Table 1. PMan process record

Attaching/detaching processes can be carried out online to allow, for example, selecting one

PMAN_init	allocates resources (shared memory, semaphores, etc.) and initializes the PMan data structures
PMAN_close	releases resources used by Pman
PMAN_procadd	register a process in the PMan table
PMAN_procdel	removes a process from the PMan table
PMAN_attach	attaches the OS process id to an already registered process, completing the registration phase
PMAN_detach	clears the process id field from a PMan record
PMAN_prec_add	specifies a precedence constraint between to processes
PMAN_prec_rem	removes the precedence constraint between two processes
PMAN_QoSupd	changes the QoS attributes of a process already registered in the PMan table
PMAN_TPupd	changes the temporal properties (period, phase, deadline or delay) of a process already registered in the PMan table
PMAN_epilogue	signals that a process has terminated the execution of one instance
PMAN_query	allows to retrieve statistical information about one process
PMAN_tick	called upon occurrence of a tick event, incrementing time within PMan and triggering the activation of processes

Table 2. PMan services

from a set of processes to carry out a given action according a desired cost function, i.e. implementing functional alternatives that can be useful during CPU overloads.
A specific feature of PMan is the support for precedence constraints among processes. For example, in a classical sampling-controller-actuation loop it is necessary to guarantee that these functions are executed strictly in this order to have the end-to-end latency minimized. With the recent advances in computing hardware, e.g., hyper-threading and multi-core CPUs, simpler techniques such as those based on fixed priorities are no longer enough to enforce the right sequencing, being necessary to use explicit synchronization primitives.

These ones become hard to use in non-trivial situations with many-to-many dependencies between processes or when the set of processes and, consequently, their precedence relations, vary dynamically. To cope with this difficulty the PMan library manages automatically the precedence constraints among processes. Applications declare precedence relationships using **PMAN_precadd** and, conversely, **PMAN_precdel** to remove previously established relationships. PMan checks all related precedences before actually releasing a process. The status of precedence constraints is updated whenever processes terminate.

The **PMAN_QoSupd** call allows changing the QoS allocated to each process at runtime. The QoS attributes depend on the underlying OS. The current implementation over Linux considers the OS priority as a QoS parameter. The application processes are assigned realtime priorities, with SCHED_FIFO scheduler, via the **sched_setscheduler** system call. The process priority, supplied as argument to **PMAN_QoSupd**, must be within the range [sched_get_priority_min, sched_get_priority_max]. Similarly, the temporal properties of one process can also be updated dynamically using **PMAN_TPupd**. The ability to change the QoS of processes at runtime is particularly useful when the environment is highly variable and/or hard to characterize, allowing the application to dynamically adapt itself, allocating more resources to the processes that, in each instant, have higher impact on the global performance.

The **PMAN_epilogue** call must be issued by every process managed by PMan, just before termination. This service is required for internal PMan management, namely verification of deadline violations and updating precedence constraints. **PMAN_query**, on the other hand, allows accessing the statistical data of each registered process, which can be useful, for example, for profiling and load management. Finally, **PMAN_tick** carries out temporal management in PMan, incrementing the tick count, activating processes, checking task deadlines, etc. This service must be requested periodically either with a system timer or with an external event. The latter mode, which is not commonly found, supports a transparent synchronization of the application execution with an external event stream, such as the arrival of image frames from a camera with automatic image capture.

4. Process synchronization with PMan: a case study

4.1 The CAMBADA vision subsystem architecture

As stated above, the PMan library was developed to address some difficulties perceived in the scope of the CAMBADA RoboCup middle-size robotic soccer team (Cambada). Currently, the vision subsystem architecture (Fig. 1) uses one catadioptric configuration implemented with a low cost Fire-wire web-camera (BCL 1.2 Unibrain camera) and a hyperbolic mirror. The camera delivers 640x480 YUV images at 30 frames per second. When a new frame becomes available, the image handling process is automatically triggered and the frame is placed in a shared memory buffer. The Color_Class[i] set of processes (Fig. 1) will then analyze the acquired image for color classification, creating a new one with color labels, i.e., an 8 bit per pixel image. This image is also placed in a shared image buffer, which is afterwards analyzed by the Obj_track[i] object detection processes (Fig. 1). The output of the detection processes is placed in the real-time database (RTDB) which can be accessed by the other processes on the system, such as the control action and world state update.

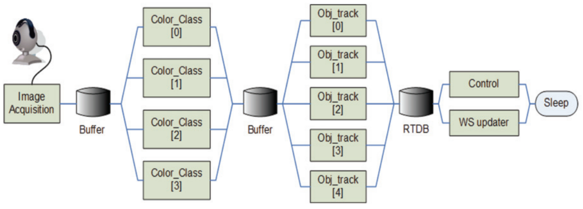

Fig. 1. CAMBADA vision subsystem architecture

4.2 Motivation for explicit precedences and relative offsets

The activation of the image-handling processes is carried out by the PMan manager right after the arrival of each new image frame. In earlier versions (Pedreiras *etal.* 2006; 2007) the PMan triggered sets of related tasks simultaneously, using priorities to enforce precedence constraints. This approach worked well with a single CPU and in the absence of hyperthreading but it failed to enforce such constraints when the computing platform of the robots was updated to Intel™ Core2Duo™ processors with two CPUs and hyper-threading. This is a common problem in real-time applications, which depend on specific features of the underlying hardware platform. When the platform is replaced, previous assumptions may fail leading to a poor application performance and possibly to a system failure. Therefore, the **PMAN_precadd** and **PMAN_precdel** primitives (Table 2) were added to the PMan library to deal with precedence constraints explicitly, as referred in Section 3.
Another identified problem was the need to adjust the control parameters individually, for each robot, due to differences in CPU processing power and thus differences in end-to-end image handling latencies. Moreover, the highly variable nature of the execution time of image processing activities further complicated the controller tuning. To solve these problems the actual release instant of the control process was decoupled from the termination of the preceding object tracking processes and it was set with a predetermined offset with respect to the image reception. This technique substantially reduces the dependence of the control performance on the underlying hardware used and it was added to the current version of PMan.

4.3 Experimental results

To experimentally verify the implementation and assess the performance of the PMan library several experiments were carried out using the CAMBADA architecture depicted in Fig. 1. Table 3 shows the process set used in the experiments.

Process	Period	Precedence list	Delay (ms)	Description
readFireWire	1	{}	0	Interface with camera. Issues the tick event.
Color_Class[0..3]	1	readFireWire	0	Color classification. Each process scans 1/4th of the image.
Obj_Track[0..4]	1	Color_Class[0..3]	0	Object tracking (ball, obstacles, blue goal, yellow goal, lines).
Control	1	Obj_Track[0..4]	20	Issues control commands.
WSUpdater	1	Obj_Track[0..4]	20	Updates the RTDB with the processed data.

Table 3. Experimental process set

Two hardware platforms were used, one based on an Intel Pentium M Processor at 1.6GHz (Asus A3N notebook PC) and another based on an Intel P4 DualCore at 2.6GHz (Asus Pundit desktop PC). Both run the Linux 2.6.22 kernel, with the timer frequency set to 1000 Hz and the High Resolution Timer Support enabled. The first experiment aimed at assessing the overhead induced by the PMan layer, namely by executing the PMAN_tick service. Table 4 shows the latencies measured from the activation of the *readFireWire* process, which calls PMAN_tick, to the start of execution of Color_Class[0], the process that executes right after. Two different scenarios have been considered. The first scenario (Immediate) corresponds to a slight modification of Table 3 in which the Delay parameter is made equal to 0 for all processes. This means that all processes execute as soon as possible, i.e., when the CPU is available and the precedence constraints are met. The other scenario (Deferred) corresponds to the parameters specified Table 3. Both scenarios differ only in the activation of the Control and WSUpdater processes, which is deferred by 20ms after the trigger in the latter case.

	Control and WSUpdater activ.	Max. (µs)	Min. (µs)	Avg. (µs)	St.dev. (µs)
PM, 1.6GHz	Immediate	40	5	12.56	4.61
	Deferred	93	15	34.30	10.75
P4 Dual Core, 2.6GHz	Immediate	35	8	10.77	1.68
	Deferred	42	19	24.18	1.78

Table 4. Upper bound on the execution time of **PMAN_tick**

The measured latencies vary between 5µs and 93µs in the notebook platform and between 8µs and 42µs in the desktop platform. As expected, the average values are lower for the desktop PC due to the higher CPU processing power. It should also be remarked that the deferred execution incurs an additional execution penalty, resulting from the need to create a wake-up thread and an extra call to the nanosleep primitive. Nevertheless, in any of the scenarios analyzed the additional overhead and latency induced by the PMAN_tick event are negligible in face of the PMan tick period considered (33ms).
Table 5 shows the results of a second experiment, aiming at verifying the effectiveness of the deferred activations, namely those of the *Control* process. When the activation is immediate, i.e. with *Delay=0*, the activation latency is highly variable (with the notebook) and hardware dependent, reaching an absolute jitter of almost 18ms in the worst-case, which is over 50% of the sampling period, a non-negligible value from the control performance point of view. With deferred activation, i.e. *Delay=20ms*, the absolute jitter is substantially reduced, being below 3.5ms for the notebook and below 35🕐s for the desktop PC. Note that in the experiments the precedence constraints are always enforced and thus part of the jitter observed in the notebook platform results from the predecessor tasks requiring more than 20ms to complete thus pushing the activation of the control process.

	Control process activation	Max. (µs)	Min. (µs)	Avg. (µs)	St.dev. (µs)
PM, 1.6GHz	Immediate	23143	5810	15448	5592
	Deferred	23465	20037	20224	414
P4 Dual Core 2.6 GHz	Immediate	5963	2168	3296	573
	Deferred	20078	20046	20054	3

Table 5. Control process activation delay

As expected, enforcing a deferral in the activation of processes that are subject to precedence constraints can have a noticeable impact on the respective regularity practically eliminating the respective jitter. Furthermore, the activation latency and associated jitter become nearly hardware independent. This increased execution predictability facilitates the tuning of feedback controllers, such as those used within the Control process, resulting in improved control performance.
Fig. 2. presents a histogram of the control process activation latencies both with deferred and immediate execution in the notebook platform. The reduction in the jitter figures is clear. With the deferred execution near 90% of the process activation latencies occur within a vicinity of 1ms of the desired value while with immediate execution the activation pattern is substantially enlarged, with latencies ranging from near 6ms to around 23ms and two clear peaks. It should be remarked that with immediate execution the start time of the control process depends solely on the completion of its predecessors and that this instant depends on the amount of processing they required. This amount depends on the richness of the images, i.e., on the number of regions that have to be checked, a parameter that depends on the environment (colors of the objects surrounding the playfield, illumination intensity and nature, etc.) and is, to a large extent, unpredictable and highly dynamic. Therefore, to allow a fair comparison among the diverse scenarios a fixed synthetic workload was generated and used throughout the experiments.
Fig. 3 presents a histogram similar to that in Fig. 2 but referring to the desktop platform. The major distinction between the results with both platforms is the strong reduction in the jitter achieved with the desktop PC caused by its substantially higher processing power. Particularly, it is worth noticing the high accuracy with which the control process is triggered in the desktop PC with deferred execution. When this process is triggered the precedence constraints are always already met and thus the residual jitter is due to the handling of OS events, only. This behavior is confirmed quantitatively by inspecting Table 5. The control process activation for the desktop platform varies between 20078🕛s and 20046🕛s, with a standard deviation of 3 🕛s.

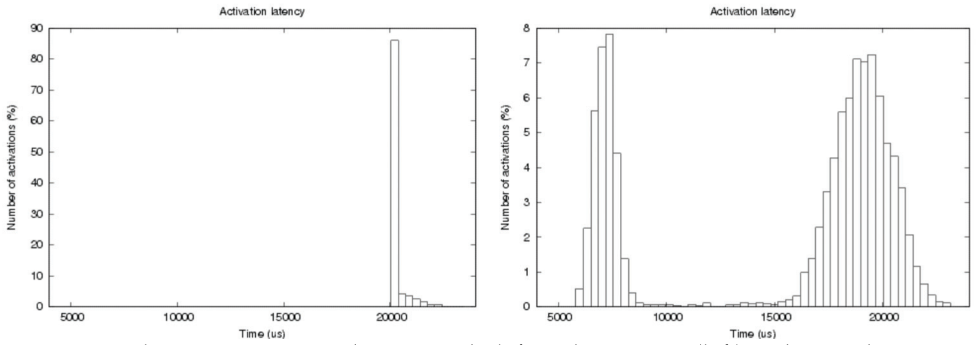

Fig. 2. Control process activation latency with deferred execution (left) and immediate execution (rigth), notebook platform

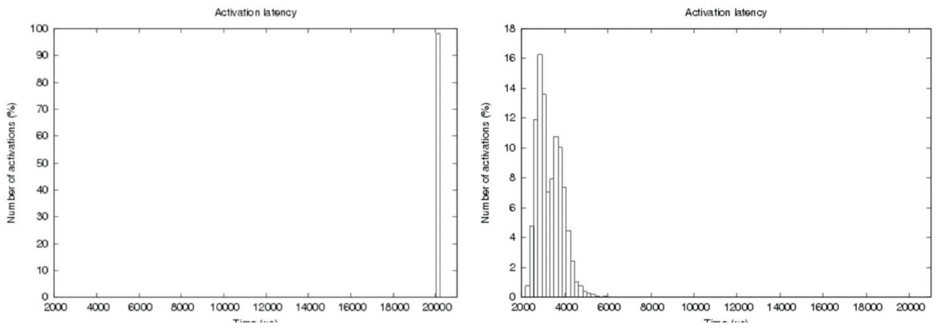

Fig. 3. Control process activation latency with deferred execution (left) and immediate execution (right), desktop platform

Fig. 4 shows an excerpt of an execution timeline in the notebook PC. Process activations are indicated by small circles. The four *Color classification* processes and the five *Object tracking* processes execute in sequence, inheriting the execution jitter of their predecessors. However, the *Control* and *World State update* processes, on top, have a deferred activation that absorbs the execution jitter of the predecessors, resulting in a higher activation regularity. Notice, nevertheless, that these processes also have precedence constraints, which are always enforced, even if the execution of the predecessors takes longer than the specified deferral delay.

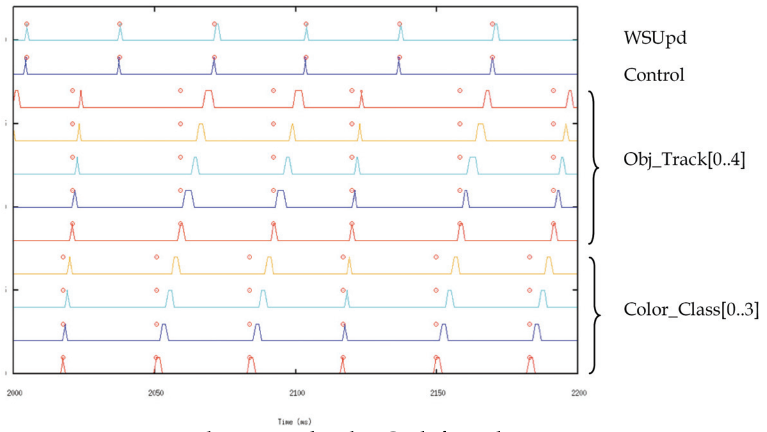

Fig. 4. Process execution timeline. Notebook PC, deferred execution

5. Conclusions

Using general purpose operating systems for soft real time applications has several advantages related with low costs and the abundance of device drivers and software tools. However, such applications still require adequate timing services, for process activation and synchronization.

In this chapter we presented a process management library that provides such services with substantial hardware independence and executes completely within user-space, being thus very flexible to deploy and use. In particular, this library provides support for periodic process activations, possibly with relative offsets and explicit precedence constraints, and also dynamic adaptation of temporal parameters and QoS attributes.
The library was developed within the scope of the CAMBADA RoboCup middle-size soccer robots. Using this application as a case study, the chapter presents several practical experiments that show the low overhead induced by the process management structure and the effectiveness of its support for precedence constraints and relative offsets with hardware independence.

6. References

Chu, H. & Nahrstedt, K. (1997). A soft real time scheduling server in UNIX operating system. *Proceedings of European Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*. Also In *Lecture Notes In Computer Science*, pp. 153-162, ISBN:3-540-63519-X , Darmstadt, Germany, September 1997.

Chu, H. & Nahrstedt, K. (1999). CPU service classes for multimedia applications. *Proceedings of IEEE Conference on Multimedia Computing and Systems* (MCS'99), pp. 296-301, ISBN: 0-7695-0253-9,. Florence, Italy, June 1999.

Gopalan, K. (2001). Real-Time Support in General Purpose Operating Systems. *ECSL Technical Report TR92, Experimental Computer Systems Lab, Computer Science Dept,* WSUpd Control Color_Class[0..3] Obj_Track[0..4] *Stony Brook University, Stony Brook, NY - 11794-4400.*

Jones, M. B.; Rosu, D. & Rosu, M. (1997). CPU reservation and time constraints: Efficient, predictable scheduling of independent activities. *Proceedings of 16th ACM Symp. On Operating Systems Principles* (SOSP'97), St. Malo, France, October 1997.

Lee, C.; Rajkumar, R. & Mercer, C. (1996). Experience with CPU reservation and dynamic QoS in real-time Mach. *Multimedia Japan*, March 1996.

Pedreiras, P.; Teixeira, F.; Ferreira, N.; Almeida, L.; Pinho, A. & Santos, F. (2006). Enhancing the reactivity of the vision subsystem in autonomous mobile robots using real-time techniques. RoboCup Symposium: Papers and Team Description Papers, RoboCup-2005: Robot Soccer World Cup IX, Lecture Notes in Artificial Intelligence, Bredenfeld, A.; Jacoff, A.; Noda, I.; Takahashi, Y. (Eds.) pp. 371-383, ISBN: 3-540-35437-9 , Springer Berlin / Heidelberg , Springer, 2006.

Pedreiras, P.; Teixeira, F., Ferreira, N.; Almeida, L.; Pinho, A. & Santos, F. (2007). A Real Time Framework for the Vision Subsystem in Autonomous Mobile Robots. In *Vision Systems Applications,* G. Obinata and A. Dutta (Eds.), pp.83—100, ARS, ISBN 978-3-902613-01-1, Vienna, Austria.

Cambada. Cooperative Autonomous Mobile Robots with Advanced Distributed Architecture. University of Aveiro, Portugal. Online, http://www.ieeta.pt/atri/cambada/

LXRT. Online, http://www.fdn.fr/~brouchou/rtai/rtai-doc-prj/doxyapi/group__lxrt.html Xenomai. http://www.xenomai.org/index.php/Main_Page