We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

6

A Real-Time Framework for the Vision Subsystem in Autonomous Mobile Robots

Paulo Pedreiras[1], Filipe Teixeira[2], Nelson Ferreira[2], Luís Almeida[1], Armando Pinho[1] and Frederico Santos[3]
[1]*LSE-IEETA/DETI, Universidade de Aveiro, Aveiro*
[2]*DETI, Universidade de Aveiro, Aveiro*
[3]*DEE, Instituto Politécnico de Coimbra, Coimbra*
Portugal

1. Introduction

Interest on using mobile autonomous agents has been growing (Weiss, G., 2000), (K. Kitano; Asada, M.; Kuniyoshi, Y.; Noda, I. & Osawa E., 1997) due to their capacity to gather information on their operating environment in diverse situations, from rescue to demining and security. In many of these applications, the environments are inherently unstructured and dynamic, and the agents depend mostly on visual information to perceive and interact with the environment. In this scope, computer vision in a broad sense can be considered as the key technology for deploying systems with an higher degree of autonomy, since it is the basis for activities like object recognition, navigation and object tracking.

Gathering information from such type of environments through visual perception is an extremely processor-demanding activity with hard to predict execution times (Davison, J., 2005). To further complicate the situation many of the activities carried out by the mobile agents are subject to real-time requirements with different levels of criticality, importance and dynamics. For instance, the capability to timely detect obstacles near the agent is a hard activity, since failures can result in injured people or damaged equipment, while activities like self-localization, although important for the agent performance, are inherently soft since extra delays in these activities simply cause performance degradation. Therefore, the capability to timely process the image at rates high enough to allow visual-guided control or decision-making, called real-time computer vision (RTCV) (Blake, A; Curwen, R. & Zisserman, A., 1993), plays a crucial role in the performance of mobile autonomous agents operating in open and dynamic environments.

This chapter describes a new architectural solution for the vision subsystem of mobile autonomous agents that substantially improves its reactivity by dynamically assigning computational resources to the most important tasks. The vision-processing activities are broken into separated elementary real-time tasks, which are then associated with adequate real-time properties (e.g. priority, activation rate, precedence constraints). This separation allows avoiding the blocking of higher priority tasks by lower priority ones as well as to set independent activation rates, related with the dynamics of the features or objects being processed, together with offsets that de-phase the activation instants of the tasks to further

Source: Vision Systems: Applications, ISBN 978-3-902613-01-1
Edited by: Goro Obinata and Ashish Dutta, pp. 608, I-Tech, Vienna, Austria, June 2007

reduce mutual interference. As a consequence it becomes possible to guarantee the execution of critical activities and privilege the execution of others that, despite not critical, have large impact on the robot performance.

The framework herein described is supported by three custom services:

- Shared Data Buffer (SDB), allowing different processes to process in parallel a set of image buffers;
- Process Manager (PMan), which carries out the activation of the vision-dependent real-time tasks;
- Quality of Service manager (QoS), which dynamically updates the real-time properties of the tasks.

The SDB service keeps track of the number of processes that are connected to each image buffer. Buffers may be updated only when there are no processes attached to them, thus ensuring that processes have consistent data independently of the time required to complete the image analysis.

The process activation is carried out by a PMan service that keeps, in a database, the process properties, e.g. priority, period and phase. For each new image frame, the process manager scans the database, identifies which processes should be activated and sends them wake-up signals. This framework allows reducing the image processing latency, since processes are activated immediately upon the arrival of new images. Standard OS services are used to implement preemption among tasks.

The QoS manager monitors continuously the input data and updates the real-time properties (e.g. the activation rate) of the real-time tasks. This service permits to adapt the computational resources granted to each task, assuring that in each instant the most important ones, i.e. the ones that have a greater value for the particular task being carried out, receive the best possible QoS.

The performance of the real-time framework herein described is assessed in the scope of the CAMBADA middle-size robotic soccer team, being developed at the University of Aveiro, Portugal, and its effectiveness is experimentally proven.

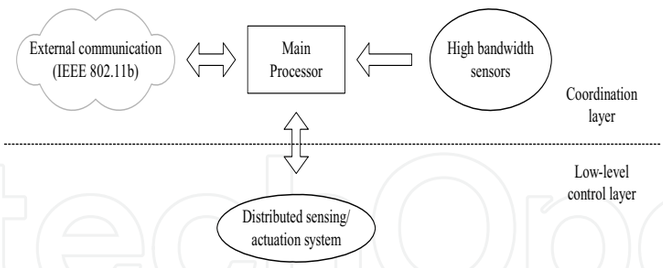

Figure 1. The biomorphic architecture of the CAMBADA robotic agents

The remainder of this chapter is structured as follows: Section 2 presents the generic computing architecture of the CAMBADA robots. Section 3 shortly describes the working-principles of the vision-based modules and their initial implementation in the CAMABADA robots. Section 4 describes the new modular architecture that has been devised to enhance the temporal behavior of the image-processing activities. Section 5 presents experimental results and assesses the benefits of the new architecture. Finally, Section 6 concludes the chapter.

2. The CAMBADA Computing Architecture

2.1 Background

Coordinating several autonomous mobile robotic agents in order to achieve a common goal is currently a topic of intense research (Weiss, G., 2000), (K. Kitano; Asada, M.; Kuniyoshi, Y.; Noda, I. & Osawa E., 1997). One initiative to promote research in this field is RoboCup (K. Kitano; Asada, M.; Kuniyoshi, Y.; Noda, I. & Osawa E., 1997), a competition where teams of autonomous robots have to play soccer matches.

As for many real-world applications, robotic soccer players are autonomous mobile agents that must be able to navigate in and interact with their environment, potentially cooperating with each other. The RoboCup soccer playfield resembles human soccer playfields, though with some (passive) elements specifically devoted to facilitate the robots navigation. In particular the goals have solid and distinct colors and color-keyed posts are placed in each field corner. This type of environment can be classified as a passive information space (Gibson, J., 1979). Within an environment exhibiting such characteristics, robotic agents are constrained to rely heavily on visual information to carry out most of the necessary activities, leading to a framework in which the vision subsystem becomes an integral part of the close-loop control. In these circumstances the temporal properties of the image-processing activities (e.g. period, jitter and latency) have a strong impact on the overall system performance.

2.2 The CAMBADA robots computing architecture

The computing architecture of the robotic agents follows the biomorphic paradigm (Assad, C.; Hartmann, M. & Lewis, M., 2001), being centered on a main processing unit (the brain) that is responsible for the higher-level behavior coordination (Figure 1). This main processing unit handles external communication with other agents and has high bandwidth sensors (the vision) directly attached to it. Finally, this unit receives low bandwidth sensing information and sends actuating commands to control the robot attitude by means of a distributed low-level sensing/actuating system (the nervous system).

The main processing unit is currently implemented on a PC-based computer that delivers enough raw computing power and offers standard interfaces to connect to other systems, namely USB. The PC runs the Linux operating system over the RTAI (Real-Time Applications Interface (RTAI, 2007)) kernel, which provides time-related services, namely periodic activation of processes, time-stamping and temporal synchronization.

The agents software architecture is developed around the concept of a real-time database (RTDB), i.e., a distributed entity that contains local images (with local access) of both local and remote time-sensitive objects with the associated temporal validity status. The local images of remote objects are automatically updated by an adaptive TDMA transmission control protocol (Santos, F.; Almeida, L.; Pedreiras, P.; Lopes, S. & Facchinnetti, T., 2004) based on IEEE 802.11b that reduces the probability of transmission collisions between team mates thus reducing the communication latency.

The low-level sensing/actuating system follows the fine-grain distributed model (Kopetz, H., 1997) where most of the elementary functions, e.g. basic reactive behaviors and closed-loop control of complex actuators, are encapsulated in small microcontroller-based nodes, interconnected by means of a network. This architecture, which is typical for example in the automotive industry, favors important properties such as scalability, to allow the future addition of nodes with new functionalities, composability, to allow building a complex

system by putting together well defined subsystems, and dependability, by using nodes to ease the definition of error-containment regions. This architecture relies strongly on the network, which must support real-time communication. For this purpose, it uses the CAN (Controller Area Network) protocol (CAN, 1992), which has a deterministic medium access control, a good bandwidth efficiency with small packets and a high resilience to external interferences. Currently, the interconnection between CAN and the PC is carried out by means of a gateway, either through a serial port operating at 115Kbaud or through a serial-to-USB adapter.

3. The CAMBADA Vision Subsystem

The CAMBADA robots sense the world essentially using two low-cost webcam-type cameras, one facing forward, and the other pointing the floor, both equipped with wide-angular lenses (approximately 106 degrees) and installed at approximately 80cm above the floor. Both cameras are set to deliver 320x240 YUV images at a rate of 20 frames per second. They may also be configured to deliver higher resolution video frames (640x480), but at a slower rate (typically 10-15 fps). The possible combinations between resolution and frame-rate are restricted by the transfer rate allowed by the PC USB interface.

The camera that faces forward is used to track the ball at medium and far distances, as well as the goals, corner posts and obstacles (e.g. other robots). The other camera, which is pointing the floor, serves the purpose of local omni-directional vision and is used for mainly for detecting close obstacles, field lines and the ball when it is in the vicinity of the robot. Roughly, this omni-directional vision has a range of about one meter around the robot.

All the objects of interest are detected using simple color-based analysis, applied in a color space obtained from the YUV space by computing phases and modules in the UV plane. We call this color space the YMP space, where the Y component is the same as in YUV, the M component is the module and the P component is the phase in the UV plane. Each object (e.g., the ball, the blue goal, etc.) is searched independently of the other objects. If known, the last position of the object is used as the starting point for its search. If not known, the center of the frame is used. The objects are found using region-growing techniques. Basically, two queues of pixels are maintained, one used for candidate pixels, the other used for expanding the object. Several validations can be associated to each object, such as minimum and maximum sizes, surrounding colors, etc.

Two different Linux processes, Frontvision and Omnivision, handle the image frames associated with each camera. These processes are very similar except for the specific objects that are tracked. Figure 2 illustrates the actions carried out by the Frontvision process. Upon system start-up, the process reads the configuration files from disk to collect data regarding the camera configuration (e.g. white balance, frames-per-second, resolution) as well as object characterization (e.g. color, size, validation method). This information is then used to initialize the camera and other data structures, including buffer memory. Afterwards the process enters in the processing loop. Each new image is sequentially scanned for the presence of the ball, obstacles, goals and posts. At the end of the loop, information regarding the diverse objects is placed in a real-time database.

The keyboard, mouse and the video framebuffer are accessed via the Simple DirectMedia Layer library (SDL) (SDL, 2007). At the end of each loop the keyboard is pooled for the presence of events, which allows e.g. to quit or dynamically change some operational parameters

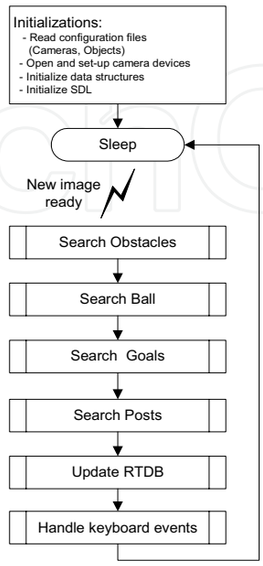

Figure 2. Flowchart of the Frontvision process

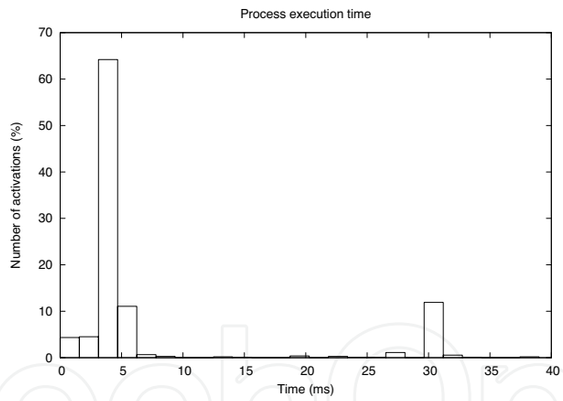

Figure 3. Ball tracking execution time histogram

4. A Modular Architecture for Image Processing: Why and How

As referred to in the previous sections, the CAMBADA robotic soccer players operate in a dynamic and passive information space, depending mostly on visual information to perceive and interact with the environment. However, gathering information from such type of environments is an extremely processing-demanding activity (DeSouza, G & Kak, A., 2004), with hard to predict execution times. Regarding the algorithms described in Section 3, it could be intuitively expected to observe a considerable variance in process

execution times since in some cases the objects may be found almost immediately, when their position between successive images does not change significantly, or it may be necessary to explore the whole image and expand a substantial amount of regions of interest, e.g. when the object disappears from the robot field of vision (Davison, J., 2005). This expectation is in fact confirmed in reality, as depicted in Figure 3, which presents a histogram of the execution time of the ball tracking alone. Frequently the ball is located almost immediately, with 76.1% of the instances taking less than 5ms to complete. However, a significant amount of instances (13.9%) require between 25ms and 35ms to complete and the maximum observed execution time was 38,752 ms, which represents 77.5% of the inter-frame period just to process a single object.

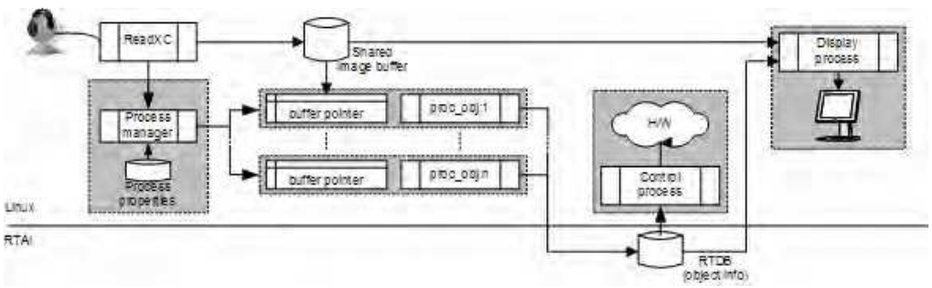

Figure 4. Modular software architecture for the CAMBADA vision subsystem

As described in Section 3, the CAMBADA vision subsystem architecture is monolithic with respect to each camera, with all the image-processing carried out within two processes designated Frontvision and Omnivision, associated with the frontal and omnidirectional cameras, respectively. Each of these processes tracks several objects sequentially. Thus, the following frame is acquired and analyzed only after tracking all objects in the previous one, which may take, in the worst case, hundreds of milliseconds, causing a certain number of consecutive frames to be skipped. These are vacant samples for the robot controllers that degrade the respective performance and, worse, correspond to black-out periods in which the robot does not react to the environment. Considering that, as discussed in Section 3, some activities may have hard deadlines, this situation becomes clearly unacceptable. Increasing the available processing power, either trough the use of more powerful CPUs or via specialized co-processor hardware could, to some extent, alleviate the situation (Hirai, S.; Zakouji, M & Tsuboi, T., 2003). However, the robots are autonomous and operate from batteries, and thus energy consumption aspects as well as efficiency in resource utilization render brut-force approaches undesirable.

4.1 Using Real-Time Techniques to Manage the Image Processing

As remarked in Section 1, some of the activities carried out by the robots exhibit real-time characteristics with different levels of criticality, importance and dynamics. For example, the latency of obstacle detection limits the robots maximum speed in order to avoid collisions with the playfield walls. Thus, the obstacle detection process should be executed as soon as possible, in every image frame, to allow the robot to move as fast as possible in a safe way. On the other hand, detecting the corner poles for localization is less demanding and can span across several frames because the robot velocity is limited and thus, if the localization

process takes a couple of frames to execute its output is still meaningful. Furthermore prediction methods (Iannizzotto, G., La Rosa, F. & Lo Bello, L., 2004) combined with odometry data may also be effectively used to obtain estimates of object positions between updates. Another aspect to consider is that the pole localization activity should not block the more frequent obstacle detection. This set of requirements calls for the encapsulation of each object tracking activity in different processes as well as for the use of preemption and appropriate scheduling policies, giving higher priority to most stringent processes. These are basically the techniques that were applied to the CAMBADA vision subsystem as described in the following section.

4.2 A Modular Software Architecture

Figure 4 describes the software modular architecture adopted for the CAMBADA vision subsystem. Standard Linux services are used to implement priority scheduling, preemption and data sharing.

Associated to each camera there is one process (ReadXC) which transfers the image frame data to a shared memory region where the image frames are stored. The availability of a new image is fed to a process manager, which activates the object detection processes. Each object detection process (e.g. obstacle, ball), generically designated by proc_obj:x, x={1,2,…n} in Figure 4, is triggered according to the attributes (period, phase) stored in a process database. Once started, each process gets a link to the most recent image frame available and starts tracking the respective object. Once finished, the resulting information (e.g. object detected or not, position, degree of confidence, etc.) is placed in a real-time database (Almeida, L.; Santos, F.; Facchinetti; Pedreiras, P.; Silva, V. & Lopes, L., 2004), identified by the label "Object info", similarly located in a shared memory region. This database may be accessed by any other processes on the system, e.g. to carry out control actions. A display process may also be executed, which is useful mainly for debugging purposes.

4.2.1 Process Manager

For process management a custom library called PMan was developed. This library keeps a database where the relevant process properties are stored. For each new image frame, the process manager scans the database, identifies which processes should be activated and sends them pre-defined wake-up signals.

Table 1 shows the information about each process that is stored in the PMan database.

The process name and process pid fields allow a proper process identification, being used to associate each field with a 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 frames, allowing each process to be triggered every n frames. The phase field permits de-phasing the process activations in order to balance the CPU load over time, with potential benefits in terms of process jitter. The deadline field is optional and permits, when necessary, to carry out sanity checks regarding critical processes, e.g. if the high-priority obstacle detection does not finish within a given amount of time appropriate actions may be required to avoid jeopardizing the integrity of the robot. The following section of the PMan table is devoted to the recollection of statistical data, useful for profiling purposes. Finally, the status field keeps track of the instantaneous process state (idle, executing).

Process identification		
	PROC_name	Process ID string
	PROC_pid	Process id
Generic temporal properties		
	PROC_period	Period (frames)
	PROC_phase	Phase (frames)
	PROC_deadline	Deadline (μs)
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 data summary

The PMan services are accessed by the following API:

- **PMAN_init**: allocates resources (shared memory, semaphores, etc) and initializes the PMan data structures;
- **PMAN_close**: releases resources used by PMan;
- **PMAN_procadd**: adds a given process to the PMan table;
- **PMAN_procdel**: removes one process from the PMan table;
- **PMAN_attach**: attaches the OS process id to an already registered process, completing the registration phase;
- **PMAN_deattach**: clears the process id field from a PMan entry;
- **PMAN_QoSupd**: changes the QoS attributes of a process already registered in the PMan table;
- **PMAN_TPupd**: changes the temporal properties (period, phase or deadline) 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 the availability of every new frame, triggering the activation of processes.

The PMan service should be initialized before use, via the **init** function. The service uses OS resources that require proper shutdown procedures, e.g. shared memory and semaphores, and the **close** function should be called before terminating the application. To register in the PMan table, a process should call the **add** function and afterwards the **attach** function. This separation permits a higher flexibility since it becomes possible to have each process registering itself completely or to have a third process managing the overall properties of the different processes. During runtime the QoS allocated to each process may be changed with an appropriate call to **QoSupd** function. Similarly, the temporal properties of one

process can also be changed dynamically by means of the **TPupd** function. When a process terminates executing one instance it should report this event via the **epilogue** call. This action permits maintaining the statistical data associated with each process as well as becoming aware of deadline violations. The **query** call allows accessing the statistical data of each process registered in the database. This information can be used by the application for different purposes like profiling, load management, etc. Finally, the **tick** call is triggered by the process that interacts with the camera and signals that a new frame is ready for processing. As a consequence of this call the PMan database is scanned and the adequate processes activated.

4.2.2 Shared Data Buffers

As discussed previously, the robot application is composed by several processes which operate concurrently, each seeking for particular features in a given frame. The complexity of these activities is very dissimilar and consequently the distinct processes exhibit distinctive execution times. On the other hand the execution time of each process may also vary significantly from instance to instance, depending on the particular strategy followed, on the object dynamics, etc.. Consequently, the particular activation instants of the processes cannot be predicted beforehand. To facilitate the sharing of image buffers in this framework a mechanism called Shared Data Buffers (SDB) was implemented. This mechanism is similar to the Cyclic Asynchronous Buffers (Buttazzo, G.; Conticelli, F.; Lamastra, G. & Lipari, G., 1997), and permits an asynchronous non-blocking access to the image buffers. When the processes request access to an image buffer automatically receive a pointer to the most recent data. Associated to each buffer there is a link count which accounts for the number of processes that are attached to each buffer. This mechanism ensures that the buffers are only recycled when there are no processes attached to them, and so the processes have no practical limit to the time during which they can hold a buffer.
The access to the SDB library is made trough the following calls:

- **SDB_init**: reserves and initializes the diverse data structures (shared memory, semaphores, etc);
- **SDB_close**: releases resources associated with the SDB;
- **SDB_reserve**: returns a pointer to a free buffer;
- **SDB_update**: signals that a given buffer was updated with new data;
- **SDB_getbuf**: requests a buffer for reading;
- **SDB_unlink**: access to the buffer is no longer necessary.

The **init** function allocates the necessary resources (shared memory, semaphores) and initializes the internal data structures of the SDB service. The **close** function releases the resources allocated by the init call, and should be executed before terminating the application. When the camera process wants to publish a new image it should first request a pointer to a free buffer, via the **reserve** call, copy the data and then issue the **update** call to signal that a new frame is available. Reader processes should get a pointer to a buffer with fresh data via the **getbuf** call, which increments the link count, and signal that the buffer is no longer necessary via the **unlink** call, which decrements the buffer link count.

4.2.3 Dynamic QoS management

As in many other autonomous agent applications, the robotic soccer players have to deal with an open and dynamic environment that cannot be accurately characterized at pre-

runtime. Coping efficiently with this kind of ambiance requires support for dynamic reconfiguration and on-line QoS management (Burns, A; Jeffay, K.; Jones, M. et al, 1996). These features are generally useful to increase the efficiency in the utilization of system resources (Buttazzo, G.; Lipari, G., Caccamo, M. & Abeni. L., 2002) since typically there is a direct relationship between resource utilization and delivered QoS. In several applications, assigning higher CPU to tasks increases the QoS delivered to the application. This is true, for example, in control applications (Buttazzo, G. & Abeni, L., 2000), at least within certain ranges (Marti, P., 2002), and in multimedia applications (Lee, C.; Rajkumar, R. & Mercer, C., 1996). Therefore, managing the resources assigned to tasks, e.g. by controlling their execution rate or priority, allows a dynamic control of the delivered QoS. Efficiency gains can be achieved in two situations: either maximizing the utilization of system resources to achieve a best possible QoS for different load scenarios or adjusting the resource utilization according to the application instantaneous QoS requirements, i.e. using only the resources required at each instant.

Process	Period (ms)	Priority	Offset (ms)	Purpose
Ball_Fr	50	35	0	Ball tracking (front camera)
BGoal / YGoal	200	25	50/150	Blue / Yellow Goal tracking
BPost / YPost	800	15	100/200	Blue / Yellow Post tracking
Avoid_Fr	50	45	0	Obstacle avoidance (front cam.)
Ball_Om	50	40	0	Ball tracking (omni camera)
Avoid_Om	50	45	0	Obstacle avoidance (omni camera)
Line	400	20	0	Line tracking and identification

Table 2. Process properties in the modular architecture

Both situations referred above require an adequate support from the computational infrastructure so that the relevant parameters of tasks can be dynamically adjusted. Two of the functions implemented by the PMAN library, namely **PMAN_TPupd** and **PMAN_QoSupd**, allow changing dynamically and without service disruption the temporal properties of each process (period, phase and deadline) and to manage additional custom QoS properties (the Linux real-time priority in this case), respectively. The robots decision level uses this interface to adjust the individual process attributes in order to control the average CPU load and to adapt the rates and priorities of the diverse processes according to the particular role that the robots are playing in each instant.

5. Experimental Results

In order to assess the performance of the modular approach and compare it with the initial monolithic one, several experiments were conducted, using a PC with an Intel Pentium III CPU, running at 550MHz, with 256MB of RAM. This PC has lower capacity than those typically used on the robots but allows a better illustration of the problem addressed in this chapter. The PC runs a Linux 2.4.27 kernel, patched with RTAI 3.0r4. The image-capture devices are Logitech Quickcams, with a Philips chipset. The cameras were set-up to produce 320*240 images at a rate of 20 frames-per-second (fps). The time instants were measured accessing the Pentium TSC. To allow a fair comparison all the tests have been executed over the same pre-recorded image sequence.

5.1 Monolithic Architecture assessment

The code of the Frontvision and Omnivision processes (Section 3) was instrumented to measure the start and finishing instants of each instance.

Process	Max. (ms)	Min. (ms)	Avg. (ms)	St.Dev. (ms)
FrontVision	143	29	58	24
OmniVision	197	17	69	31

Table 3. FrontVision and OmniVision inter-activation statistical figures

Figure 5 presents the histogram of the inter-activation intervals of both of these processes while Table 3 presents a summary of the relevant statistical figures.

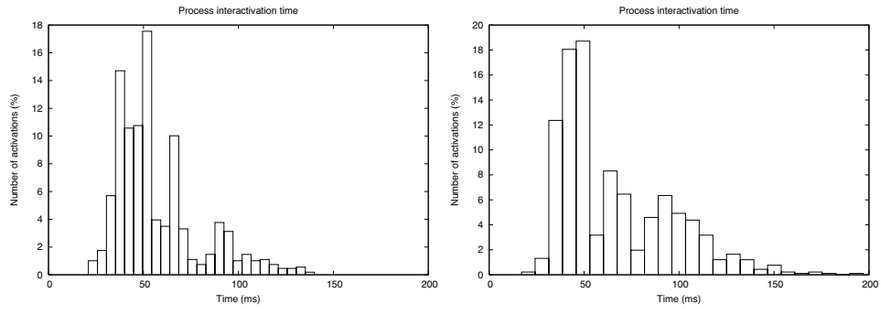

Figure 5. Histogram of the inter-activation time of the FrontVision (top) and OmniVision (bottom) processes

The response time of both processes exhibits a substantial variance, with inter-activation times ranging from 17ms to near 200ms and an average inter-activation time of 58ms and 69ms, respectively. Remembering that the image acquisition rate is 20 fps, corresponding to 50ms between frames, these figures indicate a poor performance. In fact the image processing is part of the control loop and so the high jitter leads to a poor control performance, a situation further aggravated by the significant amount of dropped frames, which correspond to time lapses during which the robot is completely non-responsive to the environment.

5.2 Modular Architecture

The different image-processing activities have been separated and wrapped in different Linux processes, as described in Section 4. Table 2 shows the periods, offsets and priorities assigned to each one of the processes.

The obstacle avoidance processes are the most critical ones since they are responsible for alerting the control software of the presence of any obstacles in the vicinity of the robot, allowing it to take appropriate measures when necessary, e.g. evasive maneuvers or immobilization.

Therefore these processes are triggered at a rate equal to the camera frame rate and receive the highest priority, ensuring a response-time as short as possible. It should be noted that these processes scan restricted image regions only, looking for specific features, thus their execution time is bounded and relatively short. In the experiments the measured execution

time was bellow 5ms for each one of the processes, therefore this architecture allows ensuring that every frame will be scanned for the presence of obstacles.

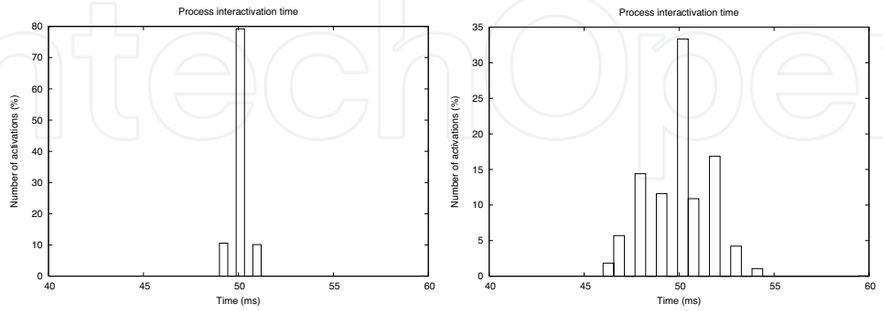

Figure 6. Front (left) and omni-directional (right) obstacle detection processes inter-activation intervals

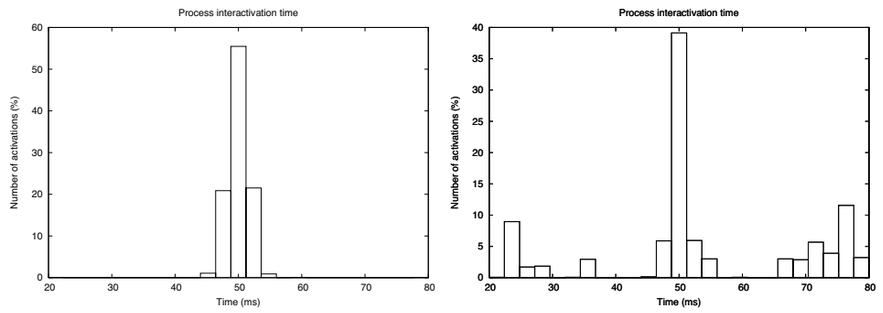

Figure 7. Omni-directional (left) and frontal (right) camera ball tracking processes inter-activation intervals

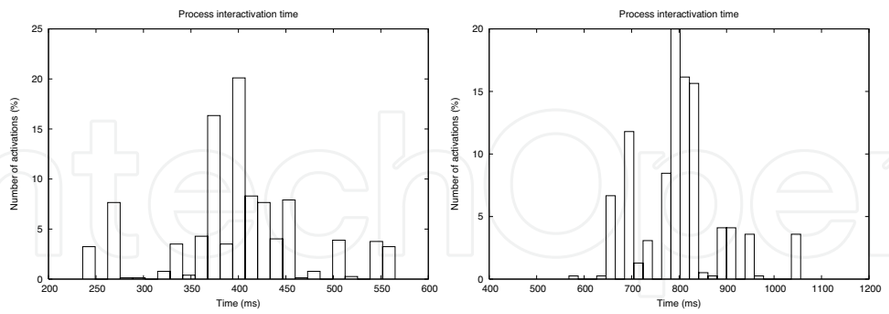

Figure 8. Line (left) and yellow post (right) tracking processes inter-activation intervals

The second level of priority is granted to the Ball_Om process, which tracks the ball in the omni-directional camera. This information is used when approaching, dribbling and kicking the ball, activities that require a low latency and high update rate for good performance.

Therefore this process should, if possible, be executed on every image frame, thus its period was also set to 50ms.
The third level of priority is assigned to the Ball_Fr process, responsible for locating the ball in the front camera. This information is used mainly to approach the ball when it is at medium to far distance from the robot. Being able to approach the ball quickly and smoothly is important for the robot performance but this process is more delay tolerant than the Ball_Om process, thus it is assigned a lower priority.

Process	Max. (ms)	Min. (ms)	Average (ms)	Standard deviation (ms)
Avoid_Fr	60.1	48.9	50.0	0.5
Avoid_Om	60.1	45.9	50.0	1.6
Ball_Om	60.1	46.0	50.0	1.6
Ball_Fr	80.0	19.9	50.0	2.1
Ygoal	362.2	61.1	207.9	58.3
BGoal	383.9	60.9	208.4	66.6
Line	564.7	235.6	399.9	71.9
BPost	1055.8	567.9	799.9	87.2
YPost	1156.4	454.4	799.6	114.3

Table 4. Modular architecture statistical data of inter-activation intervals

Some objects are stationary with respect to the play field. Furthermore, the robot localization includes an odometry subsystem that delivers accurate updates of the robot position during limited distances. This allows reducing the activation rate and priority of the processes related with the extraction of these features, without incurring in a relevant performance penalty. This is the case of BGoal and YGoal processes, which track the position of the blue and yellow goals, respectively, which were assigned a priority of 25 and a period of 200ms, i.e., every 4 frames.
The field line detection process (Line) detects and classifies the lines that delimit the play field, pointing specific places in it. This information is used only for calibration of the localization information and thus may be run sparsely (400ms). Post detection processes (BPost and YPost) have a similar purpose. However, since the information extracted from them is coarser than from the line detection, i.e., it is affected by a bigger uncertainty degree, it may be run at even a lower rate (800ms) without a relevant performance degradation.
The offsets of the different processes have been set-up to separate their activation as much as possible. With the offsets presented in Table 2, besides the obstacle and ball detection processes run every frame, no more than two other processes are triggered simultaneously. This allows minimizing mutual interference and thus reducing the response-time of lower priority processes.
Figure 6, Figure 7 and Figure 8 show the inter-activation intervals of selected processes, namely obstacle, ball, line and yellow post tracking, which clearly illustrate the differences between the modular and the monolithic architectures regarding the processes temporal behavior. The processes that receive higher priority (obstacle detection, Figure 6) exhibit a

narrow inter-activation variance, since they are not blocked and preempt other processes that may be running. Figure 7 shows the inter-activation intervals of the ball tracking processes. As stated above, the ball tracking process on the omni-directional camera has a higher priority since its data is used by more time sensitive activities. For this reason its inter-activation interval is narrower than the ball tracking process related to the front camera. As discussed in Section 4, the ball-tracking processes exhibit a significant execution time variance, since in some cases they are able to find the ball almost immediately while in other cases the whole image is scanned. For this reason the lower-priority ball-tracking process (frontal camera) exhibits a significantly higher inter-activation jitter than the higher-priority one. The same behavior is observed for the remaining processes, which see their inter-activation jitter increase as their relative priorities diminish.

Table 4 shows statistical data regarding the inter-activation intervals of these processes, which confirm, in a more rigorous way, the behavior observed above. The processes are sorted by decreasing priorities exhibiting, from top to bottom, a steady increase in the gap between maximum and minimum values observed as well as in the standard deviation. This is expected since higher priority processes, if necessary, preempt lower priority ones increasing their response-time.

Comparing the figures in Table 3 and Table 4, a major improvement can be observed with respect to the activation jitter of the most time-sensitive processes, which, for the most important tasks was reduced to 10ms (object avoidance and omni-directional ball tracking) and 30ms (frontal ball tracking). Furthermore, the standard deviation of the activation jitter of these processes is much lower (between 0.5ms and 2.1ms) and no frame drops have occurred, a situation that may have a significant impact on control performance.

During runtime higher priority processes preempt the lower priority ones, delaying its execution. This effect is clear in Table 4, with the goal, post and line processes exhibiting a much higher variability in their inter-activation times. Therefore, it can be concluded that the modular approach is effective, being able to privilege the execution of the processes that have higher impact on the global system performance.

5.3 Dynamic Qos adaptation

During runtime the robotic soccer players have to perform different roles, e.g., when a defender robot gets the ball possession and has a clear way in the direction of the opposite team goal it should assume an attacker role and some other team mate should take the defender role in its place. The relative importance of the image processing activities depends on the particular role that the robots are playing, e.g., in the situation described above the robot that is taking the defender role does not need to look for the ball in its vicinity, since this one is in the possession of a team mate, while it could benefit from a higher accuracy on the localization, achieved by tracking the field lines more often. Therefore, having the ability to change the image-processing attributes during runtime has the potential to increase the robot performance.

Another aspect that should not be neglected is that the environment strongly influences the image processing time since, depending on its *richness*, the algorithms may have to explore more or less regions of interest. As a result it is possible for the robotic players to perform differently in distinct environments or even in different times in the same environment, e.g., due to illumination variation. In these cases it may be interesting to manage the execution

rates of the image-processing activities in order to take the best possible profit of the CPU but without incurring in overloads that penalize the control performance.

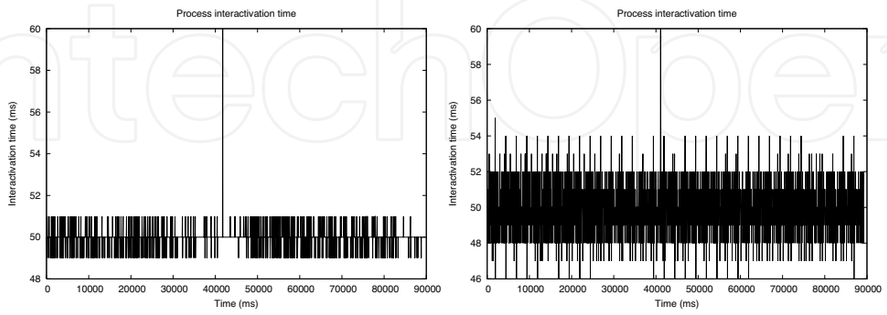

Figure 9. Inter-activation time of the high-priority frontal (left) and omnidirectional (right) avoid processes during a mode change affecting lower priority processes, only

As discussed in Section 4.2.1, the PMAN library permits to change the QoS properties of the processes, namely the period, phase, deadline and priority. To observe the impact of this service a situation was created in which the decision level requested a change in the role of a robot, from attacker to defender, as described before. Furthermore, a CPU overload was detected and thus the need to remove a lower importance process. The resulting actions were:

- to remove the ball tracking process in the omni-directional camera;
- to execute the front camera ball tracking process only once in each two frames;
- to execute the line tracking process for every frame;
- to raise its priority to 40, i.e., just below the obstacle avoidance processes.

Figure 9 and Figure 10 depict the inter-arrival time of the avoid, frontal camera ball-tracking and line tracking processes.

The first fact to be observed is that the higher priority processes are not affected, except for a small glitch on the instant of the QoS update, of similar magnitude as the jitter already observed (less than 10ms, see Table 4). This glitch may be explained by the need to access the PMAN table in exclusive mode and to call the Linux primitive sched_setscheduler()to change the priority of the line process. These operations are made within the PMAN_tick call, before the activation of the processes.

The second fact to be observed is that the line and frontal ball-tracking processes started to behave as expected immediately after the mode change, with periods of one and two frames, respectively.

The third fact to be observed is that the overload was controlled, and all the processes started to behave more regularly. This effect can be observed in medium priority processes (e.g. ball tracking) as well as in lower-priority processes (e.g. post seeking).

Therefore, it can be concluded that the PMAN services permit to change the process attributes at run-time, allowing both mode changes and CPU load management without disturbing the behavior of other processes not directly involved in the adaptation process and, consequently, it is possible to carry out the reconfiguration dynamically, since there is no service disruption.

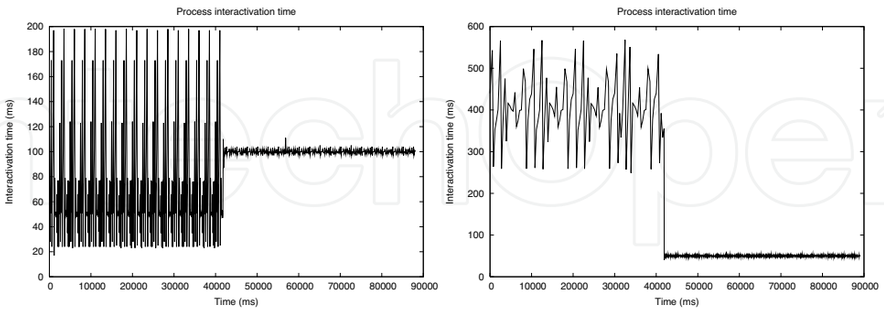

Figure 10. Inter-activation time of the frontal ball-tracking (left) and line (right) processes during a mode change in which the period of the former process was increased (50ms to 100ms) and the period of the latter was reduced (400ms to 50ms)

6. Conclusion

Computer vision applied to guidance of autonomous robots has been generating large interest in the research community as a natural and rich way to sense the environment and extract from it the necessary features. However, due to the robots motion, vision-based sensing becomes a real-time activity that must meet deadlines in order to support adequate control performance and avoid collisions. Unfortunately, most vision-based systems do not rely on real-time techniques and exhibit poor temporal behavior, with large variations in execution time that may lead to control performance degradation and even sensing black-out periods caused by skipped image frames.

In this chapter, the referred problem is identified in the scope of the CAMBADA middle-size robotic soccer team, being developed at the University of Aveiro, Portugal. Then, a new architectural solution for the vision subsystem is presented that substantially improves its reactivity, reducing jitter and frame skipping.

The proposed architecture separates the vision-based object-tracking activities in several independent processes. This separation allows, transparently and relying solely on operative system services, to avoid the blocking of higher priority processes by lower priority ones as well as to set independent activation rates, related with the dynamics of the objects being tracked and with its impact on the control performance, together with offsets that de-phase the activation instants of the processes to further reduce mutual interference.

As a consequence, it becomes possible to guarantee the execution of critical activities, e.g., obstacle avoidance and privilege the execution of others that, although not critical, have greater impact on the robot performance, e.g., ball tracking.

Finally, many robotic applications are deployed in open environments that are hard to characterize accurately at pre-runtime. The architecture herein proposed permits managing dynamically the resources assigned to tasks, e.g. by controlling their execution rate or priority, allowing a dynamic control of the delivered QoS. This approach permits either maximizing the utilization of system resources to achieve a best possible QoS for different load scenarios or adjusting the resource utilization according to the application instantaneous requirements, granting a higher QoS to the tasks that have higher impact on the global system performance.

The work described in this chapter is focused on robotic soccer robots but the results and approach are relevant for a wider class of robotic applications in which the vision subsystem is part of their control loop.

7. References

Almeida, L.; Santos, F.; Facchinetti; Pedreiras, P.; Silva, V. & Lopes, L. (2004). Coordinating distributed autonomous agents with a real-time database: The CAMBADA project. *Lecture Notes in Computer Science*, Volume 3280/2004, pp. 876-886, ISSN 0302-9743.

Assad, C.; Hartmann, M. & Lewis, M. (2001). Introduction to the Special Issue on Biomorphic Robotics. *Autonomous Robots*, Volume 11, pp. 195-200, ISSN 0929-5593.

Blake, A; Curwen, R. & Zisserman, A. (1993). A framework for spatio-temporal control in the tracking of visual contours. *International Journal of Computer Vision*, Vol. 11 No.2, pp. 127—145, ISSN0920-5691.

Burns, A; Jeffay, K.; Jones, M. et al (1996). Strategic directions in realtime and embedded systems. *ACM Computing Surveys*, Vol. 28, No. 4, pp. 751–763, ISSN 0360-0300.

Buttazzo, G.; Conticelli, F.; Lamastra, G. & Lipari, G. (1997). Robot control in hard real-time environment. *Proceedings of the 4th International Workshop on Real-Time Computing Systems and Applications*, pp. 152—159, ISBN 0-8186-8073-3, Taiwan, Oct. 1997, Taipei.

Buttazzo, G. & Abeni, L. (2000). Adaptive rate control through elastic scheduling. *Proceedings of the 39th IEEE Conference on Decision and Control*, pp. 4883-4888, ISBN 0-7803-6638-7, Dec. 2000, Sydney, Australia.

Buttazzo, G.; Lipari, G., Caccamo, M. & Abeni. L. (2002). Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, Vol. 51, No. 3, pp. 289–302, ISSN: 0018-9340.

CAN (1992). Controller Area Network - CAN2.0. *Technical Specification*, Robert Bosch, 1992.

Davison, J. (2005). Active search for real-time vision, *Proceedings of the 10th IEEE International Conference on Computer Vision*, Volume: 1, pp. 66- 73, ISBN 0-7695-2334-X.

De Souza, G. & Kak, A.(2004). A Subsumptive, Hierarchical, and Distributed Vision-Based Architecture for Smart Robotics. *IEEE Transactions on Systems, Man, and Cybernetics -- Part B: Cybernetics*, Vol. 34, pp. 1988-2002, ISSN 1083-4419.

Gibson, J. (1979). *The Ecological Approach to Visual Perception*, Lawrence Erlbaum Associates, Inc.,, ISBN 0-89859-959-8, Boston, MA.

Hirai, S.; Zakouji, M & Tsuboi, T. (2003). Implementing Image Processing Algorithms on FPGA-based Realtime Vision System, *Proceedings of the 11th Synthesis and System Integration of Mixed Information Technologies*, pp.378-385, March 2003, Hiroshima.

Iannizzotto, G., La Rosa, F. & Lo Bello, L. (2004). Real-time issues in vision-based Human-Computer Interaction. *Technical Report*, VisiLab, University of Messina , Italy.

Kitano, K.; Asada, M.; Kuniyoshi, Y.; Noda, I. & Osawa E. (1997). RoboCup: The Robot World Cup Initiative, *Proceedings of the First International Conference on Autonomous Agents* (Agents'97), W. Lewis Johnson and Barbara Hayes-Roth (Eds.), pp. 340—347, ISBN 0-89791-877-0, USA, Aug. 1997, ACM Press, N.Y.

Kopetz, H. (1997). *Real-Time Systems Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, ISBN 0-7923-9894-7, Boston, MA.

Lee, C.; Rajkumar, R. & Mercer, C. (1996). Experiences with processor reservation and dynamic qos in real-time Mach. In *Multimedia Japan* 96, Japan, April 1996.

Marti, P. (2002). Analysis and Design of Real-Time Control Systems with Varying Control Timing Constraints. *PhD thesis*, Universitat Politecnica de Catalunya, Barcelona, Spain, July 2002.

RTAI (2007), RTAI for Linux, Available from http://www.aero.polimi.it/~rtai/, accessed: 2007-01-31.

Santos, F.; Almeida, L.; Pedreiras, P.; Lopes, S. & Facchinnetti, T. (2004). An Adaptive TDMA Protocol for Soft Real-Time Wireless Communication Among Mobile Computing Agents, *Proceedings of the Workshop on Architectures for Cooperative Embedded Real-Time Systems* (satellite of RTSS 2004). Lisboa, Portugal, Dec. 2004.

SDL (2007), Simple DirectMedia Layer, Available from http://www.libsdl.org/index.php, accessed: 2007-01-31.

Weiss, G. (2000). *Multiagent systems. A Modern Approach to Distributed Artificial Intelligence*, MIT Press, ISBN 0-262-23203-0, Cambridge, MA.

Vision Systems: Applications

Edited by Goro Obinata and Ashish Dutta

ISBN 978-3-902613-01-1

Hard cover, 608 pages

Publisher I-Tech Education and Publishing

Published online 01, June, 2007

Published in print edition June, 2007

Computer Vision is the most important key in developing autonomous navigation systems for interaction with the environment. It also leads us to marvel at the functioning of our own vision system. In this book we have collected the latest applications of vision research from around the world. It contains both the conventional research areas like mobile robot navigation and map building, and more recent applications such as, micro vision, etc.The fist seven chapters contain the newer applications of vision like micro vision, grasping using vision, behavior based perception, inspection of railways and humanitarian demining. The later chapters deal with applications of vision in mobile robot navigation, camera calibration, object detection in vision search, map building, etc.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Paulo Pedreiras, Filipe Teixeira, Nelson Ferreira, Luis Almeida Armando Pinho and Frederico Santos (2007). A Real-Time Framework for the Vision Subsystem in Autonomous Mobile Robots, Vision Systems: Applications, Goro Obinata and Ashish Dutta (Ed.), ISBN: 978-3-902613-01-1, InTech, Available from: http://www.intechopen.com/books/vision_systems_applications/a_real-time_framework_for_the_vision_subsystem_in_autonomous_mobile_robots

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen