Enhancing the Reactivity of the Vision Subsystem in Autonomous Mobile Robots Using Real-Time Techniques*

Paulo Pedreiras[1], Filipe Teixeira[2], Nelson Ferreira[2], Luís Almeida[1], Armando Pinho[1], and Frederico Santos[3]

[1] LSE-IEETA/DET, Universidade de Aveiro Aveiro, Portugal
{pedreiras, lda, ap}@det.ua.pt
[2] DET, Universidade de Aveiro Aveiro, Portugal
{a23082, a21085}@alunos.det.ua.pt
[3] DEE, Instituto Politécnico de Coimbra, Coimbra, Portugal
fred@mail.isec.pt

Abstract. Interest on using mobile autonomous agents has been growing, recently, due to their capacity to cooperate for diverse purposes, from rescue to demining and security. In many of these applications the environments are inherently unstructured and dynamic, requiring substantial computation resources for gathering enough sensory input data to allow a safe navigation and interaction with the environment. As with humans, who depend heavily on vision for these purposes, mobile robots employ vision frequently as the primary source of input data when operating in such environments. However, vision-based algorithms are seldom developed with reactive and real-time concerns, exhibiting large variations in the execution time and leading to occasional periods of black-out or vacant input data. This paper addresses this problem in the scope of the CAMBADA robotic soccer team developed at the University of Aveiro, Portugal. It presents an evolution from a monolithic to a modular architecture for the vision system that improves its reactivity. With the proposed architecture it is possible to track different objects with different rates without losing any frames.

1 Introduction

Coordinating several autonomous mobile robotic agents in order to achieve a common goal is currently a topic of intense research [11,7]. This problem can be found in many robotic applications, either for military or civil purposes, such as search and rescue in catastrophic situations, demining or maneuvers in contaminated areas. One initiative to promote research in this field is RoboCup [7] where several autonomous robots have to play football in a team to beat the opponent.

As for many real-world applications, robotic soccer players are autonomous, though potentially cooperative, mobile agents that must be able to navigate in and

* This work was partially supported by the European Community through the ARTIST2 NoE (IST-004527) and by the Portuguese Government through the project FCT-POSI/ROBO/43908/2002, also partially funded by FEDER.

A. Bredenfeld et al. (Eds.): RoboCup 2005, LNAI 4020, pp. 371 – 383, 2006.

interact with their environment. Some of these actions exhibit real-time characteristics, although with different levels of criticality. For instance, the capability to timely detect obstacles in the vicinity of the robot can be regarded as a hard activity since failures, either in the temporal or value domains, can result in injured people or damaged equipment. On the other hand, activities like self-localization or tracking the ball, although important for the robot performance, are inherently soft since failing in these activities simply causes performance degradation. The capability to timely perform the required image-processing activities at rates high enough to allow visual-guided control or decision-making is called real-time computer vision (RTCV) [13].

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* [2]. 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) strongly impact the overall system performance. Therefore, the application of real-time techniques to these activities so as to improve their temporal behavior by reducing mutual interference and limiting processing demand seems adequate to improve global performance.

In this paper we propose a new modular architecture for the vision subsystem, where different objects are tracked by independent processes. Using appropriate operating system services, these processes are then scheduled according to their relative importance, with preemption. The result is a noteworthy improvement of the temporal behavior of the processes deemed to have greater impact on the overall system performance.

The remainder of the paper 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 paper.

2 The CAMBADA Computing Architecture

The computing architecture of the robotic agents follows the biomorphic paradigm [9], being centered on a main processing unit (*the brain*) that is responsible for the higher-level behavior coordination (Fig. 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 PC-based computers that deliver enough raw computing power and offer standard interfaces to connect the other systems, namely USB. The PCs run the Linux operating system over the RTAI (Real-Time Applications Interface [5]) 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 [3] based on IEEE 802.11b that reduces the probability of transmission collisions between team mates thus reducing the communication latency.

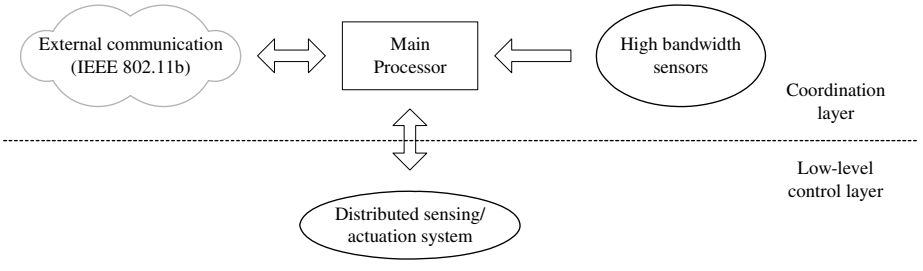

Fig. 1. The biomorphic architecture of the CAMBADA robotic agents

The low-level sensing/actuating system follows the fine-grain distributed model [8] 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 is used the CAN (Controller Area Network) protocol is used [1], 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

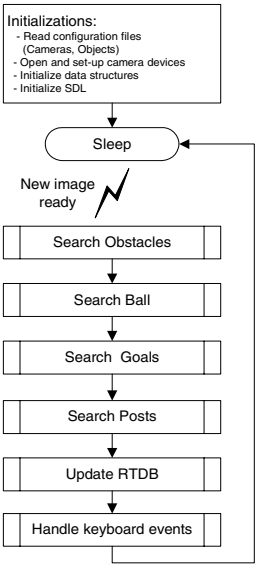

Fig. 2. Flowchart of the Frontvision process

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.

Keyboard, mouse and the video framebuffer are accessed via the Simple DirectMedia Layer library (SDL, [12]). 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.

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 [4], with hard to predict execution times. Regarding the algorithms described in Section 3, above, 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. 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, taking less than 2.5ms to complete. However, a significant amount of instances require between 17.5ms and 32.5ms to complete and, sometimes, the process requires over 50ms, which is the inter-frame period used by the cameras.

As described in Section 3, the CAMBADA vision subsystem architecture is monolithic with respect to each camera, with all the image-processing carried out

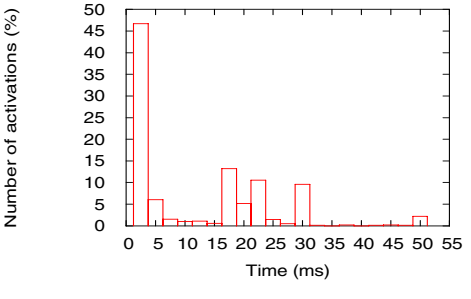

Fig. 3. Ball tracking execution time histogram captured at run-time

within two processes, the Frontvision and the Omnivision, respectively. Each of these processes tracks several objects sequentially. Thus, the following frame is acquired and analysed only after tracking all objects, 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 1, some activities, like obstacle detection, have hard deadlines this situation becomes clearly unacceptable. Increasing the available processing power could, to some extent, alleviate the situation (although not completely solve it). However, the robots are autonomous and operate from batteries, and thus energy consumption aspects are highly relevant, which renders brut-force approaches undesirable.

4.1 Using Real-Time Techniques to Manage the Image Processing

As remarked is 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 people, walls or other robots. 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. However, this activity should not block the more frequent obstacle detection. This calls for the encapsulation of each object tracking 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, for each of the two cameras used. Standard Linux services are used to implement priority scheduling, preemption and data sharing.

Each camera uses one process (*ReadXC*) to create a shared memory region where the images are buffered. The processes are periodically triggered by the cameras, whenever a new image frame is available. Each object tracking process (e.g. obstacle,

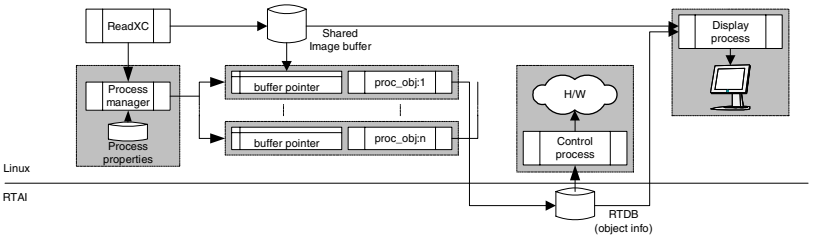

Fig. 4. Modular software architecture for the CAMBADA vision subsystem

ball), generically designated by proc_obj:x, x={1,2,…n}, is triggered by a process manager module, according to the relevant process 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 [6] (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.

Scheduling of vision related processes relies on the real-time features of the Linux kernel, namely the FIFO scheduler and priorities in the range 15-50. At this level Linux executes each process to completion, unless the process blocks or is preempted by other process with higher real-time priority. This ensures that the processes are executed strictly according to their priority (i.e., importance) with full preemption, relying solely on operating system services, in a non intrusive and transparent way. The real-time features of Linux, despite limited, are sufficient at this time-scale (periods multiple of 50ms) as long as memory swapping and disk access are avoided.

The buffer management system keeps track of the number of processes that are connected to each 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. This approach is adequate to situations where different objects have different degrees of dynamism, e.g., the ball is highly dynamic and needs being tracked in every frame but the relative goal position is less dynamic and can be tracked every four frames. Moreover, in the latter case prediction methods can be effectively used, as suggested in [10], to allow obtaining estimates of object positions based on past data.

The process activation is carried out by a process manager that keeps, in a database, the process properties, e.g. priority, period and phase. For each activation, the process manager scans the database, identifies which processes should be activated and sends them appropriate signals. The cameras are set to grab images periodically, and the arrival of each new image frame is used to activate the process manager. This framework allows reducing the image processing latency, since processes are activated immediately upon the arrival of new images. Another relevant feature that is supported is the possibility of de-phasing the process activations in the time domain, to minimize mutual interference and thus reducing their response time.

5 Experimental Results

In order to assess the benefits of the modular approach with respect to the initial monolithic one, several experiments were conducted, using a PC with an Intel Pentium III CPU, running at 833MHz, with 256MB of RAM. The PC runs a Linux 2.4.21 kernel, patched with RTAI 24.1 for the experiments involving the real-time architecture. 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. The time instants were measured accessing the Pentium TSC.

5.1 Monolithic Architecture

The code of the Frontvision and Omnivision processes (Section 3) was instrumented to measure the start and finishing instants of each instance. Figure 5 presents a histogram of the inter-activation intervals of these processes, while Table 1 presents a summary of some relevant statistical figures regarding their response-time.

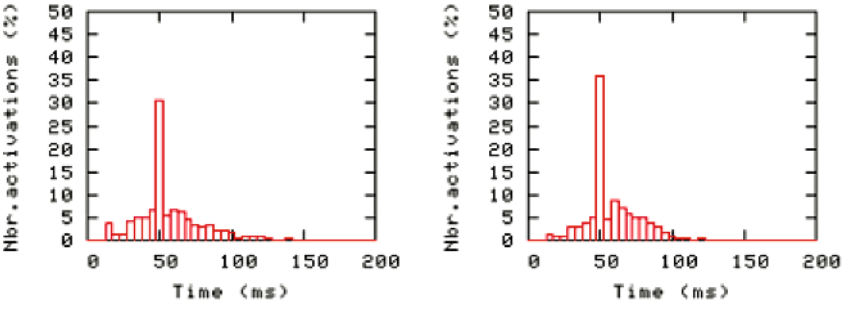

Fig. 5. Activation interval for Frontvision (left) and Omnivision (right)

The response time of both processes exhibits a substantial variance, with execution times ranging from 10ms to over 600ms and an average execution time around 44ms. Noting that the processing of a frame can only start after the previous one being fully processed, this response-time variance leads to the activation time dispersion observed in Figure 5.

Table 1. Frontvision and Omnivision response–time statistical figures

Process	Max (ms)	Min (ms)	Average (ms)	Stand. deviation
Frontvision	427.7	9.9	43.6	30.6
Omnivision	635.5	10.8	44.0	30.4

The combination of such an irregular activation pattern with highly variant response-times results in a high jitter value and number of skipped frames. Remembering that the image-processing is part of the robot control loop, this leads to an irregular and poor robot performance.

5.2 Modular Architecture

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

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 (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 remarked that these processes scan delimited image regions, only, looking for specific features, thus their execution time is bounded and relatively short. In the experiments the measured execution time upper bounds were 7ms and 9ms for *Avoid_Om* and *Avoid_Fr*, respectively. Therefore, this architecture allows ensuring that every frame will be scanned for the presence of obstacles.

Table 2. Image-processing processes properties

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
KGoal	100	30	0	Goal line tracking

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 to be successful. 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.

The *KGoal* process detects the goal lines, being mainly used by the goal keeper. Contrarily to the previous processes, the dynamics of the lines depend only on the robot movement. Furthermore, the localization of the robot within small regions is complemented with an odometry subsystem, which updates the robot position. This allows having a lower activation rate and priority of the vision-based goal lines detection without incurring in significant performance degradation.

A similar reasoning was applied to decide the attributes of the *BGoal* and *YGoal* processes, which track the position of the blue and yellow goals, respectively. Since the goals are stationary with respect to the play fields, and due to the availability of odometry data, updates can be made more sparsely and are not particularly sensitive to jitter. For this reason these processes were assigned a priority of 25 and a period of 200ms (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 de-phase the process activations as much as possible. With the offsets presented in Table 2, besides the obstacle and ball detection processes, which are executed for 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.

Figures 6, 7 and 8 show the inter-activation intervals of selected processes (obstacle, ball, goal 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, Fig. 6) exhibit a narrow inter-activation variance, since they are not blocked and preempt other processes that may be running. Figure 7 regards the inter-activation intervals of the ball tracking processes. As stated above, the ball tracking process on

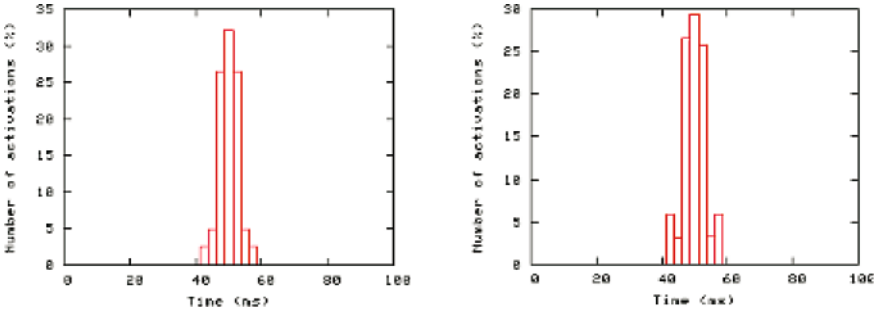

Fig. 6. Front (left) and omni-directional (right) obstacle detection processes inter-activation intervals

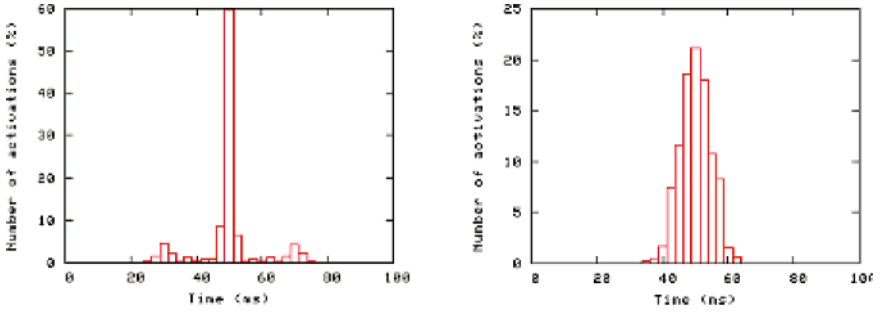

Fig. 7. Front (left) and omni-directional (right) ball tracking processes inter-activation intervals

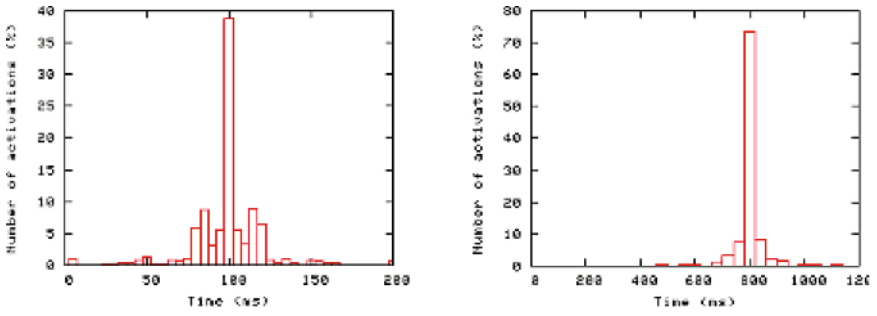

Fig. 8. Goal line (left) and yellow post (right) tracking processes inter-activation intervals

the omni-directional camera received 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. Ball-tracking processes exhibit a significantly higher response-time jitter than obstacle detection because, in the worst case, they must scan the whole image. For this reason the lower-priority ball-tracking process exhibits a much 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 3 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 data in Tables 1 and 3, a major improvement can be observed with respect to the activation jitter of the most time-sensitive processes, which was reduced from around 30ms to 2ms-3ms (object avoidance) and 3ms-9ms (ball tracking). Furthermore, since the obstacle avoidance processes have a relatively short execution

Table 3. Modular architecture: statistical data of process inter-activation intervals

Process	Max (ms)	Min (ms)	Standard deviation (ms)
Avoid_Om	56.9	43.0	1.9
Avoid_Fr	58.1	41.9	2.5
Ball_Om	63.6	35.8	4.7
Ball_Fr	75.9	25.4	8.9
KGoal	506.6	0.5	27.4
Bgoal	547.9	0.6	39.1
YGoal	654.6	0.6	42.0
Line	711.0	38.8	53.3
BPost	1159.7	541.1	60.8
YPost	1101.0	485.6	53.3

time (up to 8.5ms in this architecture) it becomes possible to guarantee their execution in every image frame allowing the robots to run at higher speed without compromising safety.

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 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 very poor temporal behavior, with large variations in execution time that may lead to control performance degradation and even sensing black-out periods (skipped image frames).

In this paper, 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, together with offsets that de-phase the activation instants of the processes to further reduce mutual interference.

As a consequence, it became 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). This result and approach are relevant for a class of robots in which the vision subsystem is part of their control loop, leading to a better control performance.

References

1. Controller Area Network - CAN2.0 (1992). Technical Specification, Robert Bosch, 1992.
2. J. Gibson (1979), “The Ecological Approach to Visual Perception”, Houghton Mifflin, Boston, MA, 1979.
3. F. Santos, L. Almeida, P. Pedreiras, L.S.Lopes, T. Facchinnetti (2004), "An Adaptive TDMA Protocol for Soft Real-Time Wireless Communication Among Mobile Computing Agents". WACERTS 2004, Workshop on Architectures for Cooperative Embedded Real-Time Systems (satellite of RTSS 2004). Lisboa, Portugal, 5-8 Dec. 2004.
4. Guilherme N. DeSouza and Avinash C. Kak (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, October 2004.
5. RTAI for Linux, available at http://www.aero.polimi.it/~rtai/

6. L. Almeida, F. Santos, T. Facchinetti, P. Pedreiras, V. Silva, L.S.Lopes (2004). "Coordinating distributed autonomous agents with a real-time database: The CAMBADA project". ISCIS'04, 19th International Symposium on Computer and Information Sciences. 27-29 October 2004, Kemer - Antalya, Turkey.
7. K. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa (1995) "RoboCup: The Robot World Cup Initiative", *Proc. of IJCAI-95 Workshop on Entertainment and AI/Alife,* Montreal.
8. H. Kopetz (1997). "Real-Time Systems Design Principles for Distributed Embedded Applications", Kluwer.
9. Proc. of the NASA Workshop on Biomorphic Robotics, (2000), Jet Propulsion Laboratory, California Institute of Technology. USA.
10. G. Iannizzotto, F. La Rosa, L. Lo Bello (2004). "Real-time issues in vision-based Human-Computer Interaction". Technical Report, VisiLab, University of Messina , Italy.
11. G. Weiss (2000), "Multiagent systems. A Modern Approach to Distributed Artificial Intelligence" MIT Press, 2000.
12. Simple DirectMedia Layer, available at http://www.libsdl.org/index.php
13. Andrew Blake, R. Curwen, A. Zisserman (1993), "A framework for spatio-temporal control in the tracking of visual contours". Int. Journal of Computer Vision, 11, 2, 127—145, 1993.