Hierarchical Distributed Architectures for Autonomous Mobile Robots: a Case Study

José Luís Azevedo, Bernardo Cunha, Luís Almeida
Universidade de Aveiro, LSE-IEETA / DETI, 3810-193 Aveiro, Portugal

{jla, mbc, lda}@det.ua.pt

Abstract

Robots are becoming commonplace in unstructured and dynamic environments, ranging from homes to offices, public sites, catastrophe sites, military scenarios. Achieving adequate performance in such circumstances requires complex control architectures, mixing adequately deliberative and reactive capabilities. This mixing needs to be properly addressed from both the software and hardware architectures point of view and, particularly, the mapping of the former onto the latter, in order to reduce mutual interference between concurrent behaviors and support the desired coordination with adequate level of reactivity. This paper discusses the benefits of using hierarchical distributed hardware architectures and presents the case study of the CAMBADA soccer robots developed at the University of Aveiro, Portugal. These robots use a distributed hardware architecture with a central computer to carry out vision sensing, global coordination and deliberative functions and a low-level distributed sensing and actuation system based on a set of simple microcontroller nodes interconnected with a Controller Area Network (CAN).

1. Introduction

Autonomous mobile robots are becoming commonplace in unstructured and dynamic environments, such as homes, offices, public facilities and military scenarios. For example, autonomous vacuum cleaners, floor washers, grass mowers, interactive toys, as well as surveillance, demining and rescue robots are going through a steep growth and are expected to grow even further in following years [3]. Acting in such environments requires complex behaviors that must be merged, coordinated and enforced. Therefore, adequate computational architectures must be used to support the effective behaviors execution, controlling various robotic components and subsystems. This architecture needs to address both the component level, which manages the basic robot subsystems and their individual actions, as well as the high-level coordination and control, which allows achieving the desired overall system behavior.

A natural way to organize the architecture model referred above is to use a hierarchical approach with the two levels clearly separated, possibly further refined in more sublevels, and using their own computing resources. This has the advantage of decoupling the levels, separate their concerns, which have substantially different requirements, and minimize mutual interference. The high-level coordination and control system is, in general, computationally intensive thus, a centralized approach based on a single PC is commonly found. This approach also has the advantage of providing standard interfaces to specific high bandwidth I/O devices, such as cameras and wireless communication. On the other hand, a distributed approach to implement the component level of a mobile robot has several advantages [1]: a) it enables a scalable design because it is easier to add functionality to the robot; b) it is more flexible since a distributed system can be easily reconfigured; c) implementation of complex tasks can be facilitated through division into simpler ones; d) basic reactive control actions involving fast local feedback loops are easier to implement.

This paper discusses the advantages of using hierarchical distributed hardware architectures in building mobile autonomous robots and presents a case study referring to the CAMBADA team of soccer robots (Cooperative Autonomous Mobile Robots with Advanced Distributed Architecture) for the Robocup Middle Size League. The CAMBADA project aims at researching in several areas typical to autonomous systems, from basic robot hardware to multi-agent systems. The robots have been developed from scratch and, unlike other approaches, using home-made mechanical parts and basic electronic modules. This paper focuses on the definition of the functional architecture, on the distributed hardware architecture and its modules, and on the mapping of the former onto the latter. The remainder of the paper is organized as follows: Section 2 discusses the use of hierarchical distributed architectures in autonomous mobile robots. Section 3 presents the general functional architecture

1-4244-0826-1/07/$20.00 © 2007 IEEE

of the CAMBADA robots and the main hardware components. Section 4 discusses the mapping of the former onto the latter and describes the hardware architecture in detail. Section 5 presents the main information flows and their synchronization. Section 6 presents some results and Section 7 concludes the paper.

2. Why using hierarchical distributed architectures in robots

After a period of evolution in autonomous robots programming, which went all the way from deliberative architectures to reactive and hybrid deliberative-reactive ones, the architecture that became, probably, the most common for controlling complex autonomous robots is the so-called behavior-based [2]. According to this architecture, the robot control is essentially distributed over several concurrent behaviors, possibly organized in a hierarchical fashion, mixing reactive behaviors with cognitive and deliberative ones. The execution requirements of these behaviors, however, are substantially different. While reactive behaviors are normally simple and must be executed at relatively fast rates, deliberative ones are more computationally demanding and naturally require slower rates.

It is curious to note that the discussions concerning the control architecture of autonomous robots generally focuses on the functional architecture and its mapping onto an adequate software architecture. Not so much has been said about the hardware architecture. This is understandable when talking of uniprocessor-based robots, in which the referred architecture mapping, e.g., using tasks in a suitable multi-tasking kernel possibly with real-time features, determines much of the robot reactive capabilities. However, the use of a uni-processor system establishes a common system resource that is shared among all behaviors, i.e., the CPU itself, raising the possibility for mutual interference with potential negative effects in terms of timeliness, e.g., with longer and slower behaviors blocking faster reactive ones. Such a blocking could endanger both the robot and the environment by reducing its reactive capabilities.

Avoiding these problems is easier, nowadays, with the help of both powerful real-time multitasking kernels and appropriate scheduling techniques [17] as well as with the high availability of inexpensive and relatively powerful microcontrollers with embedded communications capabilities [1]. The latter fact enabled the use of distributed hardware architectures, which became the preferred architectural paradigm for complex embedded systems during the last decade, with some applications to the control of autonomous mobile robots [4], [6], [7], [8], [11].

The advantages of distributed architectures extend from improved *composability,* allowing a system to be built by putting together different subsystems, to higher *scalability,* allowing to add functionality to the system by adding more nodes, more *flexibility,* allowing to reconfigure the system easily, better *maintainability,* due to the architecture modularity and easiness of node replacement, and *higher reduction of mutual interference,* thus offering a strong potential to support reactive behaviors more efficiently. Moreover, distributed architectures may also provide benefits in terms of *dependability* by creating error-containment regions at the nodes and opening the way for inexpensive spatial replication and fault tolerance.

Several distributed hardware architectures within autonomous robots are reported in the literature. The work in [7] presents a robot for orange picking that is divided in four platforms, each one with two picking arms and all four of them interconnected with an SP50 (later Foundation Fieldbus FF-H1) fieldbus. In [8] an industrial robot is presented, which is based on a PROFIBUS network.

One particular protocol that has been substantially used within mobile robots is CAN [9] due to its low price, simple usability, good reliability and timeliness properties. Examples of using this protocol can be found in [6], [10], [11]. The latter reference is particularly relevant to this work since it addresses the concerns of supporting a distributed sensing and actuation system integrated in a hierarchical architecture that also has a deliberative level. In [12] the same authors discuss the impact of real-time data transfers jitter on closed-loop control performance and propose a mixed CAN-based event/time-triggered protocol.

Within RoboCup MSL, distributed hardware architectures based on buses are not typical. In most cases, there is a single main computer that controls all functions of the robot directly. However, there are examples of specialized hardware support to off-load the main CPU of certain low-level demanding functions, such as closed-loop motor and ultra-sound sensing. This is the case of [18], in which a DSP module connected to the main CPU via a serial port is used to carry out low-level control functions. Some other cases, such as reported in [19], go a bit further in using a distributed architecture with several microcontrollers but all connected to the main CPU through USB channels, thus without the capability of direct cross communication and with relatively low reliability. Conversely, using the architectural paradigm proposed in this paper, based on an adequate bus technology, such as CAN, it is possible to achieve a substantially higher reliability and flexibility in the functions distribution and integration.

3. General architecture

The general architecture of the CAMBADA robots has been described in [4], [5]. Basically, the robots follow a biomorphic paradigm [13], each being centered on a main processing unit, the *brain*, which is responsible for the higher-level behavior coordination, i.e. the coordination layer. This main processing unit handles external communication with the other robots and has high bandwidth sensors, typically 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* (Figure 1).

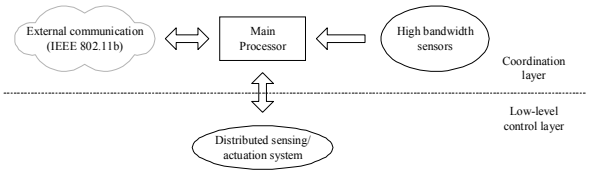

Figure 1. The biomorphic architecture of the CAMBADA robots.

At the heart of the coordination layer is the Real-Time Database (RTDB) containing both the robot local state information as well as local images of a subset of the other robots states. A set of processes update the local state information with the data coming from the vision sensors as well as from the low-level control layer. The remote state information is updated by a process that handles the communication with the other robots via an IEEE 802.11b wireless connection. The RTDB is then used by another set of processes that define the specific robot behavior for each instant, generating commands that are passed down to the low-level control layer (Figure 2).

The low-level sensing/actuation system executes four main functions as described in Figure 2, namely Motion, Odometry, Kick and System monitoring. The former provides holonomic motion using 3 DC motors. The Odometry function combines the encoder readings from the 3 motors and provides a coherent robot displacement information that is then sent to the coordination layer. The Kick function includes the control of an electromagnetic kicker and of a ball handler to dribble the ball. Finally, the *system monitor* function monitors the robot batteries as well as the state of all nodes in the low-level layer.

Finally, the low-level control layer connects to the coordination layer through a gateway, which filters interactions within both layers, passing through the information that is relevant across the layers, only. Such filtering reduces the overhead of handling unnecessary receptions at each layer as well as the network bandwidth usage at the low-level side, thus further reducing mutual interference across the layers.

4. Mapping the functional architecture onto hardware

The hardware architecture of the CAMBADA robots was designed to adapt naturally to the functional architecture described above. Firstly, two separate computing resources were employed for the two layers. Secondly, the low-level control layer was implemented with a set of simple microcontrollers organized in a set of subsystems. However, as it will be shown next, such subsystems are not completely isolated but they share computing resources, thus improving the efficiency of the global architecture and following recent trends in the embedded systems design community, from federated to integrated hardware architectures [20].

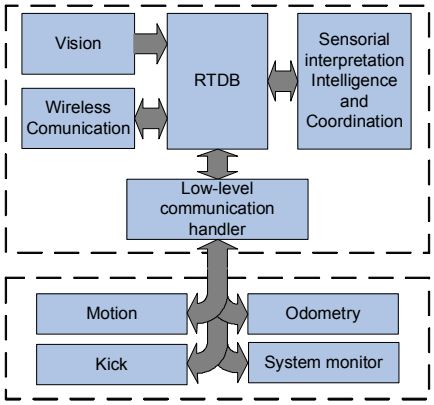

Figure 2. The robots functional architecture built around the RTDB.

Basically, the proposed approach follows the fine-grain distributed model [1] 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. The issue, then, is to control the functional mapping onto the hardware architecture to take as much advantage as possible from its distribution. This is an important step since a poor mapping may lead to extra delays and delay variations. Previous work in this direction [21], [22] shows the use of optimization techniques to minimize certain parameters such as end-to-end delays, network transactions and load asymmetries among nodes or even to meet a set of global constraints such as real-time and precedence. Instead, we follow a set of simple heuristics combined with an explicit synchronization mechanism (referred in the following section), which minimizes mutual interference, yielding good performance in terms of end-to-end delays and jitter, avoids the need for complex software infrastructures to

provide preemption support within the nodes and maximizes modularity, increasing system maintainability. Such criteria are the following:

1. *Closed-loop or sensing functions tied to specific I/O are allocated to the respective modules.*
2. *Hierarchical functions without specific I/O are allocated to dedicated modules.*

In the remainder of this section we will review the hardware architecture of the CAMBADA robots, describing its organization, its modules, with focus on the specialized hardware, as well as the mapping of the system functions.

4.1. The high-level coordination layer

The natural implementation of the main processing unit is on one PC-based computer (currently a standard 12" notebook based on an Intel Core2Duo processor) that delivers enough raw computing power and offers standard interfaces to connect to the lower layer (USB/serial ports), camera (Firewire) and other robots (WiFi). The PCs run the Linux operating system over the RTAI (Real-Time Applications Interface [14]) kernel, which provides time-related services, namely periodic activation of processes, time-stamping and temporal synchronization. The camera is part of a hyperbolic mirror-based vision system providing 360 degrees panoramic vision. The communication among team robots uses an adaptive TDMA transmission control protocol [15] on top of IEEE 802.11b that reduces the probability of transmission collisions between team mates thus reducing the communication latency.

4.2. The low-level control layer

The low-level layer has a set of nodes built around a common module described in the following section and using specialized interfacing to the robot I/O devices. These nodes are interconnected with a CAN network operating at a bit rate of 250Kbps. CAN is rather convenient since it has a deterministic medium access control, a good bandwidth efficiency with small packets and a high resilience to external interference. A gateway interconnects the CAN network to the PC at the high-level layer either through a serial port or a USB port, operating at 115Kbaud in any case.

4.3. The basic module

Despite the possibility of using heterogeneous modules, the option was to develop all modules based on the same underlying hardware, which is a more economic solution and facilitates development and maintenance. The core of each module is a PIC18Fxx8 Microchip [16] microcontroller (@40MHz, i.e., 10 MIPS) which, along with a set of useful peripherals, such as timers, PWM generators, analog to digital converter and serial communications, also integrates a CAN controller. The basic structure of every module includes the CAN port to connect to the network and also includes a 115 Kbps RS232 serial port, which is useful both to program the module firmware and for debugging purposes (Figure 3).

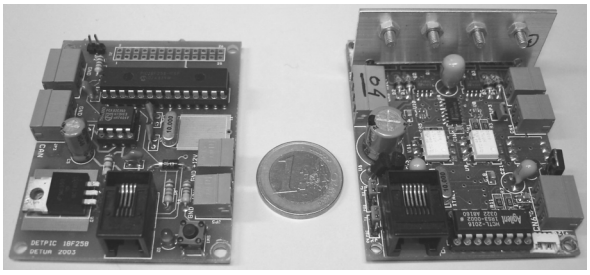

Figure 3. Basic module (left) and Motor controller (right).

4.4. The motion controller

The robot holonomic motion is obtained combining the speed of 3 DC motors (24V-150W), each with its own speed controller. Each of these controllers is a distinct module (Figure 3) of the whole distributed architecture implementing a PI closed loop speed control. It takes as inputs the motor shaft displacement, obtained through a quadrature incremental optical shaft encoder coupled to the motor, and the speed set-point. The computation of the three set-points needed to obtain a coherent robot motion is carried out by a function allocated to a new module called *holonomic*. It receives the robot velocity vector (speed, direction and heading) from the gateway and translates it into individual set-points for each motor controller. The interaction with the gateway and motor controllers is carried out through the CAN network.

The specialized hardware of these modules has two main blocks: the *logic* block that interfaces to the basic module and generates the required control signals, and the *power* block which is essentially an NMOS H-Bridge, with two high-side drivers, to actually drive the motor. The output of the *logic* block is a set of four 20 KHz PWM signals implementing a modified lock anti-phase drive. In this drive mode the motor is energized only during the on-time, in contrast with the standard lock anti-phase where the motor is energized in reverse direction during the off-time. That is, when the motor is stopped (duty-cycle of the PWM signals is 50%) the current is zero. This implementation leads to a significant gain in autonomy, whenever the motor is not rotating at its maximum speed, which is an important issue in mobile robotics.

One important characteristic of these motor controllers is the galvanic decoupling between the *logic* block and the *power* block carried out through opto-couplers. Along with improved reliability of the

system it prevents serious damages in expensive equipment (such as the notebook in the high-level layer) whenever any electric problem occurs. The drawback of this solution is the need of an extra battery for the *logic* part of the system.

4.5. Odometry

The odometry function of the robot is accomplished through the combination of 4 basic functions: the reading of the 3 encoders plus their combination to generate a coherent displacement information (Δx, Δy, $\Delta\theta$). The reading of each encoder is naturally allocated to each motor module, using the same readings as those used by the speed feedback control. The combination of the readings is carried out in a specific module, the *odometry manager*, which receives the encoder readings from the motors and sends the results to the gateway via CAN messages.

4.6. The kicking system

The kicker is an essential part of a soccer robot and, to be useful, it must allow the control of the kicking power. There are mainly three types of kickers currently in use in RoboCup: spring-based, pneumatic and electromagnetic. The spring-based kickers store energy in a spring, through mechanical means. The spring is released whenever the robot has to kick the ball. Although conceptually simple it is hard to implement and hard to modulate the kicking, since it has to be done through mechanical means.

The pneumatic kicker is a popular approach used by many teams. The robot carries an air recipient that has to be filled to a pre-defined pressure before a match. The actuator is mainly composed of a pneumatic cylinder that is controlled by means of a valve regulating the airflow from the recipient (thus allowing the shooting power to be controlled). As the shooting power depends on the pressure of the air recipient high pressure is needed; furthermore, as the number of kicks depends on the air storage capacity, a large recipient is needed.

The third approach, which is the one adopted by the CAMBADA team, is the so-called electromagnetic kicker whose main element is an electromechanical solenoid. The solenoid consists of a coil, wound around a movable iron core producing a magnetic field when an electric current passes through it. The magnetic field causes the iron core to move towards the ball, thus kicking it. Controlling the magnetic field provides control over the kicking power, and that represents a very convenient way to modulate the kicking action. The energy needed to drive the solenoid is stored in a capacitor. To get a strong kick a large magnetic field has to be created which implies the usage of reasonably high currents and/or voltages and also of large capacitors.

The kicking system is based on a basic module extended with specific I/O hardware with a *logic* and a *power* block with galvanic decoupling similarly to the motor controller. The two main components of the *power* block are: 1) a DC to DC converter circuit that stores energy in the capacitor; b) a solid-state switch that controls the discharge of the capacitor on the solenoid thus triggering the kicker.

The DC to DC converter is a typical switch-mode converter based on a boost configuration that converts 24V DC to 100V DC. In general terms, it works in two steps: 1) a DC voltage is set across an inductor during a pre-defined period of time which causes the inductor to store energy magnetically; 2) the voltage is switched off which causes the stored energy to be transferred to the capacitor. Although very simple, this circuit is very efficient resulting in a rather low capacitor recharge time. The implemented circuit works at 18KHz and, in practical terms, the recharge (from 24V to 100V) of a 80000µF capacitor takes roughly 6s. The capacitor charging process is carried out in a closed-loop way, being the voltage across the capacitor continuously monitored by the microcontroller. The output of the microcontroller is a 18 KHz / 35% PWM signal which has been found experimentally as optimal in order to minimize the charging time. This is crucial since an inefficient charging process can dramatically decrease the running time of the battery.

The second component referred above is a solid-state switch based on NMOS transistors, whose specifications (100V / 150A in our design) depend essentially on the capacitor voltage and current drawn by the solenoid.

The kicking system also includes two IR sensors used to implement a local behavior to optimize kicking: an IR barrier which is used to detect the ball when it is in the kicking position, thus avoiding false triggering; and a short distance IR sensor (less than 50 cm) which can be used, in addition to visual information, to determine more precisely the distance between the front of the robot and the ball.

Another feature implemented in this module is an active ball-handler system whose purpose is to dribble the ball throughout the game field in accordance with the RoboCup MSL rules. It has two blocks: 1) a DC motor to pull the ball towards the robot; 2) a rotation sensor, implemented as a quadrature incremental encoder, to measure the ball movement. This setup provides ball rotation feedback control, which allows setting a wide range of different ball speeds, independent of the robot motion.

The functions related to the kicking system are executed within the kicker module, without need for additional modules. The kicker interacts directly with the high-level layer through the gateway via CAN messages.

4.7. The system monitor

This function has two purposes, monitoring both battery voltage and modules run-time status. This latter purpose requires this function to be present in all modules, tracking reset situations, namely power-up reset, warm reset, brown-out reset (caused by undervoltage spikes) and watchdog reset, as well as answering to *I'm alive* requests issued by the high-level layer. However, tracking resets does not interfere with the remaining code being executed in each node since it is done before the actual functions are started, neither does handling *I'm alive* messages, which is neglectable.

On the other hand, battery monitoring is implemented in the same module as the kicker, since it already includes the specific voltage monitoring I/O. This function measures in real-time the voltage of the three NiMH batteries used in the robot, namely 2x12V for the *power* blocks of motor controllers and kicker, plus a 9.6V for the *logic* blocks. Particular care has been taken with the monitoring of the two *power* batteries, which is carried out using isolation amplifiers to maintain the galvanic isolation between *logic* and *power* blocks.

The information gathered by the system monitoring function, in all nodes, is sent to the high-level layer for remote monitoring and global coordination purposes.

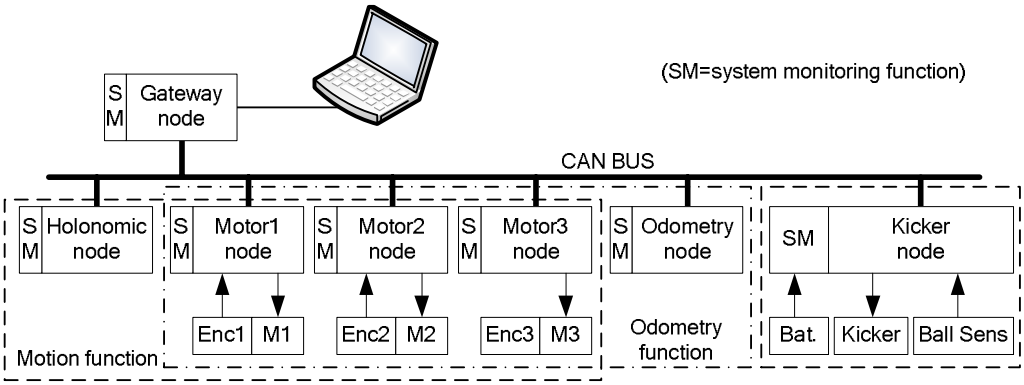

Fig. 4. Hardware architecture with functional mapping.

5. Main information flows and their synchronization

Beyond the task allocation to nodes, the way tasks are synchronized also has a substantial impact on the overall performance, namely in the end-to-end delays and jitter. There are two main paradigms concerning synchronization, either based on events, e.g., messages arriving from the CAN, interrupts from the hardware, other tasks terminating, etc., or based on time, e.g. at predefined instants.

The former method, normally referred to as *event-triggered*, is associated to a flexible execution of tasks whose start instant is not known in advance. Also, tasks with precedence constraints are normally triggered directly by their predecessors in a chain. This method tends to generate low end-to-end delays since all tasks related with one event are executed one after the other. However, the execution in sequence also causes inheritance and amplification of the tasks execution jitter, which is normally undesired for control purposes. Also, in multi-rate scenarios like the one in these robots, e.g. the motor controllers cycle at 5ms and the holonomic function cycles at 30ms, it is not possible to establish simple event chains, being necessary to use unsynchronized cycles that tend to cause rather large end-to-end delays and jitter.

The alternative is a *time-triggered* approach in which the periodic activities in the system are all synchronized and triggered with adequate relative offsets to absorb jitter of intermediate tasks in a transaction and reduce end-to-end delays. The CAMBADA robots use this approach, based on the FTT-CAN protocol (Flexible Time-Triggered communication over CAN) [23]. This protocol keeps all the information of periodic flows within a master node, implemented on another basic module, which works like a maestro triggering tasks and message transmissions [24].

The flows of information associated to the motion and odometry functions are shown in Figure 5, which also shows the synchronization framework and the relative offsets that were used to achieve low end-to-end latencies. Further details on the effective synchronization and on the explanation of Fig. 5 can be found in [5] but with larger execution times corresponding to a previous version.

The up arrows on the CAN bus represent the transmission of the trigger messages sent periodically by the FTT-CAN master, which control the execution of tasks in the nodes and the transmission of periodic (synchronous) messages on the bus. These control dependencies are illustrated with curved lines. The figure also illustrates the end-to-end latency (d_{ee}) for

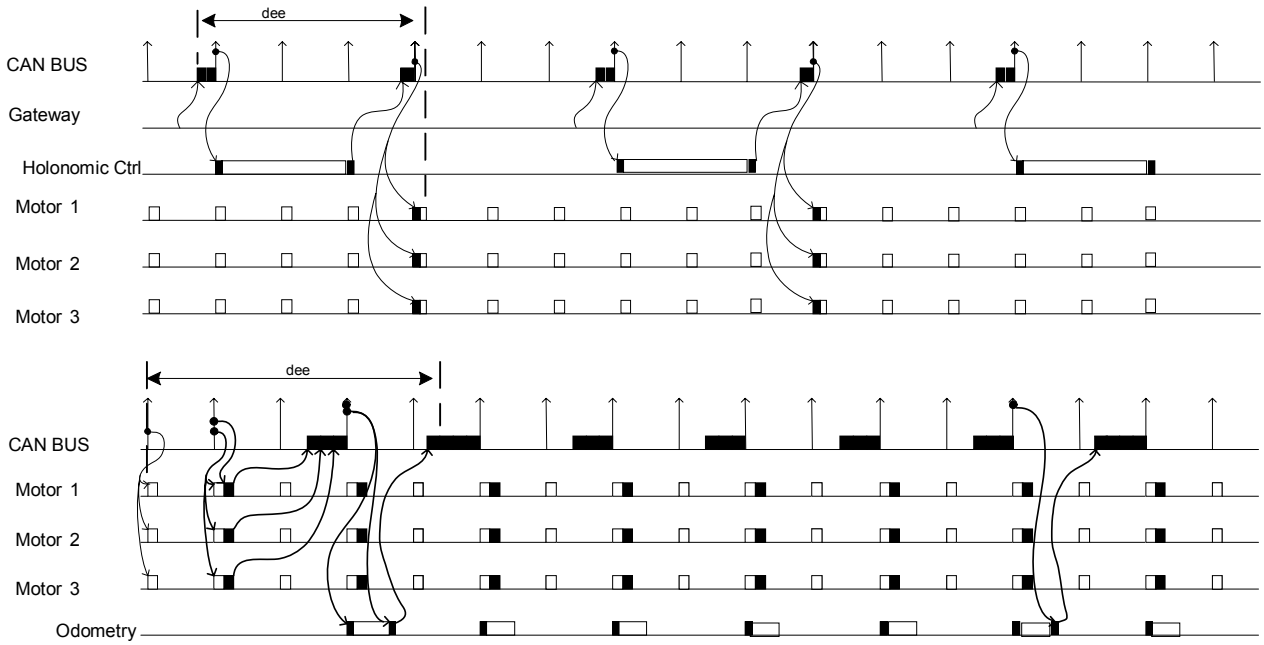

Figure 5. The information flows of the *motion* (top) and *odometry* (bottom) system functions. Notice that both flows are merged in the actual robot implementation.

each flow within the low-level layer, i.e. from the reception of a velocity vector set-point at the gateway until the respective motor setpoints are applied to the motor controllers (17ms), and from the reading of the encoders until the robot odometry information reaches the gateway (21ms).

6. Experimental results

In order to provide some quantitative evaluation of the architecture, we measured the activation jitter and the execution time jitter of the main functions, namely odometry, holonomic and the motor controllers (Table 1). These values are substantially shorter than one would obtain if such functions were executed in a PC, with all the interference caused by high-level coordination processes and operating system mechanisms. These results also show the decoupling of the activation jitter with respect to the execution jitter of the several functions, a direct consequence of the modularity of the architecture and the synchronization framework used.

	Period	Activation jitter	Execution time	Execution jitter
Holonomic	30ms	±15µs	9.6ms	±50µs
Odometry	10ms	±15µs	2.1ms	±100µs
Motor controller	5ms	±15µs	165µs	±25µs

Table 1. Activation jitter and execution time jitter of main functions.

7. Conclusion

For many years the software architectures of autonomous robots were discussed, with behavior-based becoming the most common ones, and relatively less attention was devoted to hardware architectures. However, when distributed hardware architectures became attractive in terms of cost, they became a possibility for use within robots, too, but a further step became necessary, i.e., allocating tasks to processors. On one hand, using distributed architectures may bring benefits in terms of composability, scalability, flexibility, maintainability and decoupling of concurrent functions and behaviors, on the other hand, a poor task to processor mapping may impact negatively on the reactive capabilities of the robot and thus must be carried out carefully.

This paper discussed the interest of using hierarchical distributed hardware architectures and presented the case of the CAMBADA middle-size robotic soccer team, being developed at the University of Aveiro, Portugal. These robots use a two-tiers architecture with a PC on top to execute the vision sensing and robot coordination, and a distributed system on the bottom, based on CAN, interconnecting the motors, encoders, kicker, dribbler and battery monitoring system. The paper described these subsystems as well as the allocation of their tasks to the processors. Some results were presented, namely concerning activation and execution jitter of the

various tasks, which illustrate the effectiveness of the architecture used.

8. Acknowledgement

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

References

[1] Kopetz, H., "Real-Time Systems Design Principles for Distributed Embedded Applications", Kluwer, 1997.

[2] Mataric, M. J., "Behavior-Based Robotics", in the MIT Encyclopedia of Cognitive Sciences, Robert A. Wilson and Frank C. Keil, eds., MIT Press, pp. 74-77, April 1999.

[3] Gates, B., "A Robot in Every Home", Scientific American, January 2007.

[4] L. Almeida, F. Santos, T. Facchinetti, P. Pedreiras, V. Silva, L.S.Lopes, "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.

[5] V. Silva, R. Marau, L. Almeida, J. Ferreira, M. Calha, P. Pedreiras, J. Fonseca, "Implementing a distributed sensing and actuation system: The CAMBADA robots case study", IEEE ETFA 2005, Catania, Italy. September 2005.

[6] Mock, M., Nett, E., "Real-Time Communication in Autonomous Robot Systems", Proc. 4th Int. Symp. on Autonomous Decentralized Systems, 1999, Integration of Heterogeneous Systems, 21-23 March 1999, pp. 34-41

[7] Cavalieri, S., Stefano, A., Mirabella, O., "Impact of Fieldbus on Communication in Robotic Systems", IEEE Transactions on Robotics and Automation, Vol. 13, N. 1, February 1997.

[8] Valera, A., Salt, J., Casanova, V., Ferrus, S., "Control of Industrial Robot With a Fieldbus", Proc. 7th IEEE Int. Conf. on Emerging Technologies and Factory Automation. ETFA '99. Vol. 2, 18-21 October 1999.

[9] Controller Area Network - CAN2.0, Technical Specification, Robert Bosch, 1992.

[10] Kongezos, V., Allen, C.R., "Wireless Communication between A.G.V.'s (Autonomous Guided Vehicle) and the insdustrial network C.A.N. (Controller Area Network)", Proc. 2002 IEEE Int. Conf. on Robotics & Automation Washington, DC, May 2002.

[11] J. L. Posadas Yagüe, P. Pérez, J. Simó, G. Benet and F. Blanes, "Communications structure for sensory data in mobile robots", Engineering Applications of Artificial Intelligence 15, pp341-350, 2002.

[12] P.Pérez, G.Benet, F. Blanes, J.E. Simó, "Communication Jitter Influence on Control Loops Using Protocols for Distributed Real-Time Systems on CAN bus", Proc. of IFAC SICICA 2003, Aveiro, Portugal, July 2003.

[13] Proc. of the NASA Workshop on Biomorphic Robotics, Jet Propulsion Laboratory, California Institute of Technology, USA, 2000.

[14] RTAI for Linux, available at http://www.aero.polimi.it/~rtai/

[15] F. Santos, L. Almeida, P. Pedreiras, L.S.Lopes, T. Facchinnetti, "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.

[16] Microchip website, available at www.microchip.com

[17] Giorgio Buttazzo, "Hard Real-Time Computing Systems: Predictable Scheduling Algorithms And Applications", Second Edition, Springer, 2005.

[18] Weidong Chen, Qixin Cao, Jingchuan Wang. JiaoLong2007 Team Description. Available on-line at http://robocup.sjtu.edu.cn/robocup/index.htm

[19] P. Lima, H. Costelha, J. Estilita, N. Martins, G. Neto, J. Santos. ISocRob 2006 - Team Description Paper. Available on-line at http://socrob.isr.ist.utl.pt/omnis2006.php

[20] J. Rushby, "A Comparison of Bus Architectures for Safety-Critical Embedded Systems", CSL Technical Report, September 2001.

[21] D. T. Peng, K. G. Shin, T. Abdelzaher, "Assignment and Scheduling Communicating Periodic Tasks in Distributed Real-Time Systems", IEEE Trans. on Software Engineering 23 (12):745 - 758, December 1997.

[22] Alexander Metzner, Martin Fränzle, Christian Herde, Ingo Stierand, "Scheduling Distributed Real-Time Systems by Satisfiability Checking," rtcsa, pp. 409-415, 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05), 2005.

[23] Almeida L., P. Pedreiras, J. A. Fonseca, "The FTT-CAN Protocol: Why and How, IEEE Transactions on Industrial Electronics", 49(6), December 2002.

[24] Calha, M.J., J.A. Fonseca, "Approaches to the FTT-based scheduling of tasks and messages", Proceedings of the 5th IEEE International Workshop on Factory Communication Systems (WFCS'04), Vienna, Austria, Sep/2004.