Implementing a distributed sensing and actuation system: The CAMBADA robots case study

V. Silva[1,2], R. Marau[2], L. Almeida[2], J. Ferreira[2,3], M. Calha[2,3], P. Pedreiras[2], J. Fonseca[2]
vfs@estga.ua.pt, {marau, lda}@det.ua.pt, {jjf, mjc}@est.ipcb.pt, {pedreiras, jaf}@det.ua.pt

[1]Universidade de Aveiro
Esc. Sup. Tecnologia e Gestão
3754-909 Agueda

[2]Universidade de Aveiro
LSE-IEETA / DET
3810-193 Aveiro

[3]Inst. Polit. Castelo Branco
Escola Sup. de Tecnologia
Avª do Empresário
6000-767 Castelo Branco

Abstract

The use of distributed computing architectures has become commonplace in complex embedded systems with potential advantages, for example, in terms of scalability, dependability and maintainability. One particular area in which that trend can be witnessed is mobile autonomous robotics in which several sensors and actuators are interconnected by means of a control network. In this paper we address one case study concerning the CAMBADA robots that were developed at the University of Aveiro for the Robocup Middle Size League. These robots have a distributed architecture with two layers, a coordination layer responsible for the global behaviors and a distributed sensing and actuating layer that conveys internal state information and executes coordination commands. This paper focuses on the latter layer, which is based on the FTT-CAN protocol, following a network-centric approach that provides an efficient framework for the synchronization of all systems activities. We describe the computing and communication requirements, the robot architecture, the system design and implementation, and finally we provide experimental results that show advantages with respect to a non-synchronized distributed approach.

1. Introduction

Distributed Embedded Systems (DES) are typically part of intelligent automatic equipment with a high degree of autonomy. In most cases, DES have a strong impact on human lives, either because they are used within important economic processes, e.g. complex machinery in factories, or because they control equipment that directly interacts with people, e.g. transportation systems [1].

The importance of DES has been growing steadily and it is expected to grow even further as distribution provides an efficient way to improve several desirable properties in a system, from maintainability, to scalability, composability and dependability, to name a few [2] [3]. Also, DES are a natural support for higher integration of resources in complex systems, e.g. robots, cars and planes, with a potential for lower costs and lower overall complexity [4].

However, the positive aspects of distribution do not come for granted and specific techniques and protocols must be used to achieve the desired properties. Therefore, designing and deploying such techniques and protocols is still an important research topic [1].

The control of robots, particularly autonomous mobile robots, is one of the application fields where DES have been increasingly used, seeking for cabling reductions and simplification, improved maintainability, fault-tolerance, scalability of functionality, etc.. In this paper we address a specific case study that concerns the CAMBADA robots developed at the University of Aveiro for participation at the RoboCup Middle Size League. These robots have a low level distributed sensing and actuation system based on Controller Area Network (CAN) that interconnects the motor drives, the movement controllers, the odometry system and other subsystems detailed later. This work focuses on the communication and synchronization of activities, which is carried out using the FTT-CAN [5] protocol. We show how to implement an application on top of this protocol as well as some of the benefits that arise from its use with respect to other communication alternatives based on non-globally synchronized frameworks.

The paper is structured as follows. Section 2 presents some related work, section 3 shows the general architecture of the CAMBADA robots while the respective communication and computation requirements are analyzed in section 4. Section 5 addresses some relevant implementation issues, mainly those concerning the use of the communication system and the synchronization of activities across the distributed

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.

system. Finally section 6 presents preliminary experimental results while section 7 concludes the paper.

2. Related Work

As referred before, there are several advantages that may arise from the use of distributed architectures in embedded control systems and such a distributed approach has been often used in the specific field of mobile and autonomous robotics for diverse application scenarios. For example, [6] presents a robot for orange picking that is divided into 4 platforms, each one with two picking arms. An SP50 (later Foundation Fieldbus FF-H1) fieldbus is used to provide connectivity between the four platforms and support the required data exchanges. [7] presents an industrial robot based on a ProfiBus network. The authors simulate the system operation using Matlab/ Simulink, and measure the communication delays and level of synchrony achieved among the activities carried out within the robot.

One particular protocol that has been substantially used within mobile robots is CAN [8] due to its low price, good reliability and timeliness properties. Examples of using this protocol can be found in [9], [10] [11]. The latter one is particularly relevant to this work as it addresses the concerns of supporting a distributed sensing and actuation system integrated in a more complex architecture encompassing a deliberative level that extends beyond the robot using a TCP/IP connection with an adequate temporal firewall to isolate this level from the lower one in which real-time constraints are tight. In [12] the same authors discuss the impact that the communication jitter of real-time data transfers can have on the performance of control closed-loops and propose a mixed CAN-based event/time-triggered protocol.

The control architectures referred above either use event-triggered approaches that present poor control over the communication jitter given the absence of relative offsets, or they use time-triggered approaches for the periodic traffic specified in a static way. In this work we address the issues arising from the use of FTT-CAN [5] to support the distribution of low level sensing and actuation information. This protocol provides support for both event and time-triggered traffic as well as support for flexible time-triggered communication, allowing to adapt the rates of the periodic communication on-line according to the instantaneous needs. [13] shows the interest of providing dynamic rate adaptation of the periodic information in a mobile robot but using a centralized architecture. Our work allows extending those benefits to a distributed framework.

3. General architecture

The general architecture of the CAMBADA robots has been described in [14]. Basically, the robots follow a biomorphic paradigm, 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).

At the heart of the coordination layer is the Real-Time Database (RTDB) that contains both the robot local state information as well as local images of a subset of the states of the other robots. A set of processes update the local state information with the data coming from the

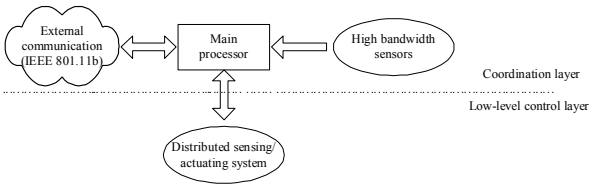

Figure 1. The biomorphic architecture of the CAMBADA robots.

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).

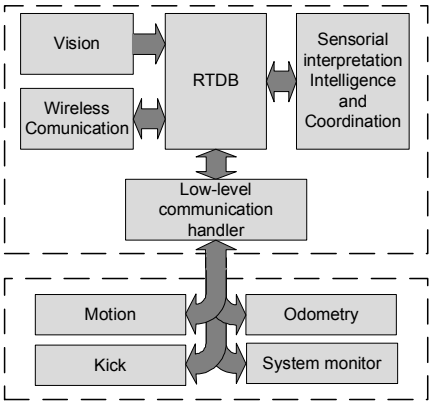

Figure 2. The robots functional architecture built around the RTDB.

The low-level sensing/actuating system follows the fine-grain distributed model [2] 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 nodes are based on the PIC microcontroller 18Fx58 [15] operating at 40MHz while the network uses the CAN protocol with a bit rate of 250Kbps.

At this level there are 3 DC motors with respective controllers plus an extra controller that, altogether, provide holonomic motion to the robot. Each motor has an incremental encoder that is used to obtain speed and displacement information. Another node is responsible for combining the encoder readings from the 3 motors and building a coherent displacement information that is then sent to the coordination layer. Moreover, there is a node responsible for the kicking system that consists of a couple of sensors to detect the ball in position and trigger the kicker. This node also carries out battery voltage monitoring. Finally, the low-level control layer is interconnected to the coordination layer by means of a gateway attached to the serial port of the PC, configured to operate at 115Kbaud. From the perspective of the low-level control layer, the higher coordination layer is hidden behind the gateway and thus, we will refer to the gateway as the source or destination of all transactions arriving from or sent to that layer.

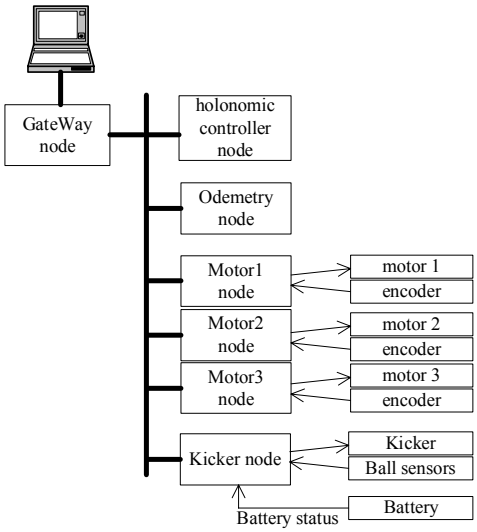

Figure 3. The hardware architecture of the low-level control layer.

4. Lower-level requirements

In the previous section we have identified the functional and hardware architectures of the low-level control layer. The specific mapping of the former over the latter generates the operational architecture which presents requirements concerning both the tasks that need being executed on each node as well as the messages that must be exchanged over the network. In this section we will analyze in detail these requirements which were used for the actual implementation. In particular, the communication requirements are shown in Table 1.

The *Motion* function depicted in Figure 2 spans across 4 nodes, the 3 motor controllers plus the holonomic controller that translates the robot velocity vector set-point received from the upper layer into individual speed set-points for each of the motors. Both the motor controllers as well as the holonomic controller execute in a periodic fashion but with different periods. The former ones execute a PI-type closed-loop motor speed control once every 5ms. This value has been deduced from the dynamics of the robot. Moreover, these tasks are relatively light, taking less than 1 ms to accomplish. On the other hand, the holonomic controller executes a cyclic conversion of the higher layer set-points once every 30ms. This node is relatively loaded as each conversion takes about 16ms to carry out. The chosen period is, nevertheless, sufficiently small to support a smooth robot motion.

In terms of communication the *Motion* function requires the periodic transfer of the robot velocity vector set-point from the gateway to the holonomic controller and then the periodic transfer of the motor speed set-points from the holonomic controller to the individual motor controllers. Both transfers are carried out once every 30ms. The former transfer requires two messages (M6.1, M6.2) to convey the linear and angular information respectively. Concerning the latter transfer, the motor speed set-points generated for the motor controllers should be applied to each motor approximately at the same time thus they are piggybacked on the same message and transferred as a broadcast (M1). Finally, the control loops of the 3 motor controllers should also be synchronized among themselves so that they generate motor actuation signals at approximately the same time.

Another important subsystem is the one corresponding to the *Odometry* function. This function also spans across 4 nodes, the 3 motor controllers plus a 4^{th} node that combines the individual encoder readings into a coherent displacement information sent up to the higher layer. The encoder readings are the same as used by the closed-loop motor speed control and thus they are sampled every 5ms, and this should be carried out synchronously in all three motors. However, depending on the desired precision in constructing the robot displacement information, these readings can be sent with a periodicity that varies from 5ms to 20ms (higher to lower precision). During the execution of certain high level behaviors the odometry information is not needed, e.g. when tracking the ball, and thus it can also be temporarily switched off. Three messages are used to convey the encoder readings (M3.1-M3.3). Upon reception of these messages, the odometry node calculates the robot position and orientation, taking approximately 4ms, and sends it to the higher layer, every 50ms, using 2 messages (M4.1, M4.2). This period is compatible with the cycles used by the processes running within the higher layer. The *Odometry* function also includes a pair of sporadic messages (M5.1, M5.2) received from the higher layer to set or reset the current

ID	Source	Target	Type	Period/mit (ms)	Size (B)	Short description
M1	Holonomic ctrl	Motor node[1:3]	Periodic	30	6	Aggregate motor speeds set points
M2	Kicker	Gateway	Periodic	1000	2	Battery status
M3.1-M3.3	Motor node [1:3]	Odometry node	Periodic	5 to 20	3*3	Wheels encoder values
M4.1-M4.2	Odometry node	Gateway	Periodic	50	7+4	Robot position + orientation
M5.1-M5.2	Gateway	Odometry node	Sporadic	500	7+4	Set/reset robot position + orientation
M6.1-M6.2	Gateway	Holonomic ctrl	Periodic	30	7+4	Velocity vector (linear+angular)
M7	Gateway	Kicker	Sporadic	1000	1	Kicker actuation
M8-M12	Every node	Gateway	Sporadic	1000	5*2	Node hard reset

Table 1. Low-level control layer communication requirements

robot position and orientation information within the odometry node. These messages are not expected to be generated within less than 500ms intervals (minimum inter-arrival time – *mit*).

Finally, the *Kick* and *System monitor* functions are integrated in the same node, the kicker controller, which is lightly loaded. The former corresponds to executing the kicking commands received from the higher layer. These are conveyed within one sporadic message (M7) which is not expected to be transmitted more often than once every second. In fact, the kicker is electromagnetic and takes about this time to recharge between consecutive kicks. On the other hand, the latter function currently encompasses the batteries level sampling which is sent up to the higher layer using a periodic message (M2) with a period of 1s, as well as a set of 5 sporadic messages (M8-M12) that inform the higher layer whenever a hard reset occurs in the respective node.

5. Low-level control layer implementation

After having defined the operational architecture of the low-level control layer and deduced the computing and communication requirements the practical implementation was carried out. Two approaches have been followed, one without and another with global synchronization among the activities executed at this layer. The former approach used communication functions of the type *send* and *receive*, as commonly found in event-triggered systems, and without any further support for synchronizing remote activities. At fixed points within the respective cycle the data would be transmitted using the *send* function and retrieved at the receiver with the *receive* function.

The temporal behavior of the approach referred above may suffer large delays due to the multiple unsynchronized chained cycles. For example, consider that a new velocity vector arrived at the holonomic controller right after it started processing one cycle. Then, the new vector would be processed one cycle later, generating speed set-points for the motors with about 30ms additional delay, i.e., the cycle time of the holonomic controller. These set-points would then be transmitted over the network within one message, possibly suffering an access delay caused by possible transmissions from other unsynchronized nodes. Finally, this message would arrive at a motor node right after this node had started one speed control cycle thus holding the new set-point until the next cycle causing a further delay of 5ms. Comparing with the case in which the new data would arrive just before the start of the cycle in which it would be used, i.e. the best-case delay, the previous situation corresponds to an additional delay of more than 35ms to process a new velocity vector. Figure 4 illustrates the impact of chained non-synchronized cycles on the end-to-end delay (d_{ee}) for the general case of two periodic tasks in different nodes, A and B, which communicate via a periodic message.

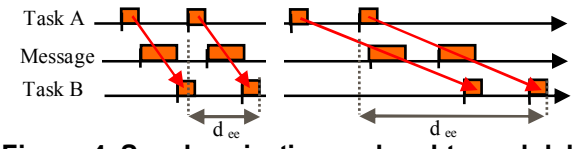

Figure 4. Synchronization and end-to-end delay.

Moreover, this delay can vary on-line due to drifts in the local clocks of the nodes, generating jitter in the control signals. These additional delays and jitter can cause degradation to the global control loops associated to high level behaviors, such as tracking the ball.

On the other hand, this non-synchronized approach has the advantage of being very simple to deploy. For this reason, it was the first approach to be implemented. As expected, the parameters of the global control loops were relatively difficult to tune and a *nervous* robot behavior was frequently observed.

Therefore, it was decided to use a communication infrastructure based on CAN that would allow building a globally synchronized framework so that relative phases among all activities in the system, including tasks execution in the nodes and message transfers over the network, could be established as appropriate to maintain the end-to-end delays of the information flows under tight bounds. The FTT-CAN [5] protocol was used for this purpose, which supports global synchronization of tasks and messages according to a network-centric approach [16]. Moreover, it combines the time-triggered (synchronous) traffic with event-triggered (asynchr-onous) traffic providing an efficient support to both periodic and sporadic messages. Another feature, and probably its most distinguishing one, is that the protocol

supports on-line changes to the synchronous traffic, allowing to switch off and on any message stream according to current needs or even to adapt the rate, for example, to control the provided Quality-of-Service.

In order to use FTT-CAN two more nodes were added to the low-level control layer to perform the Master function with replication for fault-tolerance purposes [17][18]. The following section makes a brief presentation of the basic concepts and operational aspects of FTT-CAN.

5.1. FTT-CAN basics

The FTT-CAN protocol (Flexible Time-Triggered communication over CAN) is a time triggered protocol that establishes a common notion of time across the system using the regular transmission of a particular synchronization message by a specific node called Master. This message is called the Trigger Message (TM) and its periodic transmission creates fixed duration bus time slots called Elementary Cycles (EC). Each EC is divided in two phases, one for the transmission of time-triggered traffic, synchronously with respect to the ECs framework, and another for the transmission of event-triggered traffic, within which transmissions can take place at any instant. These phases are called the synchronous and asynchronous windows, respectively, with the latter preceding the former (Figure 5). A gap is used in between these windows to guarantee that the asynchronous traffic does not interfere with the synchronous one.

According to the needs of each application, the maximum duration of the synchronous window can be bounded (LSW), leaving a minimum bandwidth always available to the aperiodic traffic, improving its responsiveness.

The Master node controls the transmission of the synchronous traffic whose periods and offsets are expressed as integer multiples of the EC duration. A master/multi-slave approach is used in which the Master sends one command per EC, possibly triggering several synchronous transmissions in the respective phase of that EC. The Master triggering commands, called EC-schedules, are conveyed within the TMs using a specific bit encoding technique (Figure 5). The EC-schedules may also include specific flags to trigger the execution of tasks within the nodes, synchronously with the ECs framework. These flags allow remote tasks to synchronize with each other as well as with the transmission of messages and using system wide offsets.

The scheduling of the synchronous part of the system, including task triggers and synchronous messages, is carried out by the Master node, on-line, based on a table that contains the synchronous requirements. This is called the Synchronous Requirements Table (SRT) and it can be updated on-line, granting the operational flexibility that characterizes this protocol.

The EC duration is configured off-line and has a significant impact on the communication and computing overhead of the system. Basically, the shorter it is the higher the overhead but also the higher the temporal resolution to express the messages properties. Therefore, the common technique is to use the shortest period as long as the corresponding overhead is admissible. Typical values range from 1 to 10ms.

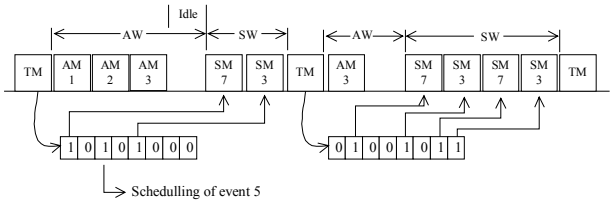

Figure 5. FTT-CAN transmission schema.

5.2. Implementation using FTT-CAN

In order to effectively use FTT-CAN in a given application it is necessary to identify the flows of information related with cyclic activities executed in the system, to determine what triggers each of those flows and then to determine which should be the appropriate offset of each transmission or activity knowing the respective transmission and execution times. This has been carried out in the previous section where the information flows within the low-level control layer of the CAMBADA robots were identified and characterized. In this section we will see how FTT-CAN was used to support the required synchronization.

The first aspect is to separate the periodic from the sporadic traffic. The latter is handled by the asynchronous subsystem similarly to a non-synchronized framework. This separation is already accomplished in Table 1. The periodic traffic is then named using FTT-CAN synchronous identifiers.

The EC duration is set to 5ms which is the shortest period among all periodic activities and messages, i.e., the closed-loop motor speed control period. For a trigger message with 5 bytes, the communication overhead is lower than 8.4% (420μs/5ms) while the computing overhead is close to 1.5% (76ms/5ms). These values were considered admissible given the application load. Particularly, the communication load according to Table 1 is close to 27% of the bus bandwidth at 250Kbps.

Knowing the EC duration, all periods are expressed in number of ECs. Then, the synchronization requirements are analyzed to identify the set of activities that needed synchronization and the respective set of synchronous triggers. These inherit periods equal to those of the related messages and are also named using appropriate FTT-CAN identifiers.

Finally, the off-sets of all messages and synchronous triggers are established so that transmissions are carried out soon after the respective data becomes available and,

conversely, activities are triggered enough in advance to generate data before but as close as possible to the respective transmission instant. Moreover, the synchronous triggers allow triggering several remote activities at approximately the same time (within a few micro-seconds), as it is required by the *Odometry* function. These concerns lead to increase the freshness of the data in the information flows, reducing the respective end-to-end latency and jitter, with a positive impact in the performance of the respective global control loops associated to the high level behaviours.

Table 2 shows the system SRT, including both synchronous messages and triggers. The off-sets extracted from the system requirements are expressed in the column *init time* and they are also expressed in number of ECs.

Figure 6 shows the timeline of the two main synchronous information flows, separately, associated to the *Motion* function (top) and the *Odometry* function (bottom). In what concerns the *Motion* function, the flow is triggered by a pair of messages (6,7) sent by the gateway with off-set 0 and arriving from the higher layer with a velocity vector. These values are received by the holonomic controller that is synchronized in trigger 12, which is produced right after the transmission of the messages, with off-set of 1EC. This trigger starts the execution of the holonomic controller to process the new velocity vector. The resulting motor speed set-points will be available after 16ms, which rounds up to 4 ECs. Thus the respective message (0) is transmitted to the motor nodes in the following cycle, i.e. with an off-set of 5ECs. Trigger 13 is used to synchronize the closed-loop speed control of each motor with the arriving set-point. The off-set is 6ECs to enforce a reduced latency between reception and use of the set-points.

The transmission of the next velocity vector, and thus the start of the next cycle, is carried out in the following EC.

In what concerns the *Odometry* function, the respective information flow starts with trigger 8, with off-set 0, which causes the synchronous sampling of the encoders in the 3 motors. These values are locally accumulated until they are transmitted. In the example, the transmission of the encoder readings is set to 2 ECs (messages 1-3) and the respective values are produced with trigger 9, in the EC before their transmission. Thus the off-set of messages 1-3 is 2ECs while the off-set of trigger 9 is 1EC. The periods of these entities can vary depending on the desired odometry precision from 1EC (highest) to 4 ECs (lowest). They can also be suspended (period set to 0) when the *Odometry* function is not needed.

The odometry node is triggered right after the transmission of the messages 1-3 carrying the encoder readings, using trigger 10 with an off-set of 3 ECs. Since it executes in less than one EC, the production of the current position and orientation (trigger 11) is carried out in that EC (same off-set of 3 ECs) while the message transmissions (messages 4,5) are assigned to the following EC thus with an off-set of 4 ECs.

6. Experimental results

In order to assess the benefits of using FTT-CAN a few experiments were carried out, comparing the system timeliness with the case in which CAN was used without support for synchronization among remote tasks. For that comparison, we measured the end-to-end delay

FTT-CAN ID	Source	Destination	Period (#ECs)	Init time (#ECs)	Short description
0	Holonomic contr	Motor node[1:3]	6	5	Motors speed setpoints
1	Motor 1 node	Odometry node	2 (0-4)	2	Encoder Count in motor 1
2	Motor 2 node	Odometry node	2 (0-4)	2	Encoder Count in motor 2
3	Motor 3 node	Odometry node	2 (0-4)	2	Encoder Count in motor 3
4	Odometry node	Gateway	10	4	Current position
5	Odometry node	Gateway	10	4	Current orientation
6	Gateway	Holonomic contr	6	0	Velocity vector (linear)
7	Gateway	Holonomic contr	6	0	Velocity vector (angular)
8	---	Motor node[1:3]	1	0	Triggers the encoder readings
9	---	Motor node[1:3]	2	1	Triggers production of messages 1,2,3 at the motor nodes (encoder readings)
10	---	Odometry node	2	3	Triggers the consumption of encoder messages 1,2,3 at the odometry node
11	---	Odometry node	10	3	Event to produce messages 4,5
12	---	Holonomic contr	6	1	Triggers the consumption of Messages 6,7 in holonomic controller
13	---	Motor nodes [1:3]	6	6	Triggers the consumption of Message 0 in the motor nodes

Table 2. Low-level control layer message set and activity triggers.

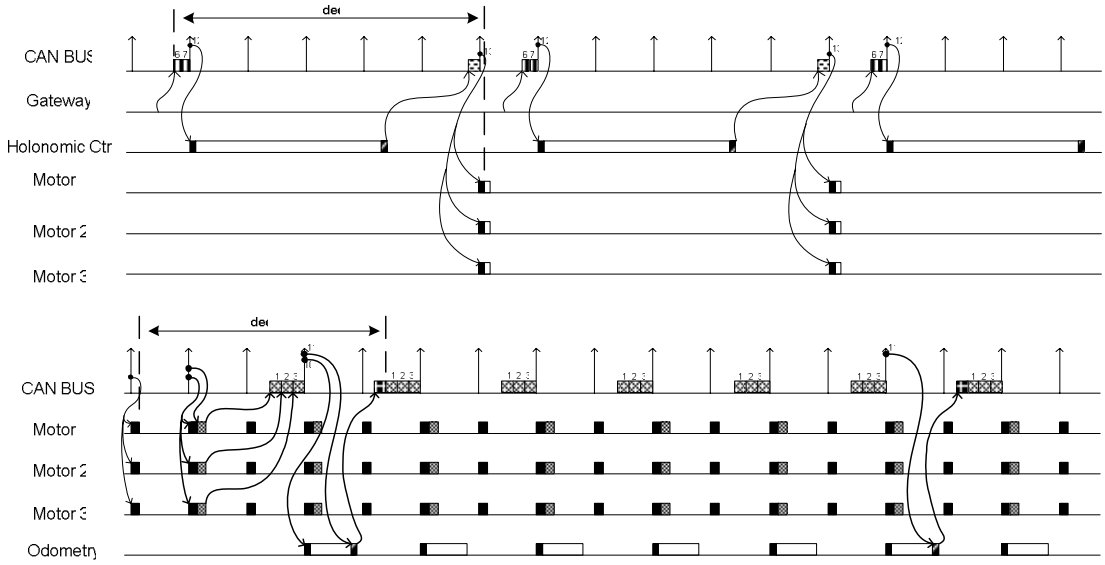

Figure 6. Timeline of the main information flows within the low-level control layer. Top: *motion*. Bottom: *odometry*.

associated with the two information flows referred before, i.e. *Motion* and *Odometry* functions. The first flow was measured from the point in which the gateway starts transmitting a velocity vector to when one of the motors receives the corresponding speed set-point. The second flow was measured from the point in which the encoder of one motor is read to when the respective new position is received by the gateway. The results are presented in Table 3, concerning the maximum and minimum values observed for the end-to-end delays (d_{ee}) of both information flows in the two approaches referred before, i.e. unsynchronized using CAN and globally synchronized using FTT-CAN.

Information flow	CAN		FTT-CAN	
(all values in ms)	max d_{ee}	Min d_{ee}	max d_{ee}	min d_{ee}
Motion	64.4	38.8	27.7	26.7
Odometry	21	12	21.7	21.6

Table 3. Timeliness of information flows.

These results are according to expected as stated in section 5. In fact, the absence of synchronization between multiple chained cycles creates large delays and, mainly, large delay variations (jitter). On the other hand, the synchronization capabilities of FTT-CAN allow establishing adequate off-sets that can be used to reduce end-to-end delays, and mainly the associated jitter. The former effect, however, i.e. reduction of end-to-end delays, is only noticeable when the cycle durations are large enough, at least 3 ECs long. For shorter cycles, as it is the case with the *Odometry* information flow, the temporal resolution of FTT-CAN limits the achievable reduction in the end-to-end delay. However, there is still a strong reduction in jitter, nearly elimination, which is probably more beneficial for control purposes than the limitation on the end-to-end delay reduction.

Another advantage of FTT-CAN is the easiness and efficiency in triggering tasks with synchronous triggers. In fact this is done without extra messages, just using the Trigger Message with additional data bits to encode them. These triggers also allow synchronizing tasks in remote nodes with relatively high precision. In the specific case of the closed-loop speed control in the 3 motors, the use of triggers allowed to synchronize all the loops within +/-130μs.

7. Conclusions

Distributed embedded systems are becoming pervasive, spanning application fields in which there are stringent real-time and safety constraints. Such architectural option has a potential for several advantages, the most important of which is, probably, the constraining of complexity within manageable bounds.

The control of robots, particularly autonomous mobile robots, is one of the application fields where distributed embedded systems have been increasingly used, seeking for cabling reductions and simplification, improved maintainability, fault-tolerance, scalability of functionality, etc.. However, increasing requirements in terms of responsiveness, arising from complex environments with fast dynamics, stress the need for adequate architectural support so that robots cope with sudden events that occur in the environment and respond to them in a timely way. One particular application that has been driving the need for more reactivity is RoboCup where teams of mobile robots play football. Robots need to move faster without colliding with each other or against the field objects and they must react promptly to the ball with sufficiently

accurate control. This is only possible if the respective control infrastructure is timely, in spite of its complexity.

In this paper we described the architecture of the CAMBADA robots developed at the University of Aveiro to participate in the RoboCup Middle Size League. The paper focused on the robots distributed low-level control layer that interconnects the motion, odometry, kicking and monitoring subsystems. The computing and communication requirements were deduced, and then the paper focused on the implementation of this layer using the FTT-CAN protocol. The performance of this implementation was compared with a previous implementation based on CAN in which there was no control over the synchronization of remote activities and chained control cycles. Experimental results were shown where the benefits of using a globally synchronized framework, such as provided by FTT-CAN, were clear. The jitter in the end-to-end delay of the main information flows was nearly eliminated and, in the case of the motion control, the end-to-end delay was substantially reduced.

Future work will address the dynamic reconfiguration capabilities supported by FTT-CAN so that the rate of the information flows is adapted on-line according to the instantaneous needs. This will allow maximizing the bandwidth available to the asynchronous communication, both to react promptly to asynchronous events or commands and to allow time for retransmissions upon errors. The released bandwidth can also be used to allow the insertion of more subsystems for complementary functionalities.

The paper focused on the robots distributed low-level control layer, only. However the control loop includes components in both upper and low-level layers (Figure 2). Therefore, the robot global performance can be further enhanced with the use of global synchronization mechanisms, allowing minimizing the overall end-to-end latency. This topic will also be addressed in future work.

References

[1] *Embedded Systems Design*, Vol. 3436, 2005, ISBN: 3-540-25107-3

[2] Kopetz, H. Real-Time Systems: Design Principles for Distributed Embedded Applications, Kluwer Academic Publishers, 1997.

[3] P. Koopman. Critical Embedded Automotive Networks. *IEEE Micro*, IEEE Press, July/August 2002.

[4] Rushby, J., Bus Architectures For Safety-Critical Embedded Systems, in Proceedings of the First Workshop on Embedded Software, *Lecture Notes in Computer Science vol. 2211*, pp 306-323, 2001.

[5] Almeida, L., Pedreiras, P., Fonseca, J.A.G., "The FTT-CAN protocol: Why and how", IEEE Transactions on Industrial Electronics, Volume 49, Issue 6, Dec. 2002, pp. 1189-1201

[6] 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.

[7] 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.

[8] Bosch, Robert [1991], CAN Specifications Version 2.0, BOSCH, Stuttgart

[9] 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

[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] G. Beccari, C. Caselli, M. Reggiani, F. Zanichelli, "Rate Modulation of Soft Real-Time Tasks in Autonomous Robot Control Systems", Proc. 11th Euromicro Conf. on Real-Time Systems, RTS'99, York, UK, June 1999.

[14] Almeida, L., Santos, F., Facchinetti, T., Pedreiras, P., Silva, V., Lopes, L., "Coordinating distributed autonomous agents with a real-time database: The CAMBADA project", Proc. ISCIS 2004 (19th Int. Symp. on Computer and Information Sciences, Kemer-Antalya, Turkey, October 27-29, 2004.

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

[16] M.J. Calha, J.A. Fonseca - "*Adapting FTT-CAN for the joint dispatching of tasks and messages*", WFCS'02 - 4th IEEE Int Workshop on Factory Communication Systems, Västerås, Sweden, August 27-30, 2002

[17] E. Martins, J. Ferreira, L. Almeida, P. Pedreiras, J.A. Fonseca. An Approach to the Synchronization of Backup Master in Dynamic Master-Slave Systems. Work-in-Progress Session of RTSS 2002, IEEE 23rd Real-Time Systems Symposium, Austin, USA, December 2002.

[18] J. Ferreira, L. Almeida, E. Martins, P.Pedreiras, J.A. Fonseca, "*Enforcing Consistency of Communication Requirements Updates in FTT-CAN*", Workshop on Dependable Embedded Systems, SRDS2003, 22nd Symposium on Reliable Distributed Systems, Florence, Italy, 6-8 October, 2003