Preprints of the 18th IFAC World Congress
Milano (Italy) August 28 - September 2, 2011

Modular Low-Cost Robotics: What Communication Infrastructure?

Andrea Bonarini * Matteo Matteucci * Martino Migliavacca *
Roberto Sannino ** Daniele Caltabiano **

* *Dept. of Electronics & Information, Politecnico di Milano, Italy (e-mail: {bonarini, matteucci, migliavacca}@elet.polimi.it).*

** *STMicroelectronics, Via C. Olivetti 2, Agrate Brianza (MB), Italy (e-mail: {roberto.sannino, daniele.caltabiano}@st.com).*

Abstract: In this paper we present a methodological approach for the development and fast prototyping of robotics applications targeted to the mass market. According to this approach, a robot is considered as a distributed hardware and software system, in which the components are "smart devices", i.e., hardware modules with onboard computation. Our proposal aims, through a modular architecture, at massive hardware/software reuse and fast prototyping. We suggest to consider a robotic application as a distributed control system, where a set of basic modules (e.g., sensors, actuators, controllers) is interconnected in a plug-and-play way by means of a common physical communication interface. The requirements of such physical interface are elicited in the paper, starting from three real case studies, and a discussion about a proposed solution is provided.

Keywords: Autonomous mobile robots, Embedded systems, Communication channels, Sensors, Actuators, CAN bus.

1. INTRODUCTION

Robotics is an active engineering field that, since years, is expected to enter the mass market (Gates, 2007), but, up to now, it did not take off as it was expected. The main problems are related to the cost of such systems, not only in terms of hardware, but also as time and resources needed to design and prototype a single robot.

To speed up this process, a novel design approach is needed to enable hardware/software reuse and fast prototyping. This, in turns, calls for a modular architecture, where off-the-shelf basic modules (e.g., sensors, actuators, controllers) are combined together in a plug-and-play way to build a complex system. At the same time, a different design methodology is required which, on one side, identifies the needed modules starting from the application specification, while, on the other, is able to design components generic enough to be effectively reused.

The goal of this paper is twofold. First, we are interested in presenting a methodological approach to define the architecture of a system, through the analysis of its *functional* and *non-functional* requirements instead of relying on what is available on the market in terms of platforms or software frameworks. Second, we would like to move a step forward in the definition of a modular architecture by defining the requirements of an open standard infrastructure to build its modules upon.

Section 2 describes the modular design approach we are proposing in details, and Section 3 reviews the actual state of the art in relation to our approach. The analysis of three case studies for the design approach we are presenting are discussed in Section 4. These case studies are analyzed through functional and non-functional decomposition of the respective applications, with the aim of selecting one communication channel that could make the modules interacting. Finally, in Section 5, we discuss under which conditions the CAN field bus specification could match the requirements previously defined and thus it could be used to base robotic application design now on, as it has been done for the automotive industry.

2. THE OTHER WAY AROUND

Robotic systems have been often developed starting from existing devices; these might have been developed for other applications for which expected cost and performance requirements are significantly different from those of a service robot market (e.g., SICK or Hokuyo laser range finders). To speed up the development of robots, while reducing the overall cost from the idea to the market product, a modular approach is recommended, enabling the reuse of hardware and software. This approach is also suggested by the idea of robot itself: a distributed system of hardware (e.g., sensors and actuators) and software (e.g., sensor filtering and conditioning, control loops, planning algorithms, etc.) modules that needs to interact with the environment in order to reach a given goal. From this perspective, hardware and software need to be developed together, defining the hardware layout of the modules at the same time of the software framework they will use to cooperate. An application then could be decomposed in functional requirements and it could be implemented by interconnecting a set of smart modules (i.e., hardware with embedded software running on it) through a proper communication infrastructure sporting an open protocol. These components could be reusable across several applications and different implementations should be available off-the-shelf in order to fulfill different non-functional requirements of the application (e.g., desired final cost, need for hard real-time, high precision, etc.). Finally they all should be glued together by a common bus to be kept fixed for all applications independently from the rest.

With this idea in mind, the task of setting up a prototype of a robot can be faced in a completely different way. If we needed a moving platform, motors had to be selected and integrated with an odometric system (we expect motors and odometry system to be two off-the-shelf smart devices). If accurate positioning is a must for the given application, lasers should be integrated with the odometric system, but, since they speak the same language on the connection bus, it should not be a big deal after all, and the same holds for the integration of other custom devices. Finally, algorithms to support a complex strategy are built upon this and they can seamless integrate since they speak natively the same language of the smart devices.

As any new scenario, this one opens several opportunities, but also new challenges. For instance, which are the pieces of software that could/should be executed on the smart devices? Can we devise a common set, for each possible class of devices, that could/should be implemented as close as possible to hardware? Which is the proper level of abstraction for the information produced and communicated by these devices? Which is the language/protocol these devices should speak and how close this should be to the algorithms that will use information produced by them? What single open infrastructure, if any, should be the one used for communication in this new scenario?

Answering all of these questions cannot be in the scope of a single paper; thus, in this work, we focus just on the latter one, and we provide the requirements of such a communication infrastructure through the analysis of three case studies where low cost (required by the mass market) is the main non functional requirement.

3. TOWARDS MODULAR ROBOTICS

In the following, we consider modular robots as robots made of modules that can provide functionalities to the robotic system, and that can be easily interconnected to each other. This would make it possible to design robots by taking the most appropriate off-the-shelf modules for the specific application, taking into account both the needed functionalities and the market requirements. Plug-and-play interconnection of these modules would make it possible to design a robot with low effort, and to reuse high quality modules available on the market. Decoupling the architecture as a distributed system, in which sensors and may not be on the same board, enables the reuse of a module (e.g., a motor driver) independently from the sensor we have (e.g., an optical encoder actuators on the motor shaft or a mice sensor used for odometry (Bonarini et al., 2004)). This requires modularity both in hardware and in software.

In the field of software engineering the problem of building modular, easily maintainable and reusable code has been deeply investigated and several programming techniques have been proposed. Looking at robotics applications with the software engineering approach, several frameworks have been developed (Mohamed et al., 2008), (Brugali and Scandurra, 2009), (Brugali and Shakhimardanov, 2010). Most of the proposed frameworks focus on software reuse, defining the interfaces that some particular application may need in order to leave the widest possible flexibility to the developers about how to implement the algorithm that will account for the specific task. Some of them, like OROCOS (Bruyninckx, 2001) and ORCA (Makarenko et al., 2006) are based on well known middleware systems, such as Corba (Vinoski, 1997) or ICE (Henning, 2004), which standardize the way different pieces of software can interface each other. They also provide a huge set of off-the-shelf software packages, that enable rapid software development through code reuse and sharing. However, this family of frameworks usually rely on an operating system for the communication stack, and in any case requires high computational power; all this makes them unsuitable to run in embedded systems (e.g., smart sensors based on 8-bit microcontrollers).

On the opposite side, many efforts have been made to design modular robots, in terms of defining a small set of base physical modules that can be composed to build a complex system. Different designs of modular robotic systems have been proposed (Duff et al., 2001), (Salemi et al., 2006), and (Fang et al., 2009). The main goal of these systems is to provide hardware modules that can be disconnected and reconnected in different arrangements to form a new configuration enabling new functionalities. Such modules are complete robotic nodes that can work alone as well as cooperating with others, however, in most of the cases, they integrate a lot of devices (e.g., sensors, actuators, communication system, and on-board logic) in a single piece of hardware. This rises the costs of the whole systems and, while allowing high reconfigurability, the resulting systems are beyond the needs of the market for most applications. Moreover, these projects are usually focused on some hardware, and the software used to enable the cooperation between modules is strictly related to that specific hardware.

In the last years, a new family of sensors, actuators and devices in general have been introduced under the name of "smart devices", meaning that they can do some computation on board instead of simply providing raw data or strictly execute a given command (Huijsing, 2008). Usually, this kind of devices are built for applications different from robotics, such as wireless sensors networks or high-end industrial control, that have requirements much different from those of service robotics, but may share some common aspects, such as the need of distributed processing (e.g., a set of sensors, placed in a building, that send their data after some filtering and classification) or the decentralization of the controller (e.g., smart motors). In the field of wireless sensor networks some software frameworks have then been developed taking into account the possibility to run on low-resource hardware (Masri and Mammeri, 2007), (Levis et al., 2005), but timing and reliability constrains are much more relaxed than in a robotic system.

It should be noticed how these solutions to modular robotics have applied a horizontal approach either from a software perspective (layered architectures or middleware) or a hardware perspective (sets of self–contained physical modules). In our view, this has to be changed and modularization should be implemented vertically, by identifying common functional components and implementing them as "smart devices". From this point of view, the key point becomes their integration and it has to be grounded in an existing physical communication infrastructure similar to what CORBA and ICE are in software middlewares, without their computational burden. In the following, the requirements of such an infrastructure will be discussed assuming that the smart modules implementing functional requirements are actually existing (e.g., a smart camera (Rana et al., 2008), some accelerometers [1], or a complete inertial measurement unit [2]), and they have been already selected in a previous step of the design cycle. An important non functional requirement has been kept fixed in all 3 applications while performing this exercise: we need low cost when aiming at the mass market.

4. CHOOSING A COMMUNICATION BUS FOR LOW COST ROBOTICS - A CASE STUDY

As examples of the proposed approach, let's consider three different kind of robotics products, ranging from robot games up to mobile service robots. We have selected these products, from increasingly demanding applications, to be representative of service robot applications. The goal is to define the common requirements of their communication infrastructure. These robots share some of the requirements, while others are specific of each application. We will analyze the full set of requirements according to functional and non–functional decomposition, by identifying the common needs and by possibly defining a common solution, suitable to all the considered systems, so that it might candidate as a general solution for a wide class of robots.

4.1 Robotic game

A small robot is designed to play a game against a human player. In this case, the functional requirements for the robot are that it has to be able to move at least at 120 cm/sec, and to detect the presence and direction of obstacles, of specific locations identified by colored standpoints, and of the human player, who may wear special, colored clothes. Moreover, it has to interact with the human player through colored LEDs and, eventually, spoken sentences. Non-functional requirements include high reliability in the game context, and a cost compatible with the corresponding mass market, e.g., comparable with that of a video game console.

To satisfy these requirements, we have taken off-the-shelf components for the hardware architecture. The robot bears three motors to drive three omnidirectional wheels, three mice for odometry, a range sensor belt (8 sonars), a camera, a speed controller for the three motors, and a high level controller to manage the game interaction (including LEDs, speech, movement).

The motors are controlled by a hardware module built on a 8-bit microcontroller that receives set points at 100Hz frequency, thus it works in open loop and the position control loop will be closed by the odometry information. In order to close the control loop using data from other modules, some real-time performance has to be reached. In the case of motion control of a game robot, a soft real–time requirement should be sufficient, as an error in the displacement (e.g., caused by a packet loss), can be tolerated. Each motor needs an 8-bit set point, so each packet sent to the controller is 3 bytes long. Estimating the amount of information to be exchanged, and the required speed, is a key issue for our goal of devising a communication channel.

A classical odometry system, with encoders attached to the motors, does not work with omnidirectional wheels, since the free motion of rollers is not related to the shaft speed, and the corresponding odometry computation will be affected by large errors. A possible solution is to use shaft encoders mounted on three free-running wheels (?), but the cost constraints imposed by the mass market, to which this robot is targeted, makes optical encoders not suitable for this application. Odometry is thus implemented by a hardware module, based on an 8-bit microcontroller, that uses three cheap mice sensors pointing to the floor to get the displacement data from the sensors, that computes locally the differential position and pose w.r.t. the previous reading, and that communicates the absolute position and pose to the rest of the system (Bonarini et al., 2005). The module can be asked for a reset in order to remove bias and cumulated errors, e.g., when the robot is in a known location. In order to control the motion closing the motor control loop on this data, let us assume updates at 100Hz with soft real-time constraints also for the odometry. The odometry module provides data about absolute displacement along orthogonal axes (2+2 bytes) as well as the absolute rotation (2 bytes).

The range sensor belt is used to avoid obstacles, and it is managed by another 8-bit microcontroller that collects distances to objects in 8 directions using a set of sonars. It should work at the maximum update speed of the sensors, so the data from the sensor belts are updated at 20Hz. Each sensor gives a 8-bit result, so each packet is 8 bytes long.

The smart camera is used to recognize the human player and the objects to interact with. It has a 32-bit ARM processor on-board for image processing. In this way, the camera will not stream images, but it will directly provide the coordinates of the matching objects in a local coordinate system. As an example, given a set of 8 shapes and 8 colors, the camera will output the list of recognized shapes in its field of view with informations about the color, the size of the box surrounding the shape and its position in the image. The the camera publishes data at 10Hz. With a 240x160 resolution, and limiting the maximum number of recognized shapes in one frame to 8, each shape is described by its position in the frame (1+1 bytes), the dimension of the surrounding box (1+1 bytes), the shape identifier and its color (3+3 bits), resulting in 5

[1] http://st.com/stonline/products/families/evaluation_boards /steval-mki004v1.htm
[2] http://st.com/stonline/products/families/evaluation_boards /product_evaluation_boards/motion_sensors/inemo.htm

Table 1. Robotic game communication bus functional requirements

	Update frequency [Hz]	Bandwidth [kbps]	Real time?
Motor controller	100	2.4	soft
Odometer	100	4.8	soft
Range sensor belt	20	1.28	soft
Camera	10	3.2	no

bytes for each shape. Each packet from this camera, can thus be up to 40 bytes long.

All the modules interact with a central unit, based on a 32-bit ARM processor, that knows the state of the game and, on the basis of the readings from the odometry module and the description of the objects seen by the embedded camera, commits commands to the motors in order to follow the trajectory that lets the game evolve.

Given the set of hardware modules that compose the robot, we can analyze their requirements with functional and non-functional decomposition for what concerns the communication channel. The functional requirements in terms of communication bus, for this robot, are summarized in table 1. The non-functional requirements for this application are that the overall system cost must be very low and thus the modules should share the same communication bus.

4.2 Balancing robot

This is a service robot designed to work in domestic environments. The functional requirements include the abilities for Self Localization And Mapping (SLAM), interaction with a user, possibility to bring up to 15 Kg of payload. Among the non-functional requirements, we mention the need to maneuver with agility in the home environment. In order to easily operate in this context, we have designed a two-wheeled balancing platform that is capable to make turns on place, and that has a lower footprint if compared to four–wheels solutions.

The robot has two low-cost high-power DC motors that independently actuate the two wheels. Each motor is controlled by a hardware module governed by a 8-bit microcontroller that follows a torque setpoint reading the current flowing in the motor through a sense resistor. The controller receives a 2 byte torque setpoint at 100Hz.

Optical encoders mounted near the wheels are read by a hardware module that computes, locally, the rotation of the wheels and communicates the robot position (2 bytes) and orientation (2 bytes). The module can be asked for a reset in order to remove bias and cumulated errors, e.g., when the robot is in a known location, or when it is going to overflow, as the central controller knows the new starting position in the world. The module thus outputs 4 bytes packets at 100Hz.

The angle of the frame with respect to the vertical position is measured by an IMU (Inertial Measurement Unit) based on a 32-bit ARM processor that reads the angular velocity from a MEMS gyroscope and the accelerations from a MEMS accelerometer to apply a Kalman filter in order to compute the pose and the angular rate of the robot. The IMU outputs the angle with the vertical and the angular rate (2+2 bytes) at 100Hz.

Table 2. Balancing robot communication bus functional requirements

	Update frequency [Hz]	Bandwidth [kbps]	Real time?
Motor controllers (2)	100	1.6	soft
Odometry	100	3.2	soft
IMU	100	3.2	soft
Range sensor belt	20	1.28	soft
Smart camera	10	10	no
Image stream	15	1000	no
Wireless connection	10	10	no

A range sensor belt, based on a 8-bit microcontroller and a set of sonars, is attached to the frame. It outputs 8 byte packets at 20Hz.

On the top of the balancing platform there is a smart camera used to interact with people (e.g., using face recognition algorithms), to localize the robot in the environment (embedded SLAM) and to avoid obstacles. As the robot can be used for telepresence applications too, the camera needs, upon request, also to stream mpeg4 video, at bitrate of about 1000kbps.

The robot is also equipped with a Zigbee based wireless module used for remote control and to interact with other appliances. This module needs to exchange asynchronous messages with an off-board computer in a publish/subscribe paradigm, we can assume 10 messages per second at a maximum rate of about 10kbps.

A central controller, based on a 32-bit ARM processor, sends commands to the motors for both the stability control, based on IMU data, and the motion control, receiving the position from the odometry module. The controller is also responsible of receiving and sending commands through the wireless module and interprets the output of the smart camera on the basis of some given behaviors.

The functional requirements in terms of communication bus, for this robot, are summarized in table 2. The non–functional requirements for this application are that the overall system cost must be low, the modules should share the communication bus, the communication framework must work on 8-bit microcontrollers, on 32-bit ARM processors and on a standard x86 PC and, at last, the camera will have an high-bandwidth channel to stream images too.

4.3 Manipulator

The last product considered here is a 6-dof robotic arm, designed to be placed on the top of the balancing platform to interact with objects in a domestic environment. As it needs to work next to humans, the arm needs a force feedback in order to prevent any injury to surrounding people. The end effector needs to rely on force feedback too, as it might need to grasp fragile objects.

The arm is built with a modular approach, as each joint integrates the motor, an optical encoder, the force sensor (current measurement on arm joints, piezo sensors on the end effector) and the controller. Each module is based on

Table 3. Manipulator communication bus functional requirements

	Update frequency [Hz]	Bandwidth [kbps]	Real time?
Joint setpoint (6)	100	1.6	soft
Joint position (6)	100	1.6	soft
Joint force feedback (6)	1000	16	hard

a 32-bit ARM processor, that shares information about position and force with the other modules, while it receives position and force set points from an external control unit. The set points are updated at a maximum rate of 100Hz (16bit integer for each joint) as the information of joint position (16bit integer too), while the force control loop runs at 1Khz (again, 16bit integer). For the position control soft real-time may be adequate, as we are not interested in very precise motion control, while the force feedback must rely on a hard real-time communication channel, to prevent damages to the environment or to avoid to damage the grasped object. Thus each module outputs 2 bytes at 1khz, 2 bytes at 100Hz, and receives 2 bytes at 100Hz.

The functional requirements in terms of communication bus, for this robot, are summarized in table 3. The non-functional requirements for this application are that the overall system cost must be low, the modules should share the communication bus and that the robotic arm should be easy to integrate in existing systems in a plug-and-play way, relying on the same communication bus.

5. DISCUSSION

The applications that we have analyzed share most of the requirements while others are strictly application specific.

To meet the common requirements (bandwidth of less than 10kbps per module, update rates under 100Hz), with the non-functional requirement of being low cost and implementable on a heterogeneous set of devices, including low power 8-bit microcontrollers, we have considered a set of field buses that have been widely reviewed (Piontek et al., 2007) (Elmenreich et al., 2003) (Kopetz, 2000), including event-driven and time triggered protocols. As expected, the most adopted bus is the CAN bus[3], which has been developed for automotive applications gaining popularity up to be implemented in a wide set of microcontrollers, ranging from low-cost 8-bit architectures up to modern 32-bit ARM devices. The CAN bus is suited to work in noisy environment, that is a plus, and many protocols have been developed upon CAN, e.g., CANOpen (?) and DeviceNet (?). CANOpen is royalty–free, that satisfy the non–functional requirement of low cost, and can be used with many frameworks, also distributed as open source like Canfestival[4], that makes easy to integrate heterogeneous modules.

If a smart camera module has to be used also to stream images, like in the tele–presence application, the requirements are completely different, as there is the need of high bandwidth and low update rates, that is not what the CAN bus was designed for. In this situation, our opinion is that is better to use two separate channels: the output of the embedded algorithms should be sent on the same bus of the other modules, to enable an easy integration, while the image stream can use a dedicated channel (e.g., firewire or wi–fi).

[3] http://www.semiconductors.bosch.de/en/20/can/index.asp
[4] http://www.canfestival.org/

The last application, that involves a distributed control loop with sensor and actuators exchanging information at 1Khz, introduces the need of hard real-time. In this way, a non-deterministic protocol like CANOpen is not suitable, and a TDMA approach needs to be investigated, as it guarantees that the packets you are interested in will arrive in a defined time slot. To better integrate with other modules running on CAN bus, we considered TTCAN (Leen and Heffernan, 2002), that is a time triggered implementation of CAN bus making it deterministic. Unfortunately, TTCAN needs a hardware component to time stamp the packets, that is not present on all CAN enabled devices. Another alternative has been presented within the COSMIC middleware project (Kaiser et al., 2003), which provides a hard real-time channel with guaranteed timing, a soft real-time channel with weaker time constraints and a non real-time channel. Such an architecture seems to be very flexible, but in order to manage the dynamic priorities needed by soft real-time messages COSMIC may need software components that could not fit on low power devices (Piontek et al., 2007). For the devices on which COSMIC cannot run, we are investigating the TTP/A protocol (Kopetz et al., 2000), that is fully implemented in software requiring only an UART port and has been ported also to low-cost 8-bit microcontrollers. The drawback is the need for a communication channel different from the one used for other modules, adding complexity to the system design.

In this work, we have presented a methodological approach to design low–cost modular robotics, and how it could be applied to the definition of the requirements of the communication infrastructure to develop such systems and to enable rapid prototyping. We have analyzed the common requirements of a set of modules that can apply to different robotic applications and we have identified the existing low–cost technologies and protocols that can be exploited to build smart modules upon. We have shown how it is possible to face rapid prototyping starting from functional and non–functional requirements, and using components off the shelf (COTS).

REFERENCES

Andrea Bonarini, Matteo Matteucci, and Marcello Restelli. A kinematic-independent dead-reckoning sensor for indoor mobile robotics. In *Proceedings. 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2004. (IROS 2004)*, volume 4, pages 3750–3755, 2004. doi: 10.1109/IROS.2004.1389998.

Andrea Bonarini, Matteo Matteucci, and Marcello Restelli. Automatic error detection and reduction for an odometric sensor based on two optical mice. In *Proceedings of International Conference on Robotics and Automation (ICRA'05)*, volume 4, pages 1687–1692, Los ALamitos, CA, 2005. IEEE Computer Press.

Andrea Bonarini, Matteo Matteucci, and Marcello Restelli. Mrt: Robotics off-the-shelf with the modular robotic toolkit. In Davide Brugali, editor, *Software Engineering for Experimental Robotics*, volume Springer Tracts in Advanced Robotics 30/2007, pages 345–364. Springer Berlin / Heidelberg, April 2007.

Davide Brugali and Patrizia Scandurra. Component-based robotic engineering (part i) [tutorial. *IEEE Robotics & Automation Magazine*, 16(4):84–96, dec 2009.

Davide Brugali and Azamat Shakhimardanov. Component-based robotic engineering (part ii). *IEEE Robotics & Automation Magazine*, 17(1):100–112, mar 2010.

Herman Bruyninckx. Open robot control software: the orocos project. *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, pages 2523–2528, 2001.

David G. Duff, Mark Yim, and Kimon Roufas. Evolution of polybot: A modular reconfigurable robot. In *Proceedings of International Symposium on the Harmonic Drive*, 2001.

Wilfried Elmenreich, Stefan Pitzek, Technische Universitt, Wien Technische, and Universitt Wien. Smart transducers - principles, communications, and configuration. In *In Proceedings of the 7th IEEE International Conference on Intelligent Engineering Systems*, pages 510–515, 2003.

Zheng Fang, Yanhua Fu, and Tianyou Chai. A low-cost modular robot for research and education of control systems, mechatronics and robotics. *2009 4th IEEE Conference on Industrial Electronics and Applications*, pages 2828–2833, may 2009.

Bill Gates. A robot in every home. *Scientific American,*, (1):58–65, 2007.

Michi Henning. A new approach to object-oriented middleware. *Internet Computing, IEEE*, 8(1):66 – 75, jan 2004.

Johan H Huijsing. *Smart Sensor Systems : Why ? Where ? How ?*, chapter 1. John Wiley & Sons, Ltd, 2008.

Jared Jackson. Microsoft robotics studio: A technical introduction. *Robotics Automation Magazine, IEEE*, 14 (4):82 –87, dec 2007.

Patric Jensfelt, Gunnar Gullstrand, and Erik Förell. A mobile robot system for automatic floor marking. *Journal of Field Robotics*, 23, June/July 2006.

Jorg Kaiser, Cristiano Brudna, and Carlos Mitidieri. A real-time event channel model for the can-bus. *Proceedings International Parallel and Distributed Processing Symposium*, (C), 2003.

Herman Kopetz. A comparison of can and ttp. *Annual Reviews in Control*, 2000.

Herman Kopetz, Michael Holzmann, and Wilfried Elmenreich. A universal smart transducer interface: Ttp/a. In *Proceedings of the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2000. (ISORC 2000)*, pages 16–23, 2000.

Gabriel Leen and Donald Heffernan. Ttcan: a new time-triggered controller area network. *Microprocessors and Microsystems*, 26(2), 2002.

Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, and David Culler. Tinyos: An operating system for sensor networks. In Werner Weber, Jan M. Rabaey, and Emile Aarts, editors, *Ambient Intelligence*, pages 115–148. Springer Berlin Heidelberg, 2005.

Alexei Makarenko, Alex Brooks, and Tobias Kaupp. Orca: Components for robotics. *Conference on Intelligent Robots*, 2006.

Wassim Masri and Zoubir Mammeri. Middleware for wireless sensor networks: A comparative analysis. *2007 IFIP International Conference on Network and Parallel Computing Workshops (NPC 2007)*, pages 349–356, sep 2007.

Nader Mohamed, Jameela Al-Jaroodi, and Imad Jawhar. Middleware for robotics: A survey. *2008 IEEE Conference on Robotics, Automation and Mechatronics*, pages 736–742, sep 2008.

H. Piontek, W. Elmenreichand, and J Kaiser. Interface design for real-time smart transducer networks examining COSMIC , LIN , and TTP/A as case study. In *Proceedings of the 15th International Conference on Real-Time and Network Systems*, pages 195–204, 2007.

Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

Vincenzo Rana, Matteo Matteucci, Daniele Caltabiano, Roberto Sannino, and Andrea Bonarini. Low cost smart-cam design. In *Proceedings of 6th IEEE Workshop on Embedded Systems for Real-tme Multimedia (ESTIMedia 2008)*, pages 27–32, Los Alamitos, CA, USA – USA, October 2008. IEEE Computer Press.

Behnam Salemi, Mark Moll, and Wei-min Shen. Superbot: A deployable, multi-functional, and modular self-reconfigurable robotic system. *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3636–3641, oct 2006.

Steve Vinoski. Corba: integrating diverse applications within distributed heterogeneous environments. *Communications Magazine, IEEE*, 35(2):46 –55, feb 1997.