EME – Low-Cost Embedded Multiprocessing Environment

Jean-Marc Ranger and François Michaud
LABORIUS - Research Laboratory on Mobile Robotics and Intelligent Systems
Department of Electrical and Computer Engineering
Université de Sherbrooke, Sherbrooke (Québec Canada) J1K 2R1
{ranger,michaudf}@gel.usherb.ca, http://www.gel.usherb.ca/laborius

Abstract

This paper presents EME, a low-cost solution for a highly modular multiprocessing hardware-software environment useful in designing autonomous mobile robots. High processing capabilities, the ability to interface a large number of sensors and actuators, extensibility, redundancy and flexibility are issues addressed by this environment. An autonomous robot equipped with two cameras, one for line-following and the other for visual communication by light signals, is used to demonstrate EME capabilities and performances.

1 Introduction

Designing autonomous robots requires the integration of many sensors and actuators on a physical base to give the robots the capacity to interact with their environment and fulfill their tasks. Even if sensors and actuators do impose limits on robot performance and autonomy, another critical factor is the on-board processing capability required for controlling the robot. On-board processing imposes important influences on the electrical and structural requirements of the robot, and also on its overall cost. For instance, using a PC104-based computer on a robot is a popular solution these days to increase the processing power of the system, compare to the use of more conventional microcontrollers. But it also means bigger controller boards, a hard disk, bigger batteries and motors, which lead to a much bigger and pricier robot overall.

However, it is technologically possible to develop low-cost multiprocessor solutions that give more processing power to such systems. Various robots already use multiple microcontrollers in various subsystems. Each subsystem is usually designed individually, and special functions are usually added so that the subsystems can exchange informations with each other. For instance, our Pioneer I robot uses a Motorola 68HC11 controller for basic sensors and actuators control, a Motorola 68332 controller for the Fast Track Vision System and another 68332 board for a LISP-based real-time operating system [10], each communicating with each other via a packet communication bus, limited to 10 packets per second.

We believe that a better solution would be to have a multiprocessing development environment that facilitate extensibility, modular design, reconfigurability and better performance, especially for robots that are limited by structural constraints and that require high on-board processing capabilities. This paper presents a multi-microcontroller environment named EME (for Embedded Multiprocessing Environment), designed to match these specifications. Section 2 describes EME and its characteristics. Section 3 demonstrates the capabilities of EME on a small mobile robot equipped with two cameras and three embedded controllers. Experimental results are described in Section 4. Section 5 presents related development environments, followed by the conclusion and future research.

2 EME – Embedded Multiprocessor Environment

There are two categories of multiprocessing systems. For the loosely coupled type, processors are basically independent. They share information over a standardized bus or network and each software always runs on the same processor. In tightly coupled multiprocessor systems, processors share the same memory space and software never knows on which processor it runs: all processors need to be identical. For EME, we adopted the loosely coupled approach because independent processors allow easier fault detection and recovery. That approach also has more flexibility as different types of microprocessors can be chosen ac-

cording to the capabilities required for each part of the robot.

2.1 Hardware specifications

For the hardware platform, we chose to develop EME for the Motorola 683xx family, more specifically on a 16 MHz 68331 microcontroller board with its own power supply, 256 Kb of RAM, 256 Kb of ROM and a RS-232 communication interface (via a *Serial Communication Interface* port) for programming. However, very few 68331 specific hardware components are used in EME to facilitate portability of this environment on other microcontrollers.

For inter-processor communication, EME uses the 68331's *Queued Serial Peripheral Interface* (QSPI) port. This is a multi-point serial bus, providing easy peripheral expansion or inter-processor communication through synchronous, half-duplex, two-wire transfers. This bus is compatible with SPI systems found on microcontrollers from Motorola or from other manufacturers. Figure 1 illustrates the QSPI interface between three 68331 microcontrollers. Experiments demonstrated that a maximum theoretical transfer rate of 4 Mbaud could be reached using a QSPI bus between two microcontrollers, but as soon as three or more devices were connected, the maximum no-failure transfer rate was around 115 kbaud [7]. But higher is the transfer rate, greater is the load on the microcontroller. For EME, tests showed that a 64 kbaud rate leads to a good and conservative compromise.

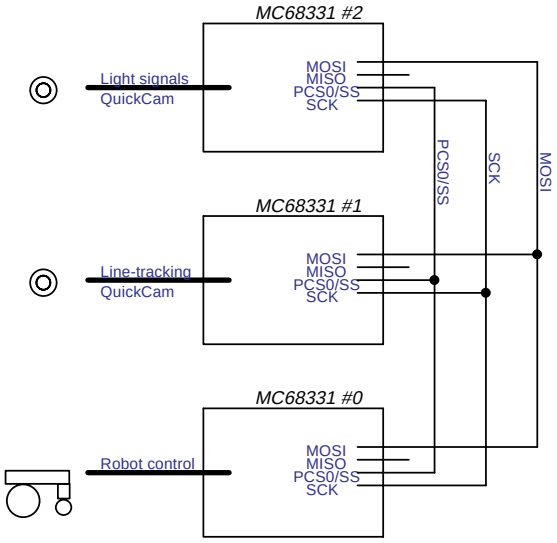

Figure 1: QSPI bus used to interconnect three 68331 microcontrollers. SCK is the serial clock signal, PCS0 is used to initiate serial transfer and MOSI is the serial data line.

2.2 Software Specifications

Each board connected by their QSPI ports has a unique ID number (from 0 to 30) set by hardware. This allows boards to identify the recipient of a message. Communication between microcontrollers is done according to a token bus protocol [6]: a token moves from one board to another, the board with the token becomes the master and manages communication on the QSPI port. No collision between messages can then occur. This greatly simplifies the errors handler and ensures that messages are received in the same order as they are sent. By monitoring the ID number of the sender indicated in the message header, all boards know which others are present. Also, a simple watchdog mechanism is implemented to detect failure of the next board to receive the token. A board that detects such a failure broadcasts a message stating which board has failed. All messages are 32 bytes long, consisting of a heading and the message itself which cannot exceed 64 kbytes. Message chopping and reassembling are transparently managed by EME.

Since EME uses a token bus protocol, there is no dedicated master or slaves. All boards are equals, with one exception at startup when board #0 is responsible for initializing EME by generating the first token. This allows master redundancy, making the system more reliable, and the control software can be distributed as required by the application. For real-time processing, each board runs a preemptive kernel with round-robin scheduling. EME kernel offers conventional services like task creation and destruction, timing mechanisms, semaphore, preemption suspension, interrupt inhibit and card identification services. The particularity of EME's kernel is the multi-microcontroller mailbox message passing service designed to exchange information between tasks. The mailbox service is the same for local message passing or for inter-board communication: EME kernel handles moving a message from one board to another when the destination is located on another board. The main advantage of this is easy reconfiguration of the application. Code can be moved from one board to another with little to no modifications. This allow moving code quickly and easily from one board to another when required, with little to no modifications.

3 Experimental Setup

To test the processing capabilities of EME, we use a small robotic platform, 10 inches in diameter, capable of moving and avoiding obstacles with infrared and collision sensors. The purpose of using EME with this platform is to give vision capabilities to the robot. As a test application, the robot must follow a black line painted on a white floor, and interact with a human operator using a light signaling device. A practical application could be a small robot for material transport that follows a path drawn on the floor (for easy modification when necessary). In our experiments, the test application is not as important as testing EME's capabilities in quickly setting up a complex real-time intelligent system, using rapid and transparent multiprocessing design at very low cost and with readily available hardware.

To accomplish this task, the robot needs two cameras. The reasons are that two independent sources of information need to be tracked without interruption in two different directions. Also, the optimal camera settings for these sources differ. Three 68331 microcontroller boards connected using the QSPI ports and two black & white Connectix QuickCam digital cameras are used. Images received from the cameras are 80x60 pixels with 4 bits encoding 16 possible brightness values (grayscale depth). Figure 2 shows a picture of the robot used, named *BigBrother* because of its height. Note that the overall height of *BigBrother* could have been reduced by designing custom boards instead of using the boards that we had at our disposal. Its overall cost is around 550$US.

The processing required to accomplish this task is distributed among the three microcontroller boards. The two boards connected to a camera are responsible for extracting the information required by board #0 to control the actions of the robot. The top board (#2) uses the top-most camera and interprets visual messages received from a light signaling device (as explained in the following paragraph). The middle board (#1) manages the line-tracking camera and its algorithm. The third board (#0) at the bottom is used to interface the other sensors of *BigBrother* (four bumps switches, two infrared emitters and one detector) and for navigation control of the robot. We initially planned to connect the robot's buzzer (controlled by a single digital output) to board #0 but no port was easily accessible on that board. However, EME allowed us to connect the buzzer to board #1 where digital output ports were available. The code for controlling the buzzer was then implemented on board #1. When boards #0 or #2 require the use of the buzzer, the mailbox message is simply readdressed toward board #1. This demonstrates how EME facilitates reconfigurability when it is required.

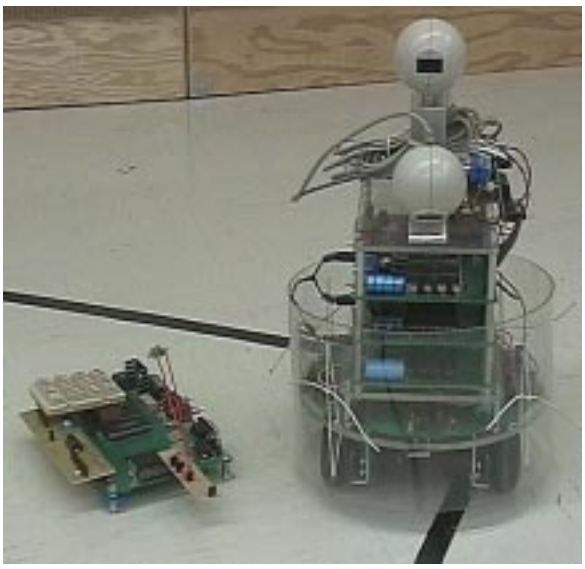

Figure 2: *BigBrother* (right) with the visual light signaling device (left).

To communicate information to the robot, we decided to use visual signals. While this method is much more limited in bandwidth than radio communication for instance, it allows the robot to know the position of the sender related to itself, directly from the communicated signals [9]. Asynchronous transmission of 11 bit messages is implemented. Each message is made of one start bit, eight data bits (an ASCII code) and two stop bits, with Manchester encoding[1] [6] is used. Manchester encoding allows easy discrimination between the light signal and other light sources. The receiver is assured that there is always a transition in the middle of every bit, allowing easy synchronization. Also, as the light flashes continuously, a short "no light" period (caused for instance by something passing between the transmitter and the receiver) will be detected instead of simply causing message corruption. In our experiments, a human operator can communicate requests to the robot using a light-signaling transmitter shown in Figure 2. This transmitter uses a telephone-style keyboard as the input device, a standard flashlight bulb and is also implemented on a 68331

[1] With this scheme a binary 1 is encoded as a low-high signal and a binary 0 as a high-low signal. There is always a transition at the center of each bit encoded. For example, the bit stream 0100 becomes 01100101.

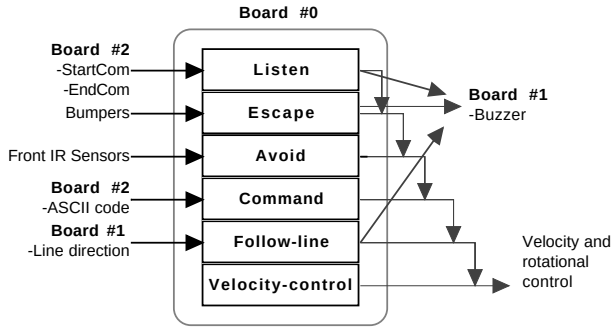

Figure 3: Behavior organization used with *BigBrother*.

board to eventually use such a device on robots for group experiments. Using digits 1 to 9, the user can communicate commands based on the position of the number on the keyboard (e.g., 4 for left, 6 for right, 2 for forward, etc.).

For signal interpretation, board #2 first extracts black regions from white ones by comparing the image brightness values with a threshold (4 in our case). Then, when more than 15 pixels are illuminated on the image, the algorithm recognizes a visual signal. The sequence of presence or absence of visual signals over time is compiled and interpreted to derive one of the 9 possible commands that can be communicated. Based on this interpretation, informations relayed to board #0 are: *StartCom* (transmitter has just been detected), *EndCom* (transmitter lost), *ProtocolError* (invalid message received) and *Data* (valid message received, followed by the ASCII code interpreted).

For line-tracking, extraction of black regions from white ones in the image is done using the same technique as for the other camera. A weight is given to each pixel (1 for black and 0 for white) and the line position is evaluated to be at the center of mass of the image. Then, based on the horizontal position of the center of mass of the black region of the image, a message is sent to board #0 to indicate the direction to take to follow the line. Possible messages are *Dir_None* (no line is detected), *Dir_FarLeft* or *Dir_FarRight*, *Dir_CloseLeft* or *Dir_CloseRight*, and *Dir_Center*.

Finally, on board #0, a behavior-based approach [1] is used to control the robot. Six behaviors are used to control the velocity and the direction of *BigBrother* using Subsumption [3] as the arbitration mechanism. These behaviors, shown in Figure 3, are: *Listen*, using states derived by board #2, this behavior makes the robot stop when a valid signal is detected, in order to get the remaining part of the message; *Escape*, making the robot move away from a collision with an obstacle; *Avoid*, making the robot move away from an obstacle detected using its front infrared sensors; *Command*, orienting the robot according to a request received from the human operator by visual communication (the actual command is given from the directives received from board #2); *Follow*, orienting the robot in front of the black line according to the directives received from board #1; and *Velocity-control*, making the robot move forward at a desired speed. Some behaviors also use the buzzer to generate sounds for signaling particular states of the robot (like 'I have lost the line', 'I cannot synchronize with the light signal', etc.).

4 Results

Overall, the behaviors are all running as expected and the robot is fulfilling its task. Runtimes are close to what can be provided by the batteries (around half an hour). Functionally, there is no sign that the robot is using a multiprocessor setup: it operates the exact same way we would expect from a high performance single processor system

For vision processing, boards #1 and #2 each reads three to four images per second, which is little but enough for the task. This limit is caused by the interface circuitry (based on a MC6821 IC) between the cameras and the 68331 boards. The line-tracking algorithm is currently a performance-limiting factor, as it shows some difficulties in differentiating the line from a dark spot on the floor or from a black wall. The problem is even worse due to the fact that the camera is really high above the ground (around 11 inches), giving it a large field of vision. Coupling a better tracking algorithm [4] with a cropped image would improve the robot's performance.

For the communication using visual signals, the first limitation is the low camera throughput. Our algorithm is implemented so that the presence or absence of a visual data bit is detected using at least two consecutive images. Reading three to four images per second, the transmission period should be greater than 0.66 sec. We chose a period of 0.7 sec, and using Manchester encoding the bit transmission time is then of 1.4 sec. Since each ASCII code transmitted is encoded on 11 bits, it takes just over 15 seconds to transmit. As increasing camera throughput is not an option in this case, but since only 9 of the 256 ASCII codes are used, reducing coding from 8 to 4 data bits would be an easy way to decrease

transmission time to just under 10 seconds. Another difficulty is that the transmitter currently uses a very small light bulb, limiting its range to a little over three feet. However, message reception is very good, with around 90% accuracy. Some rare errors occur during start bit synchronization or close to system boot-up, when the auto-frequency adjustment is not completed. Invalid communication conditions, caused for example by the presence of another light source or by turning off of the transmitter, are always correctly detected in around two seconds. The method is simple: when a light signal (on or off) stays the same for an abnormally long time (currently set a 3 times the standard length, i.e., 2.1 seconds), communication is considered incorrect. This value is directly related to transmission speed: improving camera throughput would improve this value by the same factor.

Without EME, it would have been almost impossible to implement such abilities on a robotic platform using only a single 68331 microcontroller. Image processing would have been done at an unacceptably low rate of one or two frames per second (around half of ours, since the two cameras would be connected to the same board), which is not sufficient for real-time vision processing for a robot in movement.

EME also greatly facilitates the expandability of the robot. For instance, adding another high throughput sensor would require the following steps: 1) Stack the board on top of *BigBrother*; 2) Extend the QSPI wires to the fourth board; 3) Set the ID number of the board (using hardwired connections); 4) Transfer EME and the sensor program code to on-board ROM; 5) Start the robot (the initialization process would then detect the use of four boards instead of three); 6) Add control code, on board #0 or elsewhere if more appropriate.

These steps were followed when board #2 was added to *BigBrother*, and everything worked on the first trial. Adding another board would not affect the image processing rate of two cameras. Also, while it may be possible to design new single-processor boards to increase the number of I/O ports or the amount of memory, simply connecting another board to the QSPI bus is a fast and easy method. Experimental work done with similar platforms has shown that 16 microcontrollers can be connected using the QSPI bus [7]. However, expansion is not unlimited. Even though EME's software does not limit the number of microcontrollers, as the number of boards increases, the bus bandwidth remains the same, giving less effective bandwidth to each board. Message latency will also increase as the token passes less and less often on each board. The exact message traveling time is difficult to evaluate since it depends on many factors such as the number of boards, the token circulation order, processing load of the microcontroller and presence or absence of other waiting messages. A test with EME using three boards just relaying messages indicated that the average send-to-receive time between two boards is 14 msec. This test takes into account all EME delays such as the time to exchange the token and the processing time used by the kernel. The same test with all software running for the *BigBrother* application showed an average time of 36 msec. Compared to the 100ms for the Pioneer I, we believe this is a notable improvement.

Finally, comparing *BigBrother* with a Pioneer 2 robot equipped with PC104 Pentium 233 embedded computer, its computing power is ten times less (comparison based on how long it takes to loop from 1 to 1.000.000), but *BigBrother* uses seven times less current (under 0.7A peak compared to close to 5A). Similar ratios are obtained for size and for weight. Depending on the application, EME can therefore be an very interesting option.

5 Related Work

For the design of small autonomous robots for the RoboCup competition, VUB AI Lab uses a board named RoboCube [2], a single Motorola 68332 microcontroller board with lots of hardware interfaces. The first version has 24 A/D converters, 6 D/A converters, 3 RS-232 interfaces and much more, all built-in. This solution is good if you have well-known needs, being high on the number of sensors and actuators that interface through the kind of port that is available, but being relatively low on the required processing power. In smaller applications, the board is not used to its full capacity, and no solution is provided for applications that require more hardware interfaces or more processing power. Using EME on RoboCube boards could result in a very good combination of hardware and software capabilities.

For multiprocessing systems, the Chimera [11] real-time operating environment is an example of a VMEbus-based tightly coupled processor system. Totally unlimited on the processing power side, Chimera has not been ported to microcontrollers. Its use in robotics is thus limited to applications where dedicated hardware interface boards can be afforded (in size and in price).

The environment most similar to ours is the newly released Kameleon 376 SBC board [8], made by K-Team. For multiprocessing, the Kameleon board provides two interfaces: K-Net and Multi-Microcontroller

Bus (MMA). K-Net is for communication between the main (master) processor and extension "turret" (slave) processors, using the QSPI bus. Like EME, each board has its own unique identification number and initialization of the number of processors involved is done at startup. However, communications are always initiated by the master and slaves answer only to requests from the master. EME allows more flexibility and better fault recovery by having all boards communicate with each other without having to go through the dedicated bus master. The Multi-Microcontroller Bus (MMA) provides a fast interface designed to support asynchronous multi-processor applications. But this multi-processor scheme also operates in a star configuration, with one dedicated main (or master) microcontroller located in the center. Only four slave microcontrollers can be controlled using this configuration.

Another example of a multi-microcontroller robot is the hexapod Hannibal [5]. This robot uses ten 68000-derivated controllers communicating via an I2C multipoint serial link. But only one microcontroller is doing real processing using a behaviour-based control software, the others are only acting as sensor interface circuitry. By contrast, EME is a more distributed and flexible approach. Not only the processing and the sensor interface system are distributed, the true real time kernel makes it possible to implement any kind of distributed control architecture. For example, EME allows the conception of a single robot that would have two subsumption-based controllers on two different boards, each one located on the board that is physically connected to the controlled actuator.

6 Summary and Conclusion

This paper argues that an efficient multiprocessing environment is not only useful but also necessary to design low-cost autonomous mobile robots that require a lot of sensing, actuating and complex processing abilities. This environment is an integral part of the "intelligence" manifested by the robot. EME, the embedded multiprocessing environment described in this paper, provides simple and uniform hardware-software development tools for quick and optimal design. The experimental setup shows how EME allows to control a small robot equipped with two cameras, using three embedded 68331 microcontroller boards. In future research, we plan to use EME in the design of an autonomous hexapod robot equipped with a color camera, voice recognition and on-board distributed decision abilities, using up to 9 microcontroller boards to control the robot.

Acknowledgments

This research is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Canadian Foundation for Innovation (CFI). Special thanks to S. Caron, P. Mabilleau and L. Drolet for their contributions to this project.

References

[1] R. C. Arkin. *Behavior-Based Robotics*. The MIT Press, 1998.

[2] A. Birk, H. Kenn, and T. Walle. Robocube, a universal special-purpose hardware for the robocup small robots league. In *Proc. Int. Symp. on Distributed Autonomous Robotic Systems*, 1998.

[3] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1986.

[4] N. Chen. A vision-guided autonomous vehicle: an alternative micromouse competition. *IEEE Transactions on Education*, 40(4), 1997.

[5] C. Ferrell. Robust agent control of an autonomous robot with many sensors and actuators. Master's thesis, MIT, Department of EECS, Cambridge, Massachusetts, 1993.

[6] F. Halsall. *Data Communications, Computer Networks and Open Systems*. Addison-Wesley, 1992.

[7] A. Hettman. Using the ssc (spi) interface in a multimaster system. Siemens ApNote #1632, June 1996.

[8] K-Team. *Kameleon 376 SBC User Manual*. Lausanne, 1.0 edition, 1999.

[9] F. Michaud and M. T. Vu. Managing robot autonomy and interactivity using motives and visual communication. In *Proc. Conf. Autonomous Agents*, May 1999.

[10] IS Robotics. *The Pioneer L Robot Software Guide*. Somerville, MA, Revision 2.0, 1997.

[11] D.B. Stewart, D.E. Schmitz, and P.K. Khosla. The Chimera II real-time operating system for advanced sensor-based robotic applications. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(6):1282–1295, 1992.