A mobile mini-robot architecture for research, education and popularization of science

Sol Pedre, Pablo de Cristóforis, Javier Caccavelli, and Andrés Stoliar

Departamento de Computación,
Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires
Buenos Aires, Argentina
{spedre,pdecri,jcaccav,astoliar}@dc.uba.ar

Abstract. Mobile mini-robots are commonly used for research, education and popularization of science. Often, commercially available mini-robots don't quite fit the characteristics needed for a particular task, and are difficult to adapt since they have proprietary software and hardware. Moreover, they are often quite expensive. In this work we present a relatively low-cost, reconfigurable robot equipped with a wide variety of sensors and enough processing power to allow the on-board execution of intelligent algorithms. We present the complete hardware architecture, and a modularized software architecture that makes full use of hardware interruptions and software processes to have a perfectly timed control of the robot. All these characteristics make the new mobile mini robot ExaBot a very malleable, multi task mini-robot.

Key words: mobile mini robot, reconfigurable hardware architecture, interrupt based software architecture, research and education robot platform

1 Introduction

Mobile mini-robots are commonly used for research, education and popularization of science. There are many commercial mini-robots available. However, often these mini-robots don't quite fit the characteristics we need for certain tasks and are very difficult to adapt because they have proprietary software and hardware. For example, Khepera [1] mini-robots are well known, and widely used for research and education, but are limited when modifications to their sensing or programming capabilities are needed. Another example of commercial mini robots are LEGO Mindstorm [2] - a good tool for popularization of robotics and science in high schools and universities. However, these robots are not suitable for research.

Another important drawback of commercial mini-robots is their cost, often way over the budget of many universities or study centers. This makes buying or

upgrading these robots difficult, and limits the possibilities for multi-robot system's research. Finally, the knowledge gained from the design and construction of robots from scratch is a major incentive for this task.

Taking as a premise that "There is no robotics without robots", and knowing that often commercially available robots do not have the characteristics needed for some tasks or are too expensive to acquire, developing new robot prototypes for research, education and popularization of science becomes a relevant task.

The motivation of this work is to design and build a robot prototype of reduced cost with a reconfigurable sensing system and enough processing power to allow the on-board execution of intelligence algorithms and to process the sensor's information.

In this paper we present the ExaBot: a mini-robot developed in the Laboratorio de Robótica y Procesamiento Embebido, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires. The rest of this paper is organized as follows: in section 2 we describe the general hardware architecture of the ExaBot and its possible configurations, in section 3 we describe the different sensors available and the needed hardware architecture to control them, in section 4 we describe the motors in the robot and the hardware related to them. In section 5 we describe the modularized, interrupt and process based software architecture. In section 6 we discuss the results of this work and show examples of the use of the ExaBot in research, education and popularization of science. Finally, we draw some conclusions and present ongoing work in section 7.

2 Hardware architecture

In this section we describe the general architecture of the robot: the chosen mechanical body, the different processing units and communication buses. We also present two possible architecture configurations of the ExaBot.

2.1 Mechanical kit

As our goal in this work was to design and implement the electronics and software needed for the construction of the robot, we chose to use a pre-built mechanical kit to leave most mechanical problems aside. We reviewed several models, and decided to use the Traxster Kit [3]. This kit is relatively small (229 mm length $\times$ 203 mm wide $\times$ 76 mm height) yet it can accommodate all the needed electronics and processing units. It is quite light because it is aluminum made. It comes with two direct current motors (7,2 V and 2 Amp/hour) that have built-in quadrature encoders. It has two caterpillars that allow the robot to move in different environments and go over small obstacles. Of course, the election of this mechanical kit constrained future decisions, such as the needed battery to power up the robot or the layout and design of the control board.

2.2 Architecture configuration

Fig. 1 shows the general architecture of the ExaBot. The architecture has an embedded PC104 as central processing unit and 3 Microchip PIC microcontrollers: one to control most of the sensors, and one to control each DC motor included in the Traxster Kit.

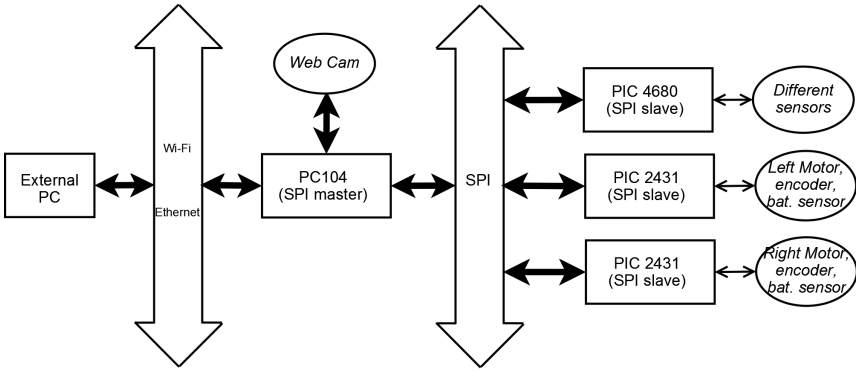


Fig. 1. ExaBot architecture

Embedded PC The Exabot is designed so it may be used as a platform for research experiences. In order for the robot to be autonomous, we included an on-board processing unit capable of executing artificial intelligence algorithms applied to robots such as automatic learning, neural networks, robotics vision, among others. The robot has an embedded ARM 9 PC104 of 200 Mhz TS-7250 [4]. This embedded PC has 2 USB ports, a serial port and an Ethernet port. It runs a Linux Kernel 2.24, and controls all the PICs, the communication with an external PC and also controls the webcam.

Communications The communication between the PC104 and the microcontrollers is by means of a SPI (Serial Peripheral Interface) bus. This is a full duplex, serial and synchronous communication bus between one master and many slaves. We chose it over other possible serial buses such as I^2C or RS232 since it is much simpler to implement and it meets exactly what we needed: fast communication between one fixed master (the PC104) and many slaves (all the PICs). To communicate with external PCs, the ExaBot has an Ethernet and a Wi-Fi connection provided by the PC104 and a wi-fi USB key.

Programming the robot To program the robot in this configuration is very simple since the PC104 has a Linux operating system. There are several open source cross-compilers and tool-chains to program embedded ARMs like the TS7250, for example using C++ and the gcc compiler.

2.3 Reconfigurable architecture

We included the embedded PC mainly to have enough processing power to run different algorithms for research purposes. However, there are several research purposes that don't need so much processing, and research is not the only goal of the ExaBot. There are many education and popularization activities for which the PC104 is not needed. For those tasks, we prepared the ExaBot to work without the PC104. In this way, the robot cost is significantly reduced (the PC104 is about 20 percent of the cost of the robot). The power consumption is also smaller, making it possible for the ExaBot to run autonomously longer. This alternate architecture is shown on Fig. 2.

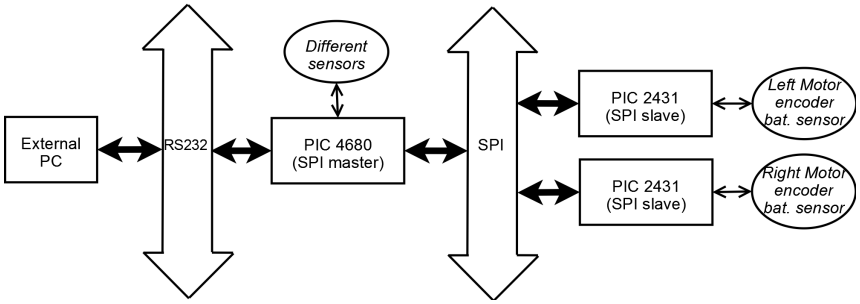


Fig. 2. ExaBot architecture without the PC104

In this configuration, the robot's control algorithm is executed in the PIC that controls the sensors (PIC18F4680).

Communications The communication between the PICs is the same SPI: the PIC18F4680 is prepared to be the master of the SPI bus. Communication with external computers changes: in the other architecture it was Wi-Fi or Ethernet from the PC104. We included an RS232 driver connected to the PIC18F4680 to provide an RS232 bus connection with external PCs if needed. This can be a wireless connection if a RadioMetrix module is added. In this way, algorithms may be run in external PCs and commands sent by a wireless radio connection to the ExaBot.

Programming In order to program the PIC18F4680 to execute different algorithms, we exported its programming interface, as well as the programming interface of the other PICs. There are several cross-compilers, tool-chains, IDEs and programmers to program these microcontrollers. This may prove very useful not only to run different tasks but also if changes to the base control of the sensors or motors are needed, or if new sensors are added (see section 3.3). Another option for high level control is to program the desired algorithm in an external PC and use the RS232 bus to send commands to the ExaBot.

2.4 Power

To power the complete robot, rechargeable Ion-Lithium batteries are used. Each cell has between 4.2 and 3.6 V, and delivers 1.9 Amp/hour. The battery has 3 cells in series that are regulated to deliver the 7.2 V needed to power the motors. In addition, 2 cells in series are regulated to 5V to power all the logic and sensors, including the PC104, PICs, telemeters, sonar, etc.

3 Sensors

A requirement of this development is that the robot may be used for a wide variety of applications. Hence, we included different types of sensors in its design. The ExaBot has bumpers, linefollowings, infrared telemeters, a sonar, battery sensors and a webcam to capture images. It also has an external port that may be used to connect several other sensors. Finally, the quadrature encoders and motor current consumption sensors prove very useful for the control of the motors. Most of these sensors can be dismounted and rearranged, turned off or even taken out of the robot if needed for particular tasks. In this way, we have a robot with the capability to control a wide variety of sensors, but that may use only a few for a particular task.

A PIC18F4680 [5] is used to control the sonar, linefollowing, bumpers and the ring of telemeters. The webcam is directly controlled from the embedded PC104. The encoders and motor current consumption sensors are used for motor control as explained in section 4.

In this section we will describe the sensors available in the ExaBot and most hardware related issues. The software architecture to control all these sensors is described in section 5.1.

3.1 Distance measurement

The distance measuring sensors are infrared telemeters and a sonar. The infrared telemeters are short range punctual sensors (the range varies depending of the model, but is less than a meter). They are relatively low cost. Sonars are non-punctual, long range (several meters) sensors, and more expensive. We chose to install a sonar in the front of the robot in order to achieve a long range vision in the front, and a ring of 8 telemeters to sense the environment surrounding the

robot in a short range. In this manner, the robot has both types of sensors but it is still low cost. Also, telemeters are faster than sonars.

Ring of telemeters The chosen telemeters are the Sharp GP2Y0A21YK0F [6] with a sensing range of 5 to 80 cm. The telemeter returns a voltage value proportional to the distance of the nearest object. This is a non-linear, analogical function. In order to digitalize this value, we use the A/D converter of the PIC18F4680. We also characterized the function before mounting the telemeters on the robot and thus built a linearizing array that given the digitalized voltage returns the lineal distance to the nearest object. Each telemeter gives a new value every 38 ± 10 ms, and has an unpredictable zone between measurements.

Sonar The chosen sonar is the Devanatech SRF05 [7] that has a sensing range of 10 mm to 4 meters. There are sonars with larger sensing ranges (like SRF10) but are considerably more expensive, which determined the choice. The sonar works in the frequency of 40 KHz (wavelength 8,65 mm). It can be triggered every 50 ms, giving a maximum of 20 measurements per second.

The sonar's output is a digital pulse whose duration is linearly proportional to the distance to the closest object. Therefore, to control it we use a CCP (Capture/Compare/PWM) module, and use a timer to measure the length of the sonar's pulse, thus having the distance to the closest object.

3.2 Other sensors

Linefollowing and bumpers The ExaBot has two line-following sensors and two contact sensors (bumpers). This sensors are connected to interrupt-on-change pins of the PIC18F4680. In this way, whenever a bumper is pressed or a line found, as the pin changes value, the interrupt is generated.

Battery sensors The ExaBot has two battery sensors, one for the motor power and one for the logic power. In this manner, it can be detected when the battery is running too low to continue with experiments.

The battery sensors sense the voltage provided by the batteries before the voltage regulators in order to sense its fall due to use. The idea is to enter that analog voltage into a PIC, digitizes it and then see how the battery is doing. As the voltage provided by the batteries is much more than the 5V the PIC needs as maximum input in its pins, we implemented a voltage divider in order to map those voltages to the 0-5V range. The battery control is actually implemented in the PICs used to control the motors, as the A/D converter of the PIC18F4680 is busy with the telemeters. Each motor PIC senses one of the batteries.

Webcam The ExaBot has a webcam that allows to capture images of the environment. The webcam is directly controlled from the embedded PC104 as shown in figure 1, using a driver of the Linux kernel 2.24 that runs in the embedded PC.

3.3 Sensor expansion ports

In the control of the telemeters, sonar, line-following and bumpers, and all the needed electronics to make the PIC18F4680 a possible master of the SPI bus (SPI chip select pins plus the control of the RS232 bus), all the 40 pins of that PIC are used up. However, the two PICs that control the motors have several pins that are not used. We planned the pins needed for motor control in those PICs in order to maximize the possible applications of exported pins .

In this way, the ExaBot has two expansion ports, each one exporting the following:

- one analog pin (to connect any analog sensor like infrared telemeters, light sensors, etc).
- one CCP pin (Capture/Compare/PWM module like the one used to control the sonar).
- a PWM pair (Pulse Width Modulation module, like the ones used to control the motors).
- the INT0 pin (external interruption pin, in order to program an external interruption).
- GND and Vcc.

These expansion ports may prove a very powerful tool to add sensors and functionality to the ExaBot. The programming needed to control these added sensors may be done with the exported programming ports for each PIC18F2431.

4 Motors

The ExaBot has two direct current motors of 7.2 V and 2 Amp/hour and built-in quadrature encoders that come in the Traxster kit. We used a PIC18F2431 [8] to control each motor and corresponding encoder.

To control the motor we used the PWM (Pulse Width Modulation) hardware module in the PIC connected with a L298 driver [9]. In this manner, depending on the length of the generated pulse, the motor goes faster or slower. The length of the needed pulse (that is, the duty) is calculated by a PID (Proportional Integrative Derivative) control algorithm using the required velocity, and the information of the encoders. This closed loop control algorithm is better described in the software architecture section 5.2.

To sense the encoders, the Quadrature Encoder hardware module of the PIC is used.

4.1 Current sensor and fault circuit

We implemented a motor current sensor in order to have a control of the current the motors are actually consuming and in that way implement a load control, and also a fault circuit.

To sense the current we used a ACS712 IC [10]. This IC is connected between the L298 driver and the motor, senses the current and outputs a voltage

indicating the current. This voltage is used in two ways. For one, it is an analogical input to the PIC, that is digitalized and used to know how much current is consumed. The software for this is further explained in section 5.2.

We also use that voltage to implement the fault circuit in order to override all PWM output and hence stop the motors if error conditions happen. The problem arises when a motor is jammed for some reason. In those cases, the encoder shows the motor is not moving, then the PID makes the PWM give full power to the motor. In that case, the motor consumes all the available current from the battery, and if not turned off surely something will be burnt. To prevent this from happening, the sensed current inputs a LM319 voltage comparator [11] that is set so a 3 Amp threshold is not surpassed. If that threshold is passed, the Fault pin of the PIC is driven low, and the Fault module by hardware overrides the PWM output to stop the motor. We also have fuses in the circuit path from the batteries so if everything fails, the fuse will be burnt and not the circuit, ICs or motor.

5 Software architecture

In this section we describe the software architecture of the ExaBot, in the hardware configuration that includes the PC104. We will describe the software in the PC104, in the PIC18F4680 that controls the sensors and in the PIC18F2431 that controls the motors in three different subsections.

All the software is programmed in a modularized way, making full use of the interrupt hardware available in the PICs and and the thread software in the embedded PC. In this manner, we have a modularized, completely timed design in which all sensing, motor control and communication happen at determined times. The real time control and perfect timing of sensor data and commands in robot control is a key issue, since the robot moves in the real world. If commands or sensing is delayed, or happen at different unknown intervals, we don't have a way of knowing if the conditions have already changed and hence the commands are no longer useful or may even be harmful. That's why we put much effort in a correct software architecture design and careful programming, and in this way, we manage to control all the hardware described in the previous section.

5.1 Software in the PIC18F4680

As we have already explained, the PIC18F4680 controls most of the sensors: the sonar, the ring of infrared red telemeters (IRs), the line-following and the bumpers. All these sensors may be turned on or off by the high level algorithm through software commands. In the hardware configuration including the PC104, this PIC is just an SPI slave.

Fig. 3 shows the modules and interruptions (ISR) of the software architecture of this PIC.

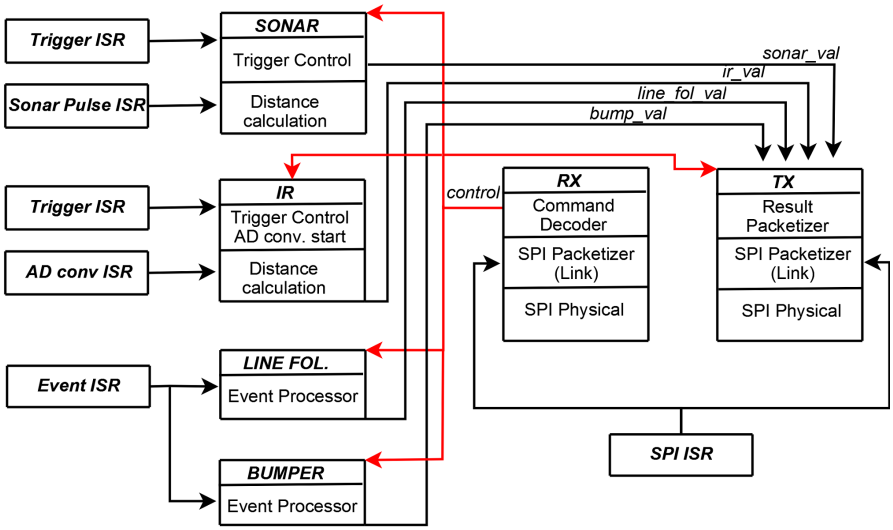


Fig. 3. PIC18F4680 Software Architecture

Modules and submodules in the sonar control

- **Trigger ISR**: This interruption tells when the Sonar must be triggered.
- **Trigger Control**: This submodule triggers the Sonar, and resets the needed timer for the next trigger ISR.
- **Sonar Pulse ISR**: This interruption stores the moment that the sonar pulse starts and ends, and tells the Sonar Module when the pulse is over.
- **Distance Calculation**: This submodule calculates the sensed distance using the information acquired by the Sonar Pulse ISR.

In this way the sonar is triggered and sensed every 50 ms as required by the sonar's data sheet.

Modules and submodules in the infrared telemeters control As the PIC18F4680 has only one A/D converter, we implemented a round-robin algorithm to control the ring of telemeters. This algorithm triggers and reads the telemeters obtaining a new value every 36 ms. It is optimized to trigger only the telemeters that are in use, and only during the minimum time needed to obtain a correct value. In this manner, power consumption is reduced and the overall time needed to obtain a value for each telemeter that is actually in use is the minimum possible. As between measurements the telemeters have an unpredictable zone, the lapse between the trigger and the start of the A/D conversion is timed to assure the measurement is correct. The submodules in this module are:

- **Trigger ISR**: This interruption tells when the next IR must be triggered, and the current IR is ready for Analogical/Digital conversion.
- **Trigger Control**: This submodule triggers the next IR (following the round-robin algorithm), and starts the A/D conversion of the telemeter that is ready.
- **AD Conv ISR**: This interruption tells when the A/D conversion is done.
- **Distance Calculation**: This submodule calculates the sensed distance using the digital value given by the A/D converter, and the pre-calculated linearizing function discussed in section 3.1.

Modules and submodules in the linefollowing and bumper control

- **Event ISR**: This interruption checks which line following or bumper has set the interrupt-on-change pin.
- **Event Processor**: This submodule process the line following or bumper that has been set.

Submodules of data reception (RX)

- **Command Decoder**: This submodule decodes the commands received (for example, which sensors turn on or off).
- **SPI Packetizer** (Link Layer): This submodule receives each byte from the Physical Layer and reconstructs the SPI packet.
- **SPI Physical Layer**: This submodule receives each byte physically.
- **SPI ISR**: This interruption tells the SPI packetizer when the Physical Layer has a new byte to be read. It is only triggered when the packet is for this PIC.

Submodules of data transmission (TX)

- **Result Packetizer**: This submodule takes the sensor values from each sensor module, and constructs the packets that will be sent to the PC104. It knows which sensors are on and sends only their data, and the corresponding control bytes for the PC104 to understand.
- **SPI Packetizer** (Link Layer): This submodule constructs the packets for SPI communication.
- **SPI Physical Layer**: This submodule sends each byte physically.
- **SPI ISR**: This interruption tells the SPI packetizer when the Physical Layer is ready to receive a new byte to be sent. It is the same interrupt for transmission and reception since in the SPI protocol, whenever a byte is sent another one is received.

5.2 Software in both PIC18F2431

As we have already explained, each PIC18F2431 controls a motor, and the battery sensor. Fig. 4 shows the modules and interruptions (ISR) of the software architecture of this PIC.

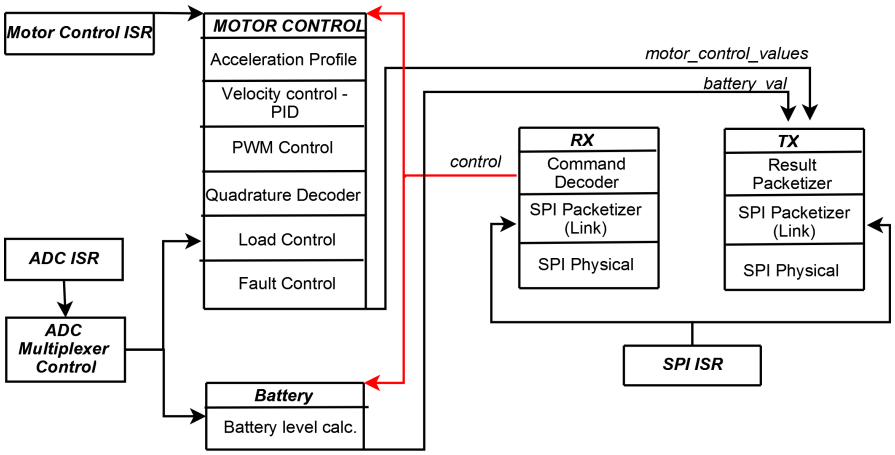


Fig. 4. PIC18F2431 Software Architecture

Modules and submodules in the motor control

- **Motor Control ISR**: This interruption tells when every submodule of the motor control must be executed, and thus the control of the motor is done.
- **Acceleration Profile**: This submodule executes an acceleration profile.
- **Velocity Control**: This submodule does the velocity control of the motor. It implements a PID - Proportional Integrative Derivative control. This is a closed loop control that uses the information of the encoders given by the Quadrature Decoder submodule, and calculates the length of the duty and the direction of the motor that is the information needed for the PWM submodule to control the motor driver.
- **PWM Control**: It controls the PWM hardware module of the PIC that produces the necessary output to control the motor driver. All the needed variables to obtain the calculated velocity are set by the PID submodule.
- **Quadrature Decoder**: it controls the quadrature encoders and provides their value for the PID submodule to calculate.
- **Load Control**: it controls the power consumption of the motors in order to check their load. If its too high, it overrides the PID directives to the PWM submodule in order to decrease the duty, and therefore decrease the motor's power consumption.
- **Fault Control**: As explained in section 4, if the motor passes a 3 Amp power consumption threshold, the fault logic by hardware overrides the PWM output stopping the motors. The software fault control submodule checks if this condition has happened and informs it to take the necessary measures.

All the control cycle of the motor is perfectly timed and occurs at a 125 Hz frequency (that is, 125 times per second).

Modules in the ADC multiplexer control and battery sensor As explained in sections 3 and 4, the motor current needed for the load control and the battery are both sensed using an analogical input and the A/D converter of the PIC18F2431. Since this PIC has only one ADC, a multiplexer control must be done in order to sense both values. In this case, there is no need for triggers as in the telemeters control explained in 3.1 because both sensors are in work all the time (they are hardware implemented sensors). The modules and submodules associated with this control are:

- **ADC ISR**: it tells the ADC Multiplexer control when to sense the following sensor.
- **ADC Multiplexer Control**: it is in charge of multiplexing the battery and the current sensors for A/D conversion. The current sensor converted value is then used by the load control, and the battery sensor by the battery level calculation.
- **Battery Level Calculation**: using the digital converted value it calculates the battery level.

Modules and submodules in the communication control The RX and TX communication is done with the same modules and submodules that in the PIC18F4680. The only difference is in the Command Decoder and the Result Packetizer submodules, since the commands to the motors differ from those to the sensors, and the resulting feedback also differs.

5.3 Software in the PC104

The PC104 is the master of the SPI communication, controls the communication with an external PC and the webcam. Fig. 5 shows the modules and interruptions of the software architecture.

Modules and submodules in the SPI communications control (RX and TX) The structure for the communications (reception and transmission) follow the OSI model (Open System Interconnection Reference Model), resulting in a layered network protocol. Our implementation in the PC104 has application, net, link and physical layers. In the PICs the communications is also implemented following the OSI model, but because they are always slaves it only has application, link and physical layers.

- **Sensor Info Decoder**: This submodules decodes the sensor information received for the main application to use. This information is read from the buffers in the net layer. This is the application layer for the RX OSI stack.
- **Command Encoder**: This submodules takes the commands issued by the main application and forms the protocol packets to be sent to the PICs. It also puts them in the correct buffers in the net layer. This is the application layer for the TX OSI stack.

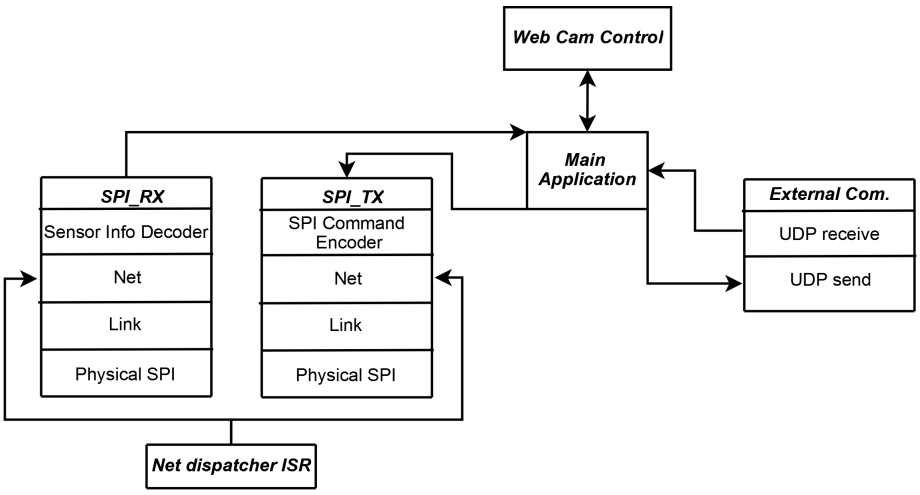


Fig. 5. PC104 Software Architecture

- **Net submodule**: This submodule is in charge of the network, managing the reception and sending of packets to the three slaves. For each slave it stores two buffers: one for received packets and another one for packets that need to be sent.
  - *Dispatching Routine* we implemented a round-robin dispatching routine that runs every time the network dispatcher ISR tells it to. This routine knows which was the last slave it sent commands to and copies the received packet for that slave from the link buffer to the slave's received buffer in the net layer. Then it sees if there's anything to send to the next slave: if there is, it sends it and returns. If there isn't, it checks the next slave and so on until it either finds something to send or checks that there is nothing to send to any slave.
  - *The RX submodule* exports several functions to get packets from the reception buffers.
  - *The TX submodule* exports several functions to put packets in the sending buffers.
- **Net Dispatcher ISR** this interruption wakes up the network dispatcher routine 100 times per second, making the SPI communication between the embedded PC and the PICs perfectly timed.
- **Link submodule**: The TX submodule is in charge of constructing the SPI packets to be sent and put them in the physical layer buffer. The RX submodule is in charge of forming the SPI packets from the received bytes in the physical layer, getting them ready for the Net layer to get.
- **SPI Physical submodule**: The Physical submodule manages the hardware module that sends and receives SPI bytes. It also configures and manages

the DIO (Digital Input Output) port pins of the embedded PC in the needed way for the SPI to work.

In this way, the SPI communication with the PIC is perfectly timed, sending and receiving 33 packets per second to each slave.

External communication We used the UDP (User Datagram Protocol) to communicate with an External PC. We found this protocol much more suitable than TCP for control from an external PC mainly because the UDP protocol sends packets as they are received, while the TCP protocol buffers packets until it has enough information to send. As we have already discussed, in robot control the timing of the commands and the return information is vital. That's why all our design stresses the importance of having perfectly timed sensing, control, communication. For robot control, receiving all the commands together and late as would happen with a TCP buffered connection is far worse than eventually loosing some command because the UDP connection doesn't guarantee packet delivery. If the robot receives ten commands together, it will execute them all but in a late moment when the environment conditions may have already changed, making those commands useless or worse. That's why we implemented an UDP connection. The implemented submodules are:

- **UDP Receive** is in charge of receiving the UDP packets from the external PC.
- **UDP Send** is in charge of sending the UDP packets to the external PC.

Main application The main application is the piece of software that unites all the previous pieces, and executes the programmed algorithm. It interprets the received sensor's data and webcam images (if running) and sends the relevant information to the external PC by means of the UDP Send submodule. It also receives commands from the external PC by the UDP Receive submodule, executes what is ordered and sends the necessary commands to the PICs using the SPI TX module.

Process programming In order for all these modules to work in a timely manner, we took a process programming approach. Currently, there is are four child processes running in the embedded PC, in order to perform both communication protocols in a timely fashion (UDP and SPI), and also run the main application:

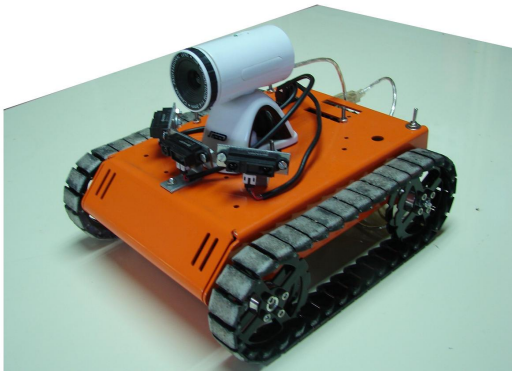
- **UDP Receive process** is always listening in the socket to see if the external PC sends something. Evidently, if we didn't program this as a separate process, all that our program would do is listen to this socket. When this process receives something, it copies the data to a shared memory location that the main application can read.
- **UDP Send process** is in charge of sending UDP packets. It reads the needed data from a shared memory location that the main application can write.

– **SPI Net control process** as we have already explained, the net dispatcher routine runs 100 times per second, and it is launched by an interruption. Therefore, this SPI network control runs in a separate process. It reads and writes the data from/to the net layer buffers we already explained. This shared memory locations are accessible from the application layer of the OSI stack. The SPI TX and RX communication run in only one process because both aspects are intimately related in the SPI protocol (that is, every time a byte is sent, a byte is received and the only way to receive from a slave is by sending a packet).
– **Main application process** The main application runs in a separate process and uses the shared memory locations to communicate with the other processes, and therefore exchange data with the external PC and the PICs.

6 Results

The main result of this work is the first prototype of the ExaBot: a mobile mini-robot architecture for research, education and popularization of science.

We have achieved our goal of building a relatively low cost (around usd 600), multi-purpose robot, with a reconfigurable sensor and hardware architecture. The software architecture and programming mode also ensures a perfectly timed control. The PC104 communicates with each PIC 33 times per second. In the sensor control, each telemeter is sensed every 36 ms, the sonar every 50 ms, the sensing for the line-following and bumpers happen every time they have a change. Finally, the control cycle of the motor is executed 125 times per second.



Main characteristics

– Dual-motor mobile chasis, 229 mm length x 203 mm wide x 76 mm height.
– Reconfigurable hardware architecture, Reconfigurable control architecture.
– Interrupt based software architecture, completely timed.
– Sensing capabilities: Ring of 8 telemeters (range 50 mm to 80 cm), Sonar (range 10 mm to 4 meters), 2 linefollowing, 2 contact sensors, color webcam 640x480, wheel encoders.
– Ion Lithium Battery (2.5 - 4 hours)
– Embedded PC with a 200 Mhz ARM9 CPU, 32 MB SDRAM, 32 MB on-board flash drive, 2 USB ports, 2 serial ports, 1 Ethernet port and wi-fi. Linux 2.4 support (hope to upgrade to 2.6)
– Open Source design with full access to source code and schematics.
– Fully programmable for autonomous operation.
– Wireless remote control up to 100m indoors (line of sight)

Fig. 6. ExaBot with 3 infrared telemeters and webam mounted

The applications for which the ExaBot is designed may be divided in three areas.

- Research: autonomous robots, robot learning (reinforcement learning, neural networks, etc), multi-robots systems, collective behaviors, sensors fusion, vision based navigation, among others.
- Education: it can be used in several undergraduate courses such as Robotics Vision, Machine Learning, Neural Networks among others.
- Popularization of science: it can be used in robotics workshops that include sensing and motors (for example Braitenberg vehicles), robots competitions (for example line tracing, sumo, etc), among others.

The ExaBot has already been successfully used in each one of those areas. In the next subsections we show an example of it's use in each area.

6.1 Research

The ExaBot is being used as a platform to develop a navigation technique in unknown environments by members of our laboratory. The problem was divided in three parts: obstacle avoidance, map construction and continous localization of the robot in that map. The aim is to find a robust solution in each of these problems that doesn't require a complex sensing system. For that reason, the ExaBot was configured to use only a ring of six telemeters, and the encoder's information.

For obstacle avoidance, a multilayer perceptron neural network was used. It has 6 input neurons (one for each sensor), 3 output neurons (one for each possible direction: left, right, forward) and a hidden layer with 3 neurons. The network was trained off-line using a training set composed of sensor values from the environment, and the correct action expected in that situation.

For map construction, a topological graph was used where each node represents a free area, and each edge the spatial connectivity between those areas. Each node has information about its approximate location in space. Every time the robot finds a new area in the environment, it creates a new node. Once the complete map of the environment is created and the robot is localized, the navigation between one point and another is decided by a shortest path graph algorithm (for example, the well known Dijkstra algorithm).

For the self localization problem, the robot has odometry information provided by the encoders. This information is not completely right, since odometry error builds up in time. It also has the map built to the moment with an approximation to where each node is really located, and the telemeter's information. The problem is that the approximated location is also derived from odometry and therefore error prone. To solve this problem, it is necessary to integrate the imprecise information from all those sources and thus obtain an estimator for the robot's localization and the nodes in the map. This is done with the application of a Kalman filter.

This is an ongoing work, where experiments are still being carried on and will be published in an specific paper.

6.2 Education

The ExaBot was successfully used in two undergraduate courses of our Computer Science Department (Departamento de Computación, Facultad de Ciencias Exactas y Naturales, UBA). In an introductory course to Computer Architectures, the students were introduced to microcontroller architecture and programming using the ExaBot as a platform. We showed the programming of the ExaBot's microcontrollers to control sensors and motors. In a Neural Network course the ExaBot was used to experience with different neural networks that allowed the robot to avoid obstacles.

6.3 Popularization of science

During 2009 we successfully used the ExaBot in workshops for high school students, particularly in a nine week workshop organized as a part of the popularization of science programs of the FCEyN, UBA. In this workshop, we taught the basic concepts of Behavior Based robotics and the Braitenberg model as a technique for programming robots. Following this model, students were able to program different behaviors, first in a simulator and then in the ExaBot. Those behaviors were presented as a way to solve problems using robots, starting from some premises, proposing hypothesis and then proving (or disproving) them in experiments comparing the expected results with the real ones. To close the workshop, students prepared scientific posters with their experiments and results that were presented together with the posters of other workshops. In this manner, we could corroborate the use of the ExaBot as a platform for Educational Robotics activities.

7 Conclusions and ongoing work

In this work we presented a new mobile mini-robot, the ExaBot, a robot for research, education and popularization of science. The design follows the premise that the robot must be usable to pursue very different tasks, must be reconfigurable and relatively low cost compared to commercially available robots with similar goals.

The hardware architecture is reconfigurable to include enough processing power to allow the on-board execution of intelligence algorithms or to pursue simpler algorithms without the embedded PC, reducing the robot's cost and power consumption. It has a wide variety of sensors that can be reconfigured, even taken off, and it also has expansion ports that may be used to add new sensors. All the programming ports are exported. The robot may be programmed and controlled in three levels: from an external PC, in the embedded PC or from the sensor PIC. The software architecture is specially designed in a modularized way, making full use of hardware interruptions and software processes to have a perfectly timed control of the robot. These characteristics make of the ExaBot a very malleable robot, fulfilling its goal to serve for research, education and popularization of science tasks.

There are several additional works that members of our laboratory are undertaking, specially the development of specific software for these robots.

As one of the applications for the ExaBot is to use it in workshops and popularization of science activities with high school students, the development of a user friendly interface that allows the programming of the robots in a graphical and simple manner is much desired. Our group is developing such an interface to program the ExaBot by defining basic behaviors that are integrated to form more complex behaviors, by means of a data flow programming approach rather than a control flow one.

Another additional development of importance is a 3D Simulator. To face research tasks a controlled simulated environment is a must: an environment where the researcher may prove his/her algorithms before real world experiences is needed.

References

1. Khepera, Khepera II and Khepera III, K-Team, http://www.k-team.com
2. Mindstorms, Lego, http://mindstorms.lego.com
3. Traxster Kit, http://www.roboticsconnection.com/p-15-traxster-robot-kit.aspx.
4. TS-7250 datasheet, http://www.embeddedarm.com/ epc/ts7250-spec-h.html.
5. PIC18F4680 datasheet, http://ww1.microchip.com/downloads/en/DeviceDoc/39625c.pdf
6. Sharp GP2Y0A21YK0F Distance Measuring Sensor datasheet, http://www.pololu.com/file/ download/gp2y0a21yk0f.pdf?file_id=0J85
7. Devanatech SRF05 Ultrasonic Ranger documentation, http://www.robot-electronics.co.uk/htm/ srf05tech.htm.
8. PIC18F2431 datasheet, http://www.datasheetcatalog.org/datasheet/microchip/39616b.pdf
9. L298 IC driver datasheet, http://www.st.com/stonline/books/pdf/docs/1773.pdf
10. ACS712 sensor current 5A 5V datasheet, http://www.allegromicro.com/en/ Products/Part_Numbers/0712/0712.pdf
11. LM319 IC comparator, http://www.fairchildsemi.com/ds/LM/LM319.pdf