# Using Robotics as a Motivational Tool: An Ada Interface to a Pioneer Robot*

Rigoberto Chil, Diego Alonso, Francisco Ortiz, Juan Pastor

Division of Systems and Electronic Engineering (DSIE)
Technical University of Cartagena, Campus Muralla del Mar, E-30202, Spain
diego.alonso@upct.es

**Abstract.** The new European Higher Education Area encourages student centred learning, which puts the focus on the learner and his needs, rather than being centred around the teacher's input. This paper presents an initiative that revolves around the use of a real robot and a robot simulator with two main objectives: make learning programming languages more appealing to undergraduate students, and to have a platform that can be still used in postgraduate and master courses. The interface with the simulator and the real robot has been programmed in Ada, and it is also being used in our current Research and Development projects.

## 1 Introduction and Motivation

Motivating students when teaching programming languages is an arduous and difficult task. One of the main reasons for this, in our opinion, is that the simplicity and limitations of the used examples and proposed problems do not motivate students enough to go into knowing programming languages in depth. Also, they give students the false impression that what they are learning is of little use, since all they do is solving simple calculations, like quadratic equations, calculating Fibonacci numbers, a number's factorial, bubble sort ten numbers, etc.

The purpose of the Bologna Process [1] is to create the *European Higher Education Area* (EHEA) by making academic degree standards and quality assurance standards more comparable and compatible throughout Europe. The Bologna Process currently has 46 participating countries, whereas there are only 27 Member States of the European Union. The new EHEA encourages the development and adoption of new learning techniques [2], such as autonomous and problem-based learning [3] for students. In this new reality, the kind of exercises described above does not meet the requirements of the EHEA. This situation is even more important when considering higher and master courses, since students must then apply transversal skills, such as teamwork ability, ability to put knowledge into practice, adaptation to new situations, etc.

Laboratories and practical classes play a crucial role in the curriculum of scientists and engineers [4], and the acquisition of practical skills is one of the main requirements of the Bologna process. In this vein, it is worth mentioning related initiatives such as the one described in [5], where the authors describe a laboratory for teaching Robotics.

* This work has been partially supported by the Spanish CICYT Project EXPLORE (ref. TIN2009-08572) and the Séneca Proyect MISSION-SICUVA (ref. 15374/PI/10).

With this in mind, we think that building simple robotics control programs will allow us to achieve a number of closely related objectives. Firstly, we think that the use of a simulator will increase students motivation as well as show them the usefulness of what they are learning. Secondly, using a simulator enables us to propose problems of different size and levels of complexity, according to the students level, without being limited by the access to a single robot. Lastly, Robotics is a traditional research area for the DSIE Research Group, and in this sense, using a robot or a simulator enables us to transfer part of our research to students, as well as to attract interested students.

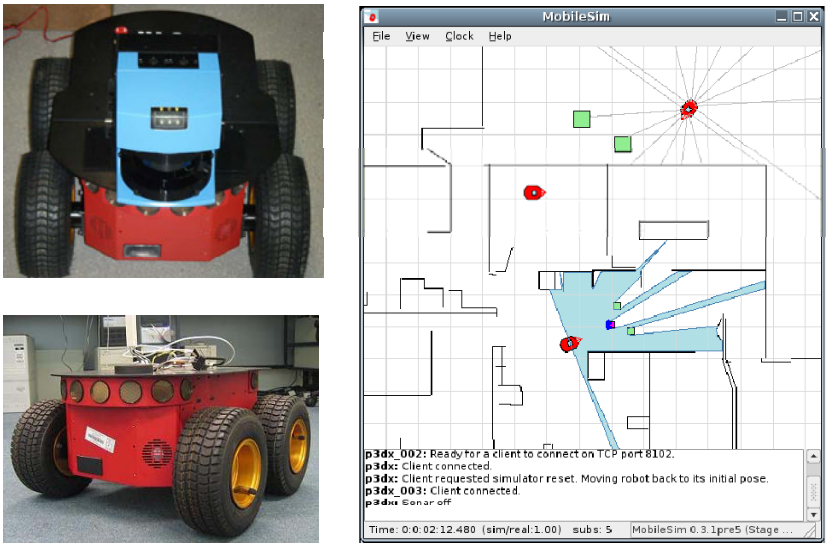
On the other hand, the use of simulators is justified by itself, since before these robots are put to work, they need to be tested under different conditions. As mobile robots are used extensively for their ability to navigate and perform tasks in unstructured environments (space exploration, military surveillance, nuclear power industry, security, etc.), simulators play a key role during these early stages of the robot development, as they will give designers an idea of how the robot is going to behave [6]. Their use can avoid the robot suffering damage in the early tests, while it makes it possible to have many people working in parallel, testing their algorithms against their own simulators. This last advantage is crucial for teaching, since students can safely and concurrently crash their virtual robots while testing their programs.

In 2010 we started teaching the subject "*Applied Computing*" in the first year of the *Bachelor in Industrial Electronics and Automation Engineering* [7], and in two years' time we will start teaching "*Real-Time Systems Programming*". Finally, we also teach "*Software Development for Real-Time Systems*" in *Master in Information and Communication Technologies* [8]. And we plan to use the simulator and the real robot in all of them.

Our research background began in the early 90s., integrating new paradigms in the service robot development process as they emerged [9,10]. During the early years (1993–1998), our efforts were directed at the development of software for various kinds of tele-operated robots to perform maintenance tasks in nuclear power plants; during a second phase (1999–2006),we built applications for ship-hull cleaning robots. All this time, we have been applying all the possibilities offered by Software Engineering, from the use of Structured and Object-Oriented programming paradigms in early developments, to the recent adoption of the Model-Driven Software Development (MDSD) approach.

Nowadays, we are involved in the EXPLORE Research and Development project, whose main objective is to develop and implement a set of methods and tools for real-time systems development incorporating design patterns, a component-oriented approach, and MDSD techniques for the design and validation as well as for the automatic code generation for the target platform. We use three robots as case study: a small robot developed by us, a golf cart also modified by us, and a commercial Pioneer 3–AT robot. And we want to use the same interface with all of them. This reason led us to develop the application described in this paper.

The Pioneer robot has helped us uniting our teaching and researching facets. On the teaching hand, its manufacturer provides a simulator that has the same interface and that accepts the same commands with the same protocol as the real robot. Thus, we have an excellent platform in which students can test their programming skills. On the other hand, the Pioneer is an excellent and robust platform that already provides the



```
p3dx_002: Ready for a client to connect on TCP port 8102.
p3dx: Client connected.
p3dx: Client requested simulator reset. Moving robot back to its initial pose.
p3dx_003: Client connected.
p3dx: Sonar off
Time: 0:0:02:12.480 (sim/real:1.00)   subs: 5   MobileSim 0.3.1pre5 (Stage ...
```

**Fig. 1.** Pictures of the Pioneer P3–AT robot (left) and screen shot of the MobileSim simulator (right). Simulator screen capture extracted from `http://robots.mobilerobots.com/wiki/MobileSim`

low-level control facilities, which allows developers to focus on higher-level aspects. Fig. 1 shows a couple of pictures of the robot and a screen-shot of the simulator.

This paper presents the design and implementation details of the already mentioned interface with both the real Pioneer P3–AT robot [11] and the MobileSim simulator [12] in Ada, as a continuation and improvement of a previous work [13]. The following two sections are devoted to present a brief review of the technical background, and to describe the developed interface with the Pioneer robot and its simulator. Section 4 presents some examples of the kind of applications that students must develop, depending on the course level. And finally section 5 concludes and presents some future work.

## 2 Technical Background: Hardware and Software Involved

Since the target robot of the software described in this paper is a *Pioneer 3-AT* mobile robot (P3–AT), it is worth describing first its main characteristics, the software installed in the robot and the available software for controlling it.

The Pioneer family of robots is sold as a research tool, used in many universities and companies around the world as the main physical research platform. These kind of robots are programmable intelligent platforms equipped with the basic devices for navigation and sensing in the real world. They are part of an extensive family of robots released in 1995 by the company Mobile Robots [14]. Specifically, the P3–AT is provided with high resolution motion encoders with inertial correction to compensate for skid steering (that is, when the wheels skid encoders are still counting revolutions even though the robot is not really moving as the encoders indicate), reversible DC motors and motor controllers, as well as the four-wheel skid steer which carries out the balanced drive system of the robot. The P3–AT robot can carry a payload of up to 40 kg, reaches speeds up to 0.7 m/s, it can climb steep up to 45% grades and sills of 9 cm. The P3–AT is equipped with eight front and eight rear sonars, and a SICK laser scanner that senses obstacles from 15 cm to 7 m. In order to handle the low-level control details of the mobile robot (e.g. maintaining the platform's drive speed and heading, acquiring sensor readings, etc.), the P3–AT uses a high-performance 32-bits micro-controller with the embedded robot control software developed by its manufacturer, ARCOS (*Advanced Robot Control and Operations Software*).

The Pioneer 3–AT requires a PC to run client software for intelligent robotics command and control operations. As shown in Fig. 2, the robot follows a client-server architecture: ARCOS operates as a server that manages all the low-level details of the mobile robot, while the client role is played by the software running on a computer connected with the robot micro-controller via the host serial link. The high-level functionality and behaviour of the robot, such as obstacle avoidance, path planning, features recognition, localization, navigation, and so on, must be provided by the client software.

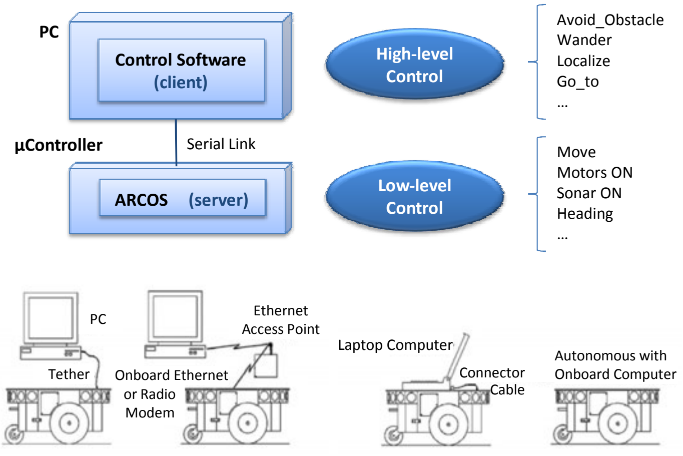


**Fig. 2.** Some possibilities to connect a client PC to the robot server

As said in the introduction, the manufacturer also offers *MobileSim*, a 2D simulator for their robots, together with its C++ source code distributed under the terms of the GNU General Public License. Communication with the simulator is done through TCP/IP using the same protocol as with the real robot (ARCOS commands), although not each and every feature and command are currently supported by the simulator. The simulator is also capable of simulating several robots, which are controlled through different network sockets. This feature provides additional possibilities to the use of the simulator, since it is now possible to develop applications where the simulated robots must collaborate.

While using ARCOS for the low-level control of the robot is mandatory, there are two main options for selecting the client software, namely: (i) use any of the available robotic frameworks, including the one provided by the manufacturer, or (ii) write your own robotics control software in order to have greater control of its behaviour. After describing briefly in the following paragraphs the advantages and disadvantages of both options, we will justify why we have adopted the second option.

Specific middleware and framework technologies for developing software for autonomous mobile robots have matured, evolved and improved during the last years. They all facilitate design reuse and provide the typical functionality that is common in the domain. The fact that robots from different manufacturers have completely different hardware platforms and their own development environments, has forced developers to pay special attention to the abstraction of the lowest levels, trying thereby to design a common API for programming them all. Thus, many manufacturers provide development platforms for their robots, like ARIA (for the robots built by MobileRobots), Mobility (for the robots developed by iRobot), Open-R (for the Sony Aibo), etc. There are also other initiatives that are not promoted by any manufacturer, such as Player/Stage, which provides a common abstraction for many robotic platforms that facilitates the development of control applications. It is possible to find a very detailed comparison of their characteristics in [6].

These frameworks standardize and simplify the low-level control aspects of the chosen robot. They usually provide access to abstract sensors and actuators, which exhibit a simpler interface and greater functionality than directly accessing the hardware through the operating system. Further raising the level of complexity, other frameworks such as OROCOS, CARMEN, ORCA, and MARIE, to mention a few, include features commonly used in robotics, such as control algorithms, localization and mapping, safe navigation, etc. A summary of the main features of these kind of high-level frameworks together with a comparison chart of their characteristics can be found in [15,16]. Finally, the manufacturer of the P3–AT offers ARIA (*Advanced Robotics Interface for Applications*), which provides a client-side interface to a variety of intelligent behaviours already programmed. ARIA is released under the GNU Public License, fully documented C++, Java and Python libraries and source code, but there is no Ada version available.

Though robotics frameworks provide much tested and working code, which enables developers to start programming the robot easily and quickly, there are some cases in which you want to have strict control of what the robot is executing, or in which using a framework is not suitable. For instance, frameworks such as ARIA offer typical algorithms for localization and mapping, but they offer no temporal guarantees and suffer the "inversion of control" (or the "Hollywood principle") problem [17], in which the flow of control of a system is inverted in comparison to procedural programming. That is, the framework code deals with the program execution order, but the business code is encapsulated by the called subroutines (developed by framework users). Thus, developers have no control over tasking characteristics, e.g. number of tasks, type, priority, etc., since they are determined by the framework design.

At the educational level, ARIA is written in C++, and though we could have used the facilities for interfacing with other languages that Ada offers, as described in [18], we decided not to do that. In that work, a Player-Ada binding is built using import C facilities. We did not do something similar because it would still present the inversion of control problem, and because using the ARIA framework would still be complex, making the student feel overwhelmed by the overall structure of the framework.

Besides, the objective we pursue in our courses is not teaching students to manage any of these frameworks (which might be of interest to other subjects more related to robotics), but using the robot as a platform for setting out different types of problems. Thus, we simply need to have direct access to the basic functionality offered by the robot, as discussed in the next section. In such cases, it is necessary to control the P3–AT robot simply via the ARCOS client-server interface.

As an added value for our research and for the robotics community, we offer a very simple Ada interface, which is as transparent as possible, so that developers know at all times what data is being used and also have complete control over tasking issues, e.g. their number, periods, priorities, etc.

## 3 Architecture of the Application

As explained in the previous section, in order to have strict control of the execution of tasks in the robot, we decided to develop an abstraction layer to access directly to ARCOS. Fig. 3 shows a deployment diagram of the two considered scenarios: direct access to the real robot through a serial port, and communication with the simulator through network sockets.

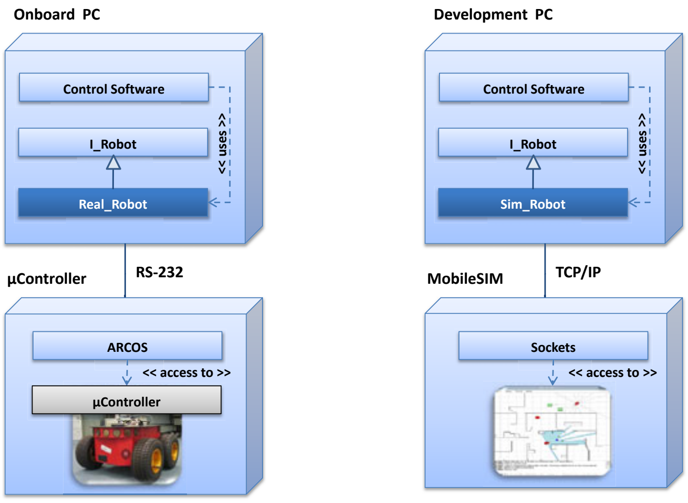


**Fig. 3.** Deployment diagram depicting the two considered scenarios: communication with the real robot and the simulator

As said before, one of the key requirements in the development was that the same program should drive the real and the simulated robots with minimum changes to the code. This is possible because the MobileSim simulator shares the same commands with the real Pioneer robot. Thus, we decided to create an interface (`I_Robot`, see Figures 3 and 4) that provides the functionality that is shared by both the simulator (`Sim_Robot`) and the real robot (`Real_Robot`), which are implemented as types derived from the already mentioned interface (see Fig. 5). These two types were implemented as protected objects, since they ensure mutual exclusion in the reading and writing access to

```
package Pioneer_P3at is
   type T_Robot_Comm is (Serial, Socket);
   type Rt_Robot_Comm_Config (Comm : T_Robot_Comm) is record
      case Comm is
         when Serial =>
            Port_Number  : Positive := 1;
            Rate         : Gnat.Serial_Communications.Data_Rate :=
                                  Gnat.Serial_Communications.B9600;
         when Socket =>
             Address_Rec : GNAT.Sockets.Sock_Addr_Type := (Addr => Inet_Addr
            ("127.0.0.1"), Port => 8101, Family => Family_Inet);
      end case;
   end record;

   type I_Robot is synchronized interface;
   procedure Read_Data (This : in out I_Robot) is abstract;
   procedure Begin_Comm (This : in out I_Robot; Comm_Parm :
         Rt_Robot_Comm_Config) is abstract;
   procedure Write_Data (This : in  out I_Robot; Buffer : in Ada.Streams.
         Stream_Element_Array) is abstract;
   function Get_Posx (This : in  I_Robot) return Integer is abstract;
   function Get_Posy (This : in  I_Robot) return Integer is abstract;
   ...
end Pioneer_P3at;
```

**Fig. 4.** Definition of the synchronized interface and its (abstract) subprograms. Only 4 subprograms are shown, but a total of 14 have been defined.

```

package Pioneer_P3at.Real_Robot is
   type Pioneer_P3at is synchronized new I_Robot with private;
private
    protected type Pioneer_P3at is new I_Robot with
       overriding procedure Read_Data;
       overriding procedure Begin_Comm (Comm_Parm : Rt_Robot_Comm_Config);
       overriding procedure Write_Data (Buffer : in Ada.Streams.
            Stream_Element_Array);
       overriding function Get_Posx return Integer;
       overriding function Get_Posy return Integer;
       ...
    end Pioneer_P3at;
end Pioneer_P3at.Real_Robot;
```

**Fig. 5.** Definition of the synchronized type implementing the communication with the real P3AT robot through a serial port. The Ada package for interfacing with the simulator is similar to this one.

the serial port and the network socket, through which the communication with the real robot and the simulator takes place. In this case, we decided to use the synchronized interface facility provided by Ada 2005 in order to implement the common interface and both protected objects. We created a totally passive structure comprising two protected types, since we decided that concurrency issues should be considered in higher layers by the code using the provided protected types. In this vein, we have a very versatile code, since it can be used in both single and multi-tasking applications.

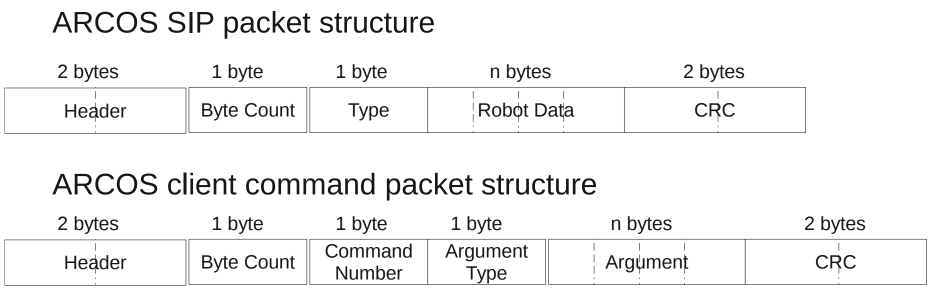


**Fig. 6.** Structure of ARCOS commands: SIP command sent by the robot (top) and command sent by the client (bottom)

Once the communication is started, ARCOS sends periodically every 100 ms a special packet containing the robot status, the *Server Information Packet* (SIP). This packet is sent without waiting for the client to request it. The client program can read data sent from the robot, and send commands back to it asynchronously. The robot can send different types of information, each one of them has a specific data structure, but the packet structure remains always the same for the Header, Byte Count and Type bytes. Fig. 6 depicts the typical structure of ARCOS commands.

On the other hand, client commands have a different structure, but as can be seen in Fig. 6, it is almost always the same, changing only the arguments that depend on the chosen command. ARCOS commands can contain four different types of argument, namely none (the argument is empty), integer (16 bits signed integer), unsigned (16 bits unsigned integer), or string (array of characters with variable length). A brief description of some of the most frequently used ARCOS commands is shown in Table 1.

**Table 1.** Some of the most frequently used ARCOS=commands

| **Command** | **Number** | **Data** | **Description** |
|---|---|---|---|
| PULSE/SYNC0 | 0 | — | Keep the watchdog alive/ First synchrony packet |
| OPEN/SYNC1 | 1 | — | Begin communication with the robot/Second synchrony packet |
| CLOSE/SYNC2 | 2 | — | Finish communication with the robot/Third synchrony packet |
| ENABLE | 4 | Int | Enable or disable the motors |
| SONAR | 28 | Int | Enable or disable all the sonars, or some specific array of sonars |
| CONFIG | 18 | — | Request a configuration SIP |
| MOVE | 8 | Int | Translate X mm forward(+) or backward(-) |
| ROTATE | 9 | Int | Rotate X degrees counter-clockwise(+) or clockwise(-) |
| VEL | 11 | Int | Translate at a velocity X (mm/sec) forward(+) or backwards(-) |

The client receives the SIP commands from ARCOS and stores the data in a structured way, allowing other parts of the program to access it. The client also takes care of sending to the robot the commands received from the higher layers, as well as keeping communications alive, by sending the ARCOS watchdog PULSE command every 100 ms.

## 3.1 Implementation of the Protected Objects

Figure 7 shows the package structure in which the code for communicating with the real and the simulated robot has been organized into. As can be seen, there is a parent package that contains the definition of the common synchronized interface (`I_Robot`), the common data types to be used, and the functions that generate the byte arrays corresponding to the ARCOS command to be sent (*Output_Gen* set of functions). Both protected types are implemented in separate child packages. They store the last update of the robot status and the communication configuration parameters. Finally, we have defined two private child packages that contain the data types that are used internally, and the supporting subprograms shared by both protected types, such as CRC calculation. It is worth describing some key details:

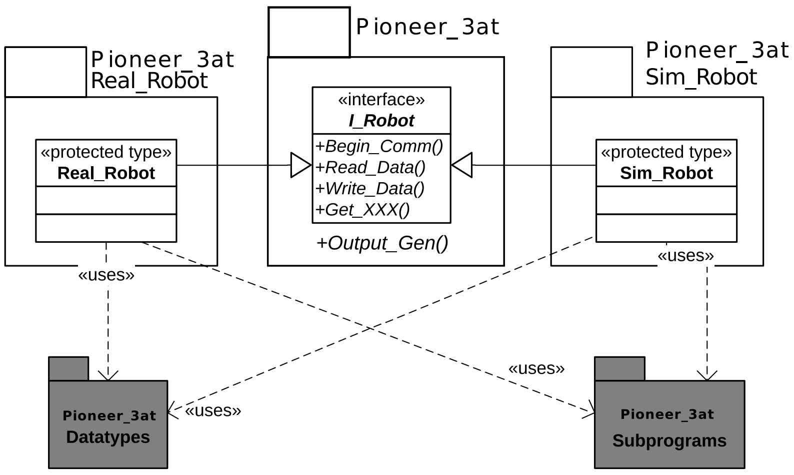


**Fig. 7.** Package diagram showing the structure of the developed code

- The *Begin_Comm* procedure must be called with the adequate configuration parameter, depending on whether the real robot or the simulator is going to be used. As said before, it is important to remember that the simulator can simulate one or more robots, which are available through different network ports. Thus, the configuration parameter (*Rt_Robot_Comm_Config*) must be set accordingly.
- The *Read_Data* procedure reads the last SIP command sent by the robot and updates the inner variables stored in the protected object. Then, it sends the watchdog command back to the robot in order to keep it alive, every 100 ms (the time that takes for ARCOS to send a new SIP). These variables can then be read by a set of *Get_XXX* functions, for instance *Get_Pos, Get_Vel, Get_Sonars*, etc.
- The *Output_Gen* functions receive a command number and optionally a set of arguments, and generate the adequate ARCOS commands. These functions only return the desired ARCOS command. It must then be sent to the robot by using the *Write_Data* procedure.

Lastly, this structure allows for a smooth transition between the code developed for the simulator and for the real robot, since it is only necessary to change the type of the protected object being used, as shown in Fig. 8. Thanks to the dispatching facilities obtained by the use of Ada interfaces, subprograms calls need not be modified.

```
  -- Using the simulator
  Robot : Pioneer_3at.I_Robot'Class := new Pioneer_3at.Sim_Robot;
  Robot_Config : Pioneer_3at.Rt_Robot_Comm_Config (Socket);

  -- Using the real robot
  Robot : Pioneer_3at.I_Robot'Class := new Pioneer_3at.Real_Robot;
  Robot_Config : Pioneer_3at.Rt_Robot_Comm_Config (Serial);

  -- Subprograms calls need not to be modified:
  Robot.Begin_Comm (Robot_Config);
  Robot.Write_Data (Pioneer_3at.Cmd'(Watchdog));
```

**Fig. 8.** Changing between the simulated and the real robot

## 4 Using the Developed Software: Templates for Students

The structure of the code presented before lends itself to different experiments with various levels of complexity for undergraduate, graduate or master students. Grade students can avoid the complexity added by concurrency and develop a simple single-tasking application that implements a *Sense-Plan-Act* control loop. At this level, students only

```
with Pioneer_3at; use Pioneer_3at; with Pioneer_3at.Sim_Robot;
use Pioneer_3at.Sim_Robot; with Ada.Real_Time; use Ada.Real_Time;

procedure Test_Simulator is
  Next_Time : Time := Clock;
  Period : constant Time_Span := Milliseconds (100);
  Robot : Pioneer_3at.I_Robot'Class := new Pioneer_3at.Sim_Robot;
  Robot_Config : Pioneer_3at.Rt_Robot_Comm_Config (Socket);
           -- we are going to use the simulator
begin
  -- Configuration
  Robot.Begin_Comm (Robot_Config);
  Robot.Write_Data (Pioneer_3at.Cmd'(Config));
  Robot.Write_Data (Pioneer_3at.Cmd'(Motors_On));
  Robot.Write_Data (Pioneer_3at.Cmd'(Sonars_On));
  -- Sense-Plan-Act loop
  for I in 1 .. 1000 loop
    Robot.Read_Data; -- 'Sense'
    -- 'Plan' and 'Act' code here!
    -- Pos_X := Robot.Get_Pos_X;
    -- Sonars := Robot.Get_Sonars;
    -- Robot.Write_Data (Pioneer_3at.Output_Gen(Cmd'(Motors), 20, 20));
    -- etc.
    Robot.Write_Data (Pioneer_3at.Cmd'(Watchdog));
    Next_Time := Next_Time + Period;
    delay until Next_Time;
  end loop;
end Test_Simulator;
```

**Fig. 9.** Excerpt of the code template given to undergraduate students

have to fill the “Plan” part of the control loop with the code that solves the proposed problem. We provide a sample template code (see Fig. 9) that students must reuse and complete.

We propose simple problems, since students do not have enough programming experience, like:

– Make the robot accelerate as time passes.
– Make the robot draw circles.
– Make the robot follow a path generated from its initial position.
– Make the robot wander in a big map without colliding with any object.

Regarding postgraduate and master students, they are presented two different kind of problems, depending on their profile. On the one hand, they have to implement a concurrent solution for the problem of controlling the simulated robot, since writing and reading commands can be sent to the robot, at most, every 100 ms. Thus, they have to design an application with two or more tasks, and one or more protected objects to store the robot’s status data in order to avoid polling the simulator. Besides, the MobileSim simulator is capable of simulating more than just one robot. This makes it possible to develop harder problems, where students must design an application where two or more robots must be controlled and must cooperate in order to fulfil a given mission. The usual structure of such programs is shown in Fig. 10.

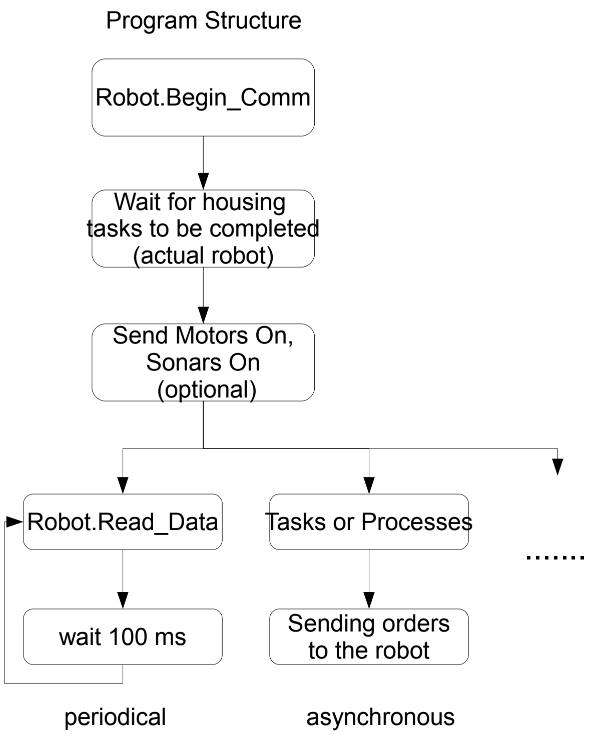

Program Structure
Robot.Begin_Comm
Wait for housing tasks to be completed (actual robot)
Send Motors On, Sonars On (optional)
Robot.Read_Data
wait 100 ms
Tasks or Processes
Sending orders to the robot
.......
periodical
asynchronous

**Fig. 10.** Flow chart describing the kind of applications postgraduate students must develop

On the other hand, master students must also solve harder problems regarding robot navigation, path planning, mapping, etc. These are mainly algorithmic problems and we are not going to describe them in this paper, but the interested reader can find more information in [19].

## 5 Conclusions and Future Research

This paper has described ongoing work that aims at developing a common Ada infrastructure that can be used in both our teaching and research duties. This infrastructure revolves around the development of an Ada interface to a real Pioneer P3–AT robot and its simulator. We hope the use of such platform will make learning programming languages more appealing to undergraduate students, while at the same time it allows us to prepare more difficult problems to postgraduate and master students, involving concurrency control and robot cooperation in complex missions.

The Ada interface has been programmed by using the synchronized interface facility provided by Ada and then implementing to protected objects: one for interfacing with the real robot and another for the simulator. We found variant records very useful for the configuration procedure, since the protected type representing the real robot and the one representing the simulator have different configuration parameters (serial port number and serial port parameters versus IP and port number), and we were able to define a single procedure to perform the configuration using a variant record instead of adding two configuration subprograms. However, we experienced some troubles using the library distributed with GNAT for the control of the serial port in Linux.

Regarding future research, we plan to develop a thin network layer in order to make it possible to use any programming language to interact with the Ada interface with the MobileSim. We will then be able then to extend this to other courses where we teach different programming languages.

## References

1. The Bologna Process web page, `http://ec.europa.eu/education/higher-education/doc1290_en.htm` (checked February 2011)
2. Benlloch-Dualde, J.V., Blanc-Clavero, S.: Adapting Teaching and Assessment Strategies to Enhance Competence-Based Learning in the Framework of The European Convergence Process. In: Proceedings of the 37th ASEE/IEEE Frontiers in Education Conference, Milwaukee, USA (October 2007)
3. Markham, T., Larmer, J., Ravitz, J.: Project Based Learning. A Guide to Standards-Focused Project Based Learning for Middle and High School Teachers, Buck Institute (2003)
4. Boud, D., Dunn, J., Hegarty-Hazel, E.: Teaching in Laboratories. Society for Research into Higher Education, Guildford (1986)
5. Hassan, H., Domnguez, C., Martnez, J.M., Perles, A., Albaladejo, J.: Remote Laboratory Architecture for the Validation of Industrial Control Applications. IEEE Transactions on Industrial Electronics 54(6), 3094–3102 (2007)
6. Kramer, J., Scheutz, M.: Development environments for autonomous mobile robots: A survey. Autonomous Robots 22(2), 101–132 (2007)

7. Web page of the School of Industrial Engineering of the Universidad Politcnica de Cartagena, Bachelor in Industrial Electronics and Automation Engineering, `http://www.etsii.upct.es/giti_en.htm` (checked November 2011)
8. Web page of the Master in Information and Communication Technologies of the Universidad Politcnica de Cartagena (in Spanish), `http://www.dte.upct.es/doctorado` (checked November 2010)
9. Iborra, A., Alonso, D., Ortiz, F.J., Franco, J.A., Snchez, P., Álvarez, B.: Design of service robots. IEEE Robotics and Automation Magazine, Special Issue on Software Engineering for Robotics 16(1), 24–33 (2009)
10. Ortiz, F.J., Alonso, D., Álvarez, B., Pastor, J.A.: A reference control architecture for service robots implemented on a climbing vehicle. In: Vardanega, T., Wellings, A.J. (eds.) Ada-Europe 2005. LNCS, vol. 3555, pp. 13–24. Springer, Heidelberg (2005)
11. Pioneer 3 operations manual web page, `http://robots.mobilerobots.com/docs/alldocs/P3OpMan6.pdf` (checked November 2010)
12. Web page of the MobileRobots/ActivMedia MobileSim simulator, `http://robots.mobilerobots.com/wiki/MobileSim` (checked November 2010)
13. Chil, R.: Desarrollo de un protocolo de comunicacin en tiempo real, usando el lenguaje Ada, para comunicarse con el robot Pioner P3–AT, Master Thesis (in Spanish), Universidad Politcnica de Cartagena (2010)
14. Web page of the MobileRobots company, `http://www.mobilerobots.com` (checked November 2010)
15. Mohamed, N., Al-Jaroodi, J., Jawhar, I.: Middleware for Robotics: A Survey. In: Proceedings of the 2008 IEEE Conference on Robotics, Automation and Mechatronics, Chengdu, China, pp. 736–742 (September 2008)
16. Web page of the Robot Standards and Reference Architectures (RoSTa), Coordination Action funded under EU's FP6, `http://wiki.robot-standards.org/index.php/Current_Middleware_Approaches_and_Paradigms`
17. Fayad, M., Schmidt, D.: Object-Oriented Application Frameworks. Special Issue on Object-Oriented Application Frameworks, Comm. of the ACM 40(10), 32–38 (1997)
18. Mosteo, A., Montano, L.: SANCTA: an Ada 2005 general-purpose architecture for mobile robotics research. In: Abdennahder, N., Kordon, F. (eds.) Ada-Europe 2007. LNCS, vol. 4498, pp. 221–234. Springer, Heidelberg (2007)
19. Murphy, R.: Introduction to AI robotics. The MIT press, Cambridge (2000) ISBN 0-262-13383-0