Robotics, Computer Science curricula and Interdisciplinary activities

J. Arlegui[1] , E. Menegatti[2], M. Moro[2], A. Pina[1],

[1] Public University of Navarra, Dept. of Psychology and Pedagogy & Math and Computer Engineering
31006 Pamplona, Spain,
{arleguip,pina}@unavarra.es
[2] Univ. of Padova, Dept. of information Engineering
35131 Padova, Italy
{emg, mike}@dei.unipd.it

Abstract. In this paper, we present four examples of how to use robotics to foster student learning of complex Computer Science concepts. We propose to use Robotics not as a subject on its own, but as a tool for teaching/learning purposes. Following the examples presented in this paper, we discuss several ideas about Computer Science curricula, inter-disciplinary activities and teaching-learning methodologies.

Keywords: Robotics, Constructivism, Computer and Information Science Education, NXT.

1 Introduction

The 2005 ACM Computing curricula report [6] presents a reasoned guide to the topics in the different kinds of computer science degree programs they are proposing. Among the computing and non-computing topics, we find that learning these topics could be reinforced by the use of Robotics as a learning tool (especially for Computer Architecture and Organization, Software design & development, Mathematical foundations and Interpersonal communication). In this work, we are focusing on Robotics not as the field of study, but as a tool to teach other subjects in a computer science curricula (or more in general, in scientific curricula). From the point of view of innovation in the computer Science Curricula, Denning and McGettrick point out that "The first challenge is to embed the foundational practices of innovation into the curriculum, so that students learn innovation by doing…..The intention is that innovation should become an essential aspect of their attitude of mind…."[7]. The curricula in computer science (an other disciplines) should innovate, using for example "learning by doing" formula and should re-think the Theory-Practice equilibrium during the 3/4/5 years study of a degree (and maybe offer topics like Robotics in the first year, with a "Learning by Doing" approach). To carry out this kind of curricula innovation, we need to deeply revise methodological issues [8][9]. From our didactical experience, we see that an adequate educational use of robots in

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 10-21

computer science can promote a proactive learning and a cooperating work by groups (defining the right group problems to be solved and leaving the groups to evolve themselves); and this has to deal with methodological issues. Even if these aspects are not discussed in this paper, the authors are working under a theoretical constructivism/connectionism background and with enquiry based or group project based approaches. This is the approach followed in the TERECoP project (Teacher Education on Robotics-Enhanced Constructivist Pedagogical Methods, http://www.terecop.eu/). The main goal of the project is to develop a framework for secondary-level teacher education courses in order to enable teachers to implement a robotics-enhanced constructivist learning in school classrooms [1][2][10]. At the same time, the authors of this work at the University of Padua have a long-term experience in RoboCup, one of the most important robotics competitions [http://www.robocup.org/]. The University of Padua since 1997 has a RoboCup team composed of master students and organizes competitions like the Seventh RoboCup International Competitions and Symposium in 2003 (Padua, Italy [3]). The activity of coordinating and guiding several teams of students in building and programming the autonomous soccer robots gave us the possibility to understand how a practical realization of a robot can contribute to stimulate the students' interest and skills in ICT related technologies (and other non computing abilities). Two examples of robotic projects not related to soccer but realized by students previously involved in our RoboCup team can be found in [4] and [5]. In this paper we present four examples of possible implementations of interdisciplinary activities for Computer Science curricula using robotics as a tool.

The paper organization is as follows. In Sec. 2 we describe the robotic platform we selected for implementing the proposed didactical experiences. In Sec. 3, 4, 5 & 6 we present examples in the typical computer science field (thread synchronization and multitasking. analytical vs numerical approaches applied to the robot self-location problems, sorting problems, and a simple implementation of the Turing machine). At the end some conclusions and reflections are outlined.

2 Needed/Wanted features for the robotic platform

We considered different robotic platforms that could fulfill some requirements like to allow programming with different paradigms & levels, to offer many degrees of complexity (to be able to be used in pre-university levels) or to remain simple but with significant possibilities of expansion . Our final choice was the NXT LEGO technology, because it fullfills the previous requirements and moreover it is possible to start working with it almost immediately (no electrical or other hardware or software arrangements are necessary). Another advantage of the NXT LEGO technology we are interested in is the different programming languages and programming environments available. For instance, with the NXT LEGO is possible to use the original LEGO graphical programming environment NXT-G, or the C-like NXC or the Java based LeJOS-NXJ. Moreover, one has the possibility to use several operating systems and/or platforms (URBI, Universal Real-time Behavior Interface,

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 10-21

for Windows, Mac OS X, Linux or NXT-Symbian running on Symbian 6.0 Java-enabled mobile phones).

3 An example in Synchronization & Multitasking (Operating Systems topic)

3.1 Objectives

Multitasking and Synchronization are fundamental concepts in courses like: Operating Systems, Advanced/Concurrent Programming, Real-time Programming. A deep comprehension of the reasons of introducing multitasking can be achieved only running simulated or real examples of simultaneous tasks, particularly when they show interferences and synchroniza-tion/communication needs. Robotics can provide a real environment where the need of multitasking is easily shown by means of simple multi-behavioral examples.

3.2 Carrying out the experience

The NXT robot is constructed as a basic "tribot", a cart with two independent driven wheels and a caster wheel on the rear. This enables to turn left or right applying different powers to the two motors. A third motor moves up and down an arm: this action is independent from the turning motion. Three sensors are connected: one light sensor directed to the ground, one sound sensor, and one touch sensor enabling the user to provide an asynchronous signal.

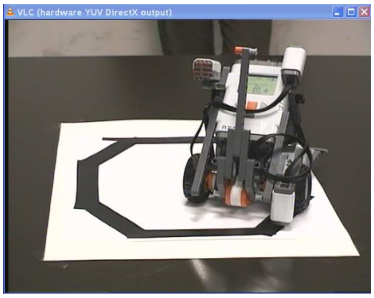
Three robot behaviors are programmed into the robot. The first behavior is the so called line follower: the robot follows the edge of a thick black line by swinging left to right and vice-versa depending on the reading of the brightness sensor. (figure 1). The robot follows clockwise the internal edge of the line by turning left when the brightness is over a certain threshold, and by turning to the right when the reading is under the threshold. For this first behavior, the controlling program is a infinite loop with a switch, based on the light sensor, between the two described motion commands.

The second behavior is to lower the robot arm for a certain number of seconds when a loud sound (e.g. a beat of hands) is detected. Also this behavior can be implemented with an infinite loop. The code of these two loops on their own is straightforward and not particularly significant, therefore we do not present it in detail. However, if one wants to activate both behaviors at the same time, a simple solution could be to insert the body of the two loops above in sequence as the body of a single loop.

```
LoopUntil(FOREVER) { // "Sequential" solution
    if (LightSensor(IN_PORT_3) > THRESHOLDLIGHT)
    { "turn left" } else { "turn right" }
```

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 10-21

```
      if (SoundSensor(IN_PORT_2) > THRESHOLDSOUND)
      { “arm down” “wait” “arm up” }}
```



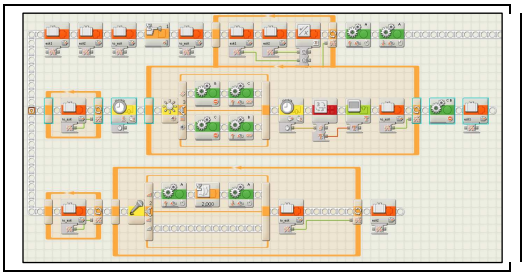


Figure 1 & 2. Scenario for the first example & defining 3 threads in NXT-G

Running the program, the robot shows very effectively the non controlled interaction between the two behaviors that arises. If the “wait” before the “arm up” command is of several seconds the robot will not turn left when crossing the black line (because the processing of the brightness sensor is delayed) and the robot will exit from the circuit stripe, failing its main behavior of line following.

This negative interference can be avoided allowing two different tasks to control separately the two behaviors, provided that some form of time sliced scheduling is implemented in the run time environment, as in the case of NXT. Next code allows to verify a correct multitasking behavior for the robot (the scheduler actually maintains active both tasks).

```
  // “Multitasking” solution
  Task followLine() {
    LoopUntil(FOREVER) {
       “Follow line code”
   }
  Task Arm() {
    LoopUntil(FOREVER) {
       “arm down up code”
   } }
```

Now, think to add a third behavior to stop the robot when the touch sensor is pressed. This lead to the need of a synchronized solution where the two controlling loops are exited in specific points as soon as possible after the touch sensor has been pressed (using common synchronization variables, see code & figure 2).

```
  Task followLine() {// with synchronization
    LoopUntil(LOGIC, Var(toExit, READ), FALSE){}
      // wait initial synchronization
    LoopUntil(LOGIC, Var(toExit, READ), TRUE)
      {“Follow line code”}
      // main loop exited when the variable is true
    Move (OUT_PORT_BC, STOP, BRAKE);
      // stop definitely
    Var(exit1, WRITE, TRUE);  // ended signal
  }
  Task Arm() {
    LoopUntil(LOGIC, Var(toExit, READ), FALSE){}
      // wait initial synchronization
    LoopUntil(LOGIC, Var(toExit, READ), TRUE)
```

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 10-21

```
      {"arm down up code"}
      // main loop exited when the variable is true
    Var(exit2, WRITE, TRUE);  // ended signal
  }
  Task StopRobot() {
    Var(exit1, WRITE, FALSE);  // init variables
    Var(exit2, WRITE, FALSE);
    Var(toExit, WRITE, FALSE);
    WaitUntil(TOUCHSENSOR, IN_PORT_1, PRESSED);
      // wait for touch sensor pressed
    LoopUntil(LOGIC,
    And(Var(exit1, READ), Var(exit2, READ)), TRUE){}
      // wait for two tasks completion
    Move (OUT_PORT_A, FORWARD, 30, DEG, 50, BRAKE);
    Move (OUT_PORT_A, BACK, 30, DEG, 50, BRAKE);
  // a final event, the arm moves up/down for 50 degrees}
```

3.3 Analysing the results

The usefulness of both multitasking and synchronization is made evident with simple robotic experiments that manifest concurrency problems, when present, in quite natural manner. We used these examples during 3rd year Computer Science Engineering degree in Operating Systems topic.

4 Analytical vs Numerical solution of a self-positionning problem

4.1 Objectives

A common problem in robotics is to permit the robot to calculate its current position with respect to a given 2D Cartesian reference using its sensors' data. Powerful robots can perform this calculation with sufficient precision thanks to complex sensors like cameras, lasers or sonars and some landmarks. In NXT the only basic sensor giving a sufficient degree of precision is the sonar sensor able to return its distance from an obstacle within a reasonable range (less than 2.5 m) with a precision of +/- 3 cm.

If the robot knows its distance, namely d1 and d2, from two obstacles, it can be easily shown that the position of the robot is given by one of the two the intersection of the two circumferences centered in each one of the two obstacles and with radius r1 and r2 respectively equal to d1 and d2. This analytical solution may be problematic in case of NXT because its run-time allows only integer calculation. This suggests to examine a different approach that calculate the position through subsequent approximations.

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 10-21

4.2 Carrying out the experience

The setup of the experiment includes a tribot with the sonar sensor mounted on the third motor making possible an horizontal exploration, two narrow obstacles put on known positions in front of the robot and a target point (figure 3). Assuming that all other objects (or walls) within the angle of observation are more distant than a minimum, the obstacles are identifiable when the sensor gives distances significantly less than that minimum or simply they are the closest objects in the surrounding world.

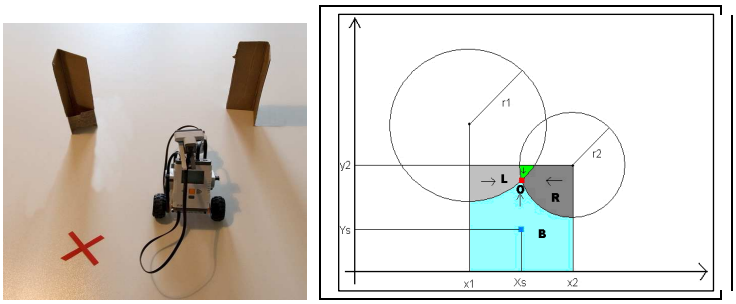


Figure 3 & 4. Scenario for the second example & its "geometric" solution

Given (x_1, y_1), (x_2, y_2) and (X_r, Y_r) respectively the coordinates of the two obstacles and the unknown coordinates of the robot, and r_1 and r_2 the two distances returned from the sonar sensor, the analytical solution is given by:

$$\begin{cases} (x - x_1)^2 + (y - y_1)^2 - r_1^2 = 0 \\ (x - x_2)^2 + (y - y_2)^2 - r_2^2 = 0 \end{cases}$$

To simplify the calculation, one of the two equations can be substituted by their difference:

$$Ax + By + C = 0$$

$$\begin{cases} A = -2x_1 + 2x_2 \\ B = -2y_1 + 2y_2 \\ C = x_1^2 - x_2^2 + y_1^2 - y_2^2 - r_1^2 + r_2^2 \end{cases}$$

which is the equation of the so called radical axis, the set of all points equidistant from the two obstacles. We must then calculate the solutions:

$$y^2\left(\frac{B^2}{A^2} + 1\right) + y\left(2\frac{BC}{A^2} + 2x_1\frac{B}{A} - 2y_1\right) +$$

$$+\frac{C^2}{A^2} + 2x_1\frac{C}{A} + x_1^2 + y_1^2 - r_1^2 = 0$$

Knowing *a priori* that Y_r is less than $\min(y_1, y_2)$, this allows to choose the correct solution between the two ones calculated from the previous equation X_r is then obtained from Y_r and the axis equation.

The second method starts with a first approximation of the solution given by:

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 10-21

$$\begin{cases} Xs = \dfrac{x1 + x2}{2} \\ Ys = \dfrac{\min(y_1, y_2)}{2} \end{cases}$$

The area of interest is divided into four convergence areas denoted in the figure with the letters L (left), O (over), R (right), B (below) that recall the relative position of the approximation (later on called AP) with respect to the final solution (the intersection of the two circles, later on called SOL). The following rules are applied (say $d_1=d(AP, O_1)$ and $d_2=d(AP, O_2)$ the distance of AP from respectively obstacles 1 and 2):

- AP is in L if $d_1<r_1$ and $d_2>r_2 \Rightarrow$ increase X_s
- AP is in R if $d_1>r_1$ and $d_2<r_2 \Rightarrow$ decrease X_s
- AP is in O if $d_1<r_1$ and $d_2<r_2 \Rightarrow$ decrease Y_s
- AP is in B if $d_1>r_1$ and $d_2>r_2 \Rightarrow$ increase Y_s
- AP≡SOL if $d_1=r_1\pm\varepsilon_1$ and $d_2=r_2\pm\varepsilon_2$

When the calculated distance of AP from the two obstacles coincides to the corresponding radius, apart from a small resolution imprecision (given by ε_1 and ε_2), AP represents the final solution (implemented in NXC, *http://bricxcc.sourceforge.net/nbc*).

The first part of the program must detect the two obstacles and to measure their distance from the robot exploring the space with the sonar head.

Got the two distances in the r1 and r2 variables, the calculation of the analytical solution is straightforward even it presents some difficulties (no floating point computations, no sqrt function available, etc…). For these reasons the terminating condition is evaluated on the square of d_i and r_i previously calculated and avoiding the square root calculation. In fact it results (a similar relation stands also for d_2 and r_2):

$d_1=r_1\pm\varepsilon_1 \Rightarrow d_1^2=r_1^2+\varepsilon_1^2\pm2r_1\varepsilon_1 \Rightarrow d_1^2-r_1^2=\varepsilon_1^2\pm2r_1\varepsilon_1$

Considering the limitation of the sonar sensor, a value of 1 as the minimum for ε_1 (and ε_2) is reasonable: when such a value is approached, you obtain: $|d_1^2-r_1^2|=|1\pm2r_1|$

The 'numerical' solution appears a bit simpler and more understandable:

```
// calculate the square of the distances
r1=r1*r1;
r2=r2*r2;
// first approximation
xr=(x1+x2)/2;
if (y1<y2) yr=y1/2;
else yr=y2/2;
// loop to converge to the solution
do {
  // square of the approximated distances
  // from the two obstacles
  d1=(x1-xr)*(x1-xr)+(y1-yr)*(y1-yr);
  d2=(x2-xr)*(x2-xr)+(y2-yr)*(y2-yr);
  // update the approximation on the basis of
  // the area of proximity (see explanation above)
  if ((d1<r1) && (d2<r2)) yr=yr-1;
  else if ((d1>r1) && (d2>r2)) yr=yr+1;
  else  if ((d1<=r1) && (d2>=r2)) xr=xr+1;
```

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 10-21

```
    else  if ((d1>=r1) && (d2<=r2)) xr=xr-1;
    // evaluate the approximation
    if (d1>=r1)
      conf1=1+2*r1;
    else
      conf1=2*r1-1;
    if (d2>=r2)
      conf2=1+2*r2;
    else
      conf2=2*r2-1;
    d1=abs(d1-r1);
    d2=abs(d2-r2);
  }
  while ((d1>conf1) || (d2>conf2));
```

Given the calculated position in X_r and Y_r, the code to reach a target position requires to know the ratio between the angle performed by the motors connected to the wheels and the linear movement of the robot. The rest of the code presumes this knowledge and, apart this important detail, it is straightforward and not presented in detail. An implementation in NXT-G has been also done even though it gives a very large and not so easily understandable program.

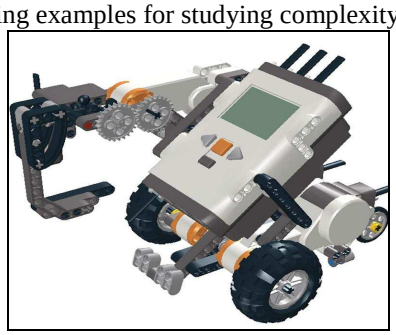
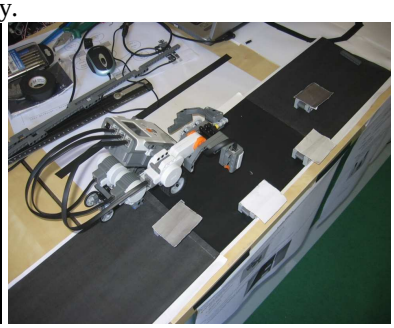
4.3 Analysing the results

NXT is enough powerful to support a rather difficult task like self-positioning, even with evident limitations. The analytical solution requires a knowledge about 2D analytical geometry which is common for an engineer student. The proposed solution shows the differences between the two approaches and makes the students appreciate the suitability of the numerical approach.

5 Sorting

5.1 Objectives

Apart from their practical applications, sorting algorithms are a wide class of interesting examples for studying complexity.

Figures 5 & 6. The special “tribot” used for selection sort & sorting 4 items

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 10-21

We chose two of them, selection and heap sort, as representative respectively of the O(n2) and O(n log n) subclasses, because of their relatively simple implementations with NXT. The detailed theory of these algorithms are out of the scope of this presentation and it can found in any book on fundamentals of data structures and algorithms (for instance in [11]). Moreover the heap sort NXT implementation is still under development, so we limit ourselves to the description of the selection sort implementation.

5.2 Carrying out the experience

For this example the robot is the usual tribot with two motorized wheels, plus a motorized rotating arm used to shift items laterally (fig. 5). Limiting ourselves to the standard sensors included in a kit, we decided to sort objects on their brightness, so we used a light sensor to measure the reflected light of gray colored paper labels glued on the items to be sorted.

One of the initial decisions was to select a physical characteristic we could use to provide values to be compared during the sorting. Limiting ourselves to the standard sensors included in a kit, we decided to use a light sensor to measure the reflected light of gray colored paper labels glued on the items to be sorted. The robot moves back and forth along one of the side of a black strip on which n items with different gray labels on the top are initially put on predefined positions along a straight line but in a random order. When the robot moves the light sensor, mounted on the robot on the same side of the rotating arm, can read the grey level of each label (fig. 6, with 4 items). The robot makes n passages: during each passage it reads all the n positions looking for the item with the lightest label. When found, it (possibly) comes back to it and activates the rotating arm to shift the item: this action corresponds to the 'selection'. Even if it is not shown, you can imagine that the shifted item drops down on a slide so that, one by one, the sorted items are enqueued in the decreasing order. The black strip has the lowest gray level and therefore the absence of an item previously shifted is recognizable.

5.3 Analysing the results

The more meaningful result of this experiment is the 'live' quadratic behaviour of the robot which makes actually n2 light readings to complete the task. This can be easily put in relation with the two nested cycles in the code.

```
int i, j, count, n, found, max, read;
task main()
{
  SetSensorLight(IN_1);
  n = 4;
  for (i = 1; i <= n; i++) {  // external cycle
    max = 0;
    found = 0;
    for (j=1; j <= n; j++) {  // internal cycle
      RotateMotor (OUT_BC, 40, 360);  // go forth
      read = Sensor(IN_1);
      if (read > max) {
```

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 10-21

```
          // new max
          found = j;
          max = read;
        }}
      // back to max
      RotateMotor (OUT_BC, 40, ((found-n)*360));
      RotateMotor (OUT_A, 40, 360);  // select item
      // back to start
      RotateMotor (OUT_BC, 40, -(found*360));
    }}
```

6 A Turing machine

6.1 Objectives

A Turing machine (TM) is a well known computer theory model to study function computability [12]. Formally is a model of computation controlled by a finite state machine equipped with a read/write head on a unbounded sequential tape: depending on the current state and the symbol read on the tape, the machine can change its state, write a new symbol onto the tape, and move the head to the left or right. When for each couple (state, symbol) the specified action is unique, the machine is deterministic (DTM), non-deterministic (NTM) otherwise; due to the theoretical proof of equivalence between a DTM and a NTM, in the following we talk simply to TM referring to DTM. In the proposed experiment, we implemented a didactical TM (with one-direction tape, an alphabet of 2 symbols and 2 possible states) performing integer additions with operands encoded with short bit streams. In our case the necessary limitations are represented by a binary alphabet and a tape with a limited number of slots.

6.2 Carrying out the experience

The read/write head of the simulated TM is a car able to shift LEGO blocks: some blocks are put on predefined positions that represent the limited number of slots of the simulated tape. Each block can be shifted on one of two positions which represent the binary value assigned to the slot; the current position is 'read' using the sonar sensor (fig. 7 and 8).

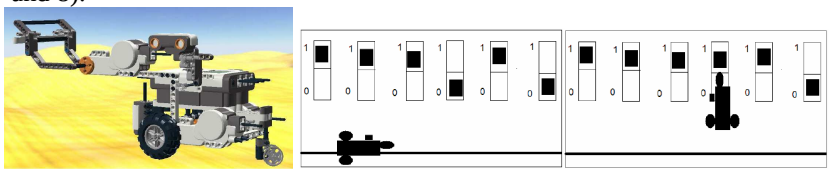


Figure 7 & 8. The Turing car & the car moving and "writing"

The problem to solve in this experiment with the TM is to perform an add function on integer values. A value i is represented by a sequence of i bit 1, whereas a sum

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 10-21

expression of two values is the concatenation of the two coded value separated by one 0. For instance: 111=3, 11111=5, 111011=3+2.

The rules the TM must apply are summarized in the following table; the initial state is 0 and the slots contain the expression sequence to be evaluated. The number of necessary slots can be estimated in reason of the input expression and the sum value, padding to the right with zeros, at least one terminating zero, the initial sequence if shorter than such a number.

Current state	Input read symbol	Next state	Symbol to be written	Tape (i.e. car) direction
0	0	0	0	>
0	1	1	1	>
1	0	2	1	>
1	1	1	1	>
2	0	3	0	<
2	1	2	1	>
3	0	ERR	--	--
3	1	4	0	>
4	0	END	--	--
4	1	END	--	--

With 7 slots, an input 1110110 is elaborated as follows (underlined the slot under reading, in square brackets the state):

[0] 1110110 – [1] 1110110 – [1] 1110110 – [1] 1110110 – [2] 1111110 – [2] 1111110 – [2] 1111110 – [3] 1111110 – [4] 1111100

6.3 Analysing the results

TM is a very general computation model over which a teacher can deal with a large variety of interesting problems. Its simple definition and elegant power can be appreciated when you see the TM car simulating it. Our implementation can be easily modified to study and implement different resolving algorithms.

7 Conclusions

Using different approaches for programming the robot, it is possible to introduce in an easy way advanced programming skills and motivate the students to examine and exploit complex models and programming paradigms.

Topics and experiences presented in this paper were related to computer science at university levels, but robots as “learning tools" can be exploited by also teachers from different disciplines and from previous education levels, as demonstrated by other examples developed in the TERECoP project framework. Guiding examples must be used as suggestions to teachers to prepare their own experiences taking into account their specific didactical objectives, the initial competence of their classrooms, the

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 10-21

operative environment. In any case, it is important to use adequate methodologies, to coordinate/integrate the activities within the curricula and with the other colleagues. In the next months we have to deal with the practical and organization issues to apply these issues at high school and university levels.

Acknowledgments. This paper was partly based on work done in the frame of the project "Teacher Educations on Robotics-Enhanced Constructivist Pedagogical methods" (TERECoP) funded by The European programme Socrates/Comenius/Action 2.1, Agreement Nº 128959-CP-1-2006-GR-COMENIUS-C21 2006-2518/001-001 S02.

References

1. Alimisis D. et al, 2007. Robotics & Constructivism in Education: the TERECoP project. In Proceedings of the 11th European Logo Conference (http://www.eurologo2007.org/, 19 - 24 August, Bratislava, Slovakia)
2. Arlegui J. et al, 2007. Los entornos LEGO y LOGO en robótica educativa. In Proceedings of EDUTEC 2007 (Latin American conference on Education & Tecnology, http://www.utn.edu.ar/edutec200), October 2007, Buenos Aires (in Spanish)
3. Enrico Pagello, Emanuele Menegatti, Daniel Polani, Ansgar Bredenfel, Paulo Costa, Thomas Christaller, Adam Jacoff, Martin Riedmiller, Alessandro Saffiotti, Elizabeth Sklar, Takashi Tomoichi RoboCup-2003: New Scientific and Technical Advances AI Magazine, American Association for Artificial Intelligence (AAAI) Volume 25, Num. 2, Summer 2004, pp. 81-98
4. E. Menegatti, A. Pretto, A. Scarpa, A. Tellatin, S. Tonello, A. Lastra, A. Guatti An interactive robotic sculpture EUROBOT 2006 Workshop on Educational Robotics, Acireale (ITALY), June 2006
5. A. Lastra, P. Alberti, E. Menegatti Experimentations on advanced robotics for academic education EUROBOT 2006 Workshop on Educational Robotics, Acireale (ITALY), June 2006
6. Computing Curricula: 2005 Overview Report (The Joint Task Force for Computing Curricula 2005, http://www.acm.org/education/curric_vols/CC2005-March06Final.pdf), ACM, AIS & IEEE-CS.
7. Peter J. Denning and Andrew McGettrick. Recentering Computer Science. Communications of the ACM, November 2005/Vol. 48, No. 11.
8. Harel, I. and Papert S., Constructionism, Learning by Design, and Project-based Learning, 2001. In M. Orey (Ed.), Emerging perspectives on learning, teaching, and technology. Available Website: http://www.coe.uga.edu/epltt/LearningbyDesign.htm
9. Loidl S. et a : Preparatory Knowledge: Propaedeutic in Informatics, in Proceeding of ISSEP 2005: pp. 104-115, Klagenfurt, Austria.
10. Alimisis P.: Designing Robotics-Enhanced Constructivist Training for Science and Technology Teachers: the TERECoP Project, Proceedings of ED-MEDIA 2008-World Conference on Educational Multimedia, Hypermedia & Telecommunications, p. 288-293, Vienna, Austria.
11. Goodrich M. T., Tamassia R., Data Structures and Algorithms in Java, 4th edition, Wiley & Sons, 2006.
12. J.E. Hopcroft, R. Motwani, J.D. Ullman, Introduction to Automata Theory, Languages and Computation, 3d Ed., Addison-Wesley, 2007.