20

Teaching Introductory Programming Concepts with Lego MindStorms in Greek High Schools: A Two-Year Experience

Maya Sartatzemi[1], Vassilios Dagdilelis[2] and Katerina Kagani[1]
University of Macedonia
[1]Department of Applied Informatics,
[2]Department of Educational and Social Policy
Thessaloniki,
Greece

1. Introduction

Even today, teaching the basic principles of programming still presents numerous difficulties for students, which in the last several decades have comprised the subject of extensive research (Mayer, 1989; Mendelsohn et al., 1990; Du Boulay, 1989; Du Boulay et al. 1989; Brusilovsky et al., 1994). On the one hand attempts have been made to investigate the factors that render the learning of programming principles difficult, and on the other, to examine the effectiveness of programming languages and integrated environments, as well as the actual methods applied to the teaching of programming.
The difficulties, according to (Du Boulay, 1989; Du Boulay et al. 1989) fall into one of the following categories:

1. Orientation: what programming is; the types of problems that can be solved; the positive effects of learning programming.
2. Notional machine: difficulties in understanding the general properties of the "machine" which the student learns to control and its relation to the physical machine.
3. Rules of the programming language (notation): syntax and semantics as an extension of the properties and behaviour of the notional machine.
4. Learning the established structures.
5. Pragmatics: acquiring the ability to determine, develop, control and debug a program with the available tools.

Many researchers agree with the view that one of the most significant factors which contributes or even gives rise to the above-mentioned difficulties is the use of the classic approach (Brusilovsky et al., 1997; Xinogalos, 2002) to the teaching of programming. By this we mean teaching based on:

- The use of a general purpose language (such as Pascal, C, etc);
- The use of a professional environment for this language; and
- The proposed problems that require the processing of numbers and/or symbols.

Much research has shown that the problems arising from general-purpose languages are due to the fact that they are complex and novices need to learn a lot in order to be able to

perform relatively simple tasks (Brusilovsky et al., 1994; Brusilovsky et al., 1997). Many such languages require the user to key in a very "dark" code in order to achieve insignificant and commonplace outcomes. Frequently, the students are either taught the syntax before writing their first program, which can be rather discouraging, or they develop programs without having any interest in them whatsoever.

In order to overcome these obstacles, various approaches have been proposed. One application, which appears to be particularly popular with students in secondary level education, is based on the use of computer-controlled models, such as the Logo turtle or Karel's Robot (Pattis et al., 1995; Bergin et al., 1997; Becker, 2001). Besides these simulated models, an attempt has lately been made to apply physical models in the teaching of programming. Generally speaking, the use of models, and especially robots, seems to exert an irresistible allure on young aged students. Many educators believe that robots are a powerful way to motivate learning (Hirts et al., 2003) thus they are used to increase student interest, interaction and retention.

The teaching of programming with the application of robots and appropriate environments has numerous benefits, as many researchers point out (Kaskalis et al, 2001; Hirst et al, 2003; Lawhead & Bland, 2003;). More specifically, the advantages of this approach are:

- It is concrete since students program things they can handle, to behave in ways they can observe in the physical world.
- It is incremental.
- It is creative.
- It allows manipulation within a constrained context.
- It provides immediate feedback.
- Children's constructions have behaviour (i.e. anthropomorphic behaviour which is very interesting.
- It uses a variety of skills.
- It allows complete novices to create interesting outcomes (e.g. "go collect a tennis ball" rather than "Hello world").

The above view appears to be internationally widespread, which also corresponds to the accepted perception in the Greek educational system. Of course, a successful (or otherwise) introduction to programming as a subject in any educational system cannot but be influenced by local factors. In the Greek educational system, for instance, and more specifically at secondary level, teaching programming is quite restricted time-wise: both third year junior and first year senior high school students are taught the basic concepts of programming in 10 one-hourly lessons. Within this short time period they are taught the role of a programming language, the basic commands for input/output data, the selection commands, as well as learning to use a programming environment in order to develop, edit, debug and run their programs. Usually, the teachers apply a general-purpose language and unavoidably the students meet with difficulties in their attempt to understand the program's behaviour and associate it to their activities. So far, the results of these endevours have not been particularly promising, which appears to be the case in other countries as well.

In this research we present the findings of the studies conducted in the last two years on teaching the basic concepts of programming to third year junior and first year senior high school students with the application of robots, and more specifically LEGO MINDSTORMS (LM). LM have been available on the market since 1996 through the LEGO Company and

enable the construction of a large range of physical models, which can be programmed with the help of a RCX processor integrated into them.
Our research team conducted 3 series of lessons to junior and senior high school students. The results from the 1st series of lessons were presented in a previous paper (Satratzemi et al., 2005). Our latest findings are included in this chapter, whose structure is as follows: In section 2 we give a description of LM, and the programming languages and environments used. Then, we give a brief description of the lessons where LM are used worldwide. In the 3rd section an analytical description is given of the lessons taught by our research team to the two different groups of junior and senior high school students. Following this, the comparative results of the students' assignments of both school years are presented. Lastly, the chapter ends with a discussion of the findings.

2. Description of Lego Mindstorms, programming languages and environments

The first products marketed by LEGO consisted of simple small bricks, which children assembled in order to build houses, castles, towers, bridges and so on. In this way they first came into contact with the basic ideas of architecture and civil engineering, in the same way that the renowned "Meccano" first brought young people into contact with mechanical engineering. Later, the LEGO kits were enriched with the addition of new pieces such as gears, axes, motors and batteries, with whose help the children were able to create mechanical constructions, like automobiles, lorries, cranes and ferris wheels. Within the framework of these constructions the children were gradually initiated in the science of engineering. The first-generation kits, therefore, allowed children to form structures, while the second-generation kits enabled them to create mechanisms.
In the last few years, new LEGO kits have appeared, LM, third-generation kits, which permit children to produce "behaviours". These kits are used to make a wide range of machines, which present specific behaviours, such as a robot, a machine that moves in reaction to a person's or other machine's movements. Now by attaching sensors onto the electronic LEGO bricks, children can transform a simple door into a "talking" door that "welcomes" those who enter it; or a door which counts how many people enter each day (Resnick, 1993). LM embodies the constructionism learning theory described in the book Mindstorms: Children, Computers, and Powerful Ideas by Seymour Papert.
The first retail version of LM was released in 1998 and marketed commercially as the Robotics Invention System (RIS). The current version was released in 2006 as Lego Mindstorms NXT.
The original Mindstorms Robotics Invention System kit comprises a large range of bricks, sensors, motors and other components, which can be used to make physical models. These extra components are attached onto a larger LEGO brick, on which the RCX processor is integrated. With the appropriate programming of RCX, one can create a wide variety of LEGO robots.
The three-steps involved in the creation of an autonomous robot are as follows:

1. Construct the robot according to the instructions given in the kit or by using one's imagination.
2. Program development using the iconic programming environment, which accompanies the kit or another programming language.
3. Load the program onto the robot, using the Infrared transmitter.
4. Program execution.

In order for the robot to express a particular behaviour, it is usual for steps 2 to 4 to be repeated several times.
RCX (Robotic Control X) is an autonomous microcomputer integrated onto a LEGO brick, which can be programmed through a computer and allows the construction to interact with the environment autonomously, independently of the computer.
This processor has:

- Three input ports (1, 2, 3);
- Three output ports (A, B, C);
- A small one line, LCD screen;
- A microphone; and
- Four buttons.

At the input ports sensors are connected (touch, light, temperature), while at the output ports motors and lamps are connected. The buttons are to turn the RCX on/off, to choose the program that will be executed and to start/stop the execution.
Due to RCX's ability to control motors or lights and to collect data with the help of sensors, both adults and children can easily make creations that move, react and carry out various tasks.
Inside the RCX there is a microcontroller Hitachi H8 with 32K external RAM. Its function is to control three motors, three sensors and an infrared serial communication port. Also, in the interior of RCX there is a ROM 16K chip, where a program driver is stored which is executed as soon as RCX is activated (http://graphics.stanford.edu/~kekoa/rcx/).
A firmware program is also loaded which requires 16K storage space. Both these programs – driver and firmware –undertake the translation and execution of the programs loaded onto RCX by the user to control the robot (http://emhain.wit.ie/~p98ac25/).

2.1 Programming of RCX

In order to program RCX it is necessary to have two software on the computer (http://mapageweb.umontreal.ca/cousined/lego/4-RCX/Survey/):

- An operating system (OS): without this operational system RCX cannot carry out any activities.
- A series of communication processes (libraries): these are necessary only for the computer to be able to send information to or receive data from RCX. These libraries can be a set of .exe or .dll or .ocx (ActiveX) files. The libraries should assist the program to transmit/save/retrieve data to and from the microcomputer.

Despite the fact that the two elements mentioned above were intended for different platforms (the operating system for RCX, the libraries for the computer), they must be compatible so that communication between computer and RCX is possible (http://mapageweb.umontreal.ca/cousined/lego/4-RCX/Survey/).
The first time RCX is activated, the code stored in its memory (ROM) is executed. Following, the software "firmware" must be loaded. This concerns RCX's operational system, which is responsible for the management of the system's resources and the execution of the programs that are loaded. As soon as firmware is loaded onto RCX, the user can load their program. This software receives the programs from the infrared transmitter and responds to the buttons found on RCX, which are used to start or stop the program's execution. It facilitates communication between the motors and sensors (robot I/O), assists the basic types of data and operations (e.g., a math library), it supports tools in order to make interaction with the

robot feasible. There are, however, particular circumstances, where firmware is not enough since greater control of RCX is required, e.g. in the use of more variables. In order to deal with such situations, alternative type of software have been developed that can be used in LM RCX Controller; these are legOS, leJOS and PbFORTH.

Along with the LM kit the company also provides an Active-X called "Spirit.ocx". This operates as an interface between the program created by the user and RCX. It includes a series of predefined procedures and variables, which support communication with RCX. With the help, therefore, of this file one can create a form in Access, for instance, which is comprised of buttons that control the robot's movement. Moreover, theoretically one can program RCX using any programming language whatsoever (http://emhain.wit.ie/~p98ac25/).

The software which comes with LM to create the programs is relatively simple as it was mainly designed for children and for adults who do not have programming experience.

Program loading – execution:

Communication between the computer and RCX occurs with infrared rays. An infrared transmitter (IR- transmitter – tower) is connected to the serial port of the computer (http://emhain.wit.ie/~p98ac25/). With the help of IR, programs are loaded onto RCX. 6KB of memory are available for the storage of each program the user loads onto RCX (http://graphics.stanford.edu/~kekoa/rcx/). From the moment that a program is loaded onto RCX, the robot is ready to operate autonomously. The absence of wires facilitates the robot's free movement. The only thing the user needs to do is to press the correct button on RCX in order to start program execution. The execution can be repeated. In addition, there is the possibility to load many programs on the computer and each time to activate and execute whichever the user wishes.

Below are presented the programming languages and environments used in the lessons we conducted. More specifically, these include the programming environment and language Robolab, the language NQC, and the environment Bricx Command Center.

2.2 Robolab

ROBOLAB was created with the aim to meet the needs of the educational community for software that could be used by students of different ages and varying programming abilities.

This programming environment is the result of the cooperation among the following institutions: Tufts University, LEGO Dacta (the LEGO department which is involved with educational material) and National Instruments.

It is a graphical programming environment based on LabView; it is appropriate for PC and MAC; and it has numerous levels that require varying degrees of programming ability.

The environment's programming abilities are divided into two sections so that they can meet the needs of students of different levels. In both sections, icons are used for the representation of commands and structures.

The first and basic section, called Pilot can be seen in Fig. 1. Its limitations concern the number and the sequence of icons that the student uses for their attempt to be successful. It is suitable for young ages, as it allows for the quick and easy syntax of programs, it uses icons such as lights, arrows and clocks, which can help students during program use.

Programming in this section mainly involves choosing from the pull-down menu so that a simple program is in sequential form.

In Pilot there are 4 programming levels, which gradually increase in degree of difficulty going from the simpler programs to more complex ones with more selections for the sensors and more parameters. At the highest level, Pilot can be used to create programs which are implemented in 2 or more steps depending on the program's needs. Even though the ROBOLAB software includes a very detailed manual, the Pilot section, allows first-time users to create programs quickly and to gradually progress to more complicated programming. Especially in regards to children, teachers have ascertained that they can proceed through the 4 levels of Pilot with very little guidance.
Pilot does not enable users to fully exploit RCX's capabilities; it simply provides an environment where one can observe the connection between programming and the actions executed by RCX.

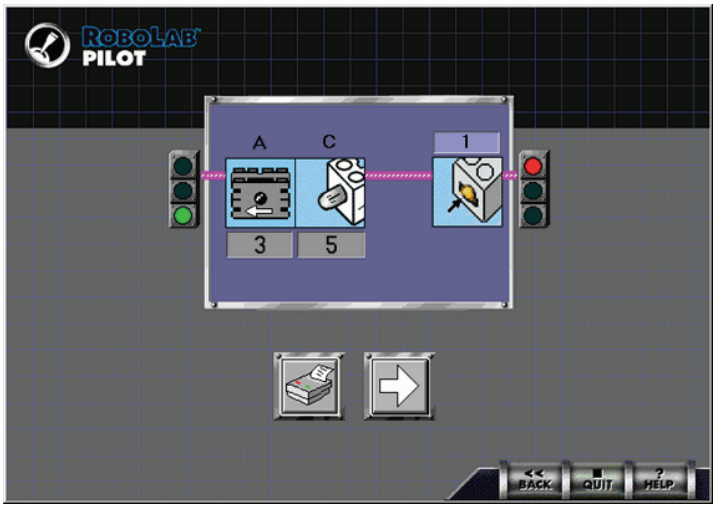


Fig. 1. A screenshot of Pilot section

These capabilities are sufficient for primary school teachers and pupils, who do not need to reach a higher level of programming. On the other hand, more experienced programmers can use Pilot to test their creations or their programming ideas quickly and easily.
The second and more advanced section is called Inventor (Fig. 2) and was created to satisfy the needs of users at higher levels who have more programming ability. The environment in this section is likewise graphical, however, it offers the user more flexibility. Here too there are 4 levels related to complexity.
In Inventor the commands that form a program are chosen from a function pallet. This gives users an unlimited number of choices, as well as a high level of responsibility for the program's success.
Inventor provides a higher level of programming structures, such as loops, variables, selection structures, multiple selections and the possibility to create subroutines. For this reason, programmers can take advantage of all of RCX's properties.
The transition from Pilot to Inventor presupposes that the user understands the rationale with which the commands-icons must be selected and connected in order to develop a program.

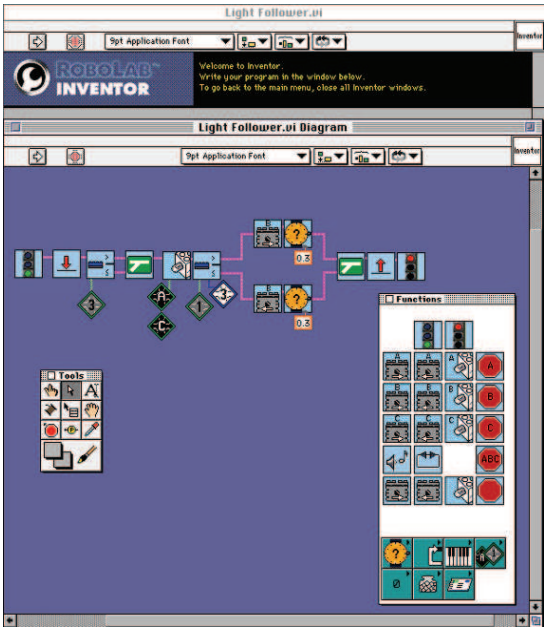


Fig. 2. A screenshot of Inventor section

The main advantage of ROBOLAB is that it applies a learning model where the large number of available commands is dependent on the level that the user has reached (http://www.umcs.maine.edu/~pbrick/).

2.3 Not Quite C (NQC)

Not Quite C was developed by David Baum and is available freely on the Internet for Windows, Unix and MacOS. It is a simple programming language with syntax similar to C; it is based on firmware, which accompanies the Robotics Invention System and it does not use Spirit.ocx. It is not suitable for teaching programming to novices. On the contrary, it is a good solution for someone who knows to program in C (http://www.baumfamily.org/nqc/). It assists the use of variables, a counter, an array and subroutines, a fact that makes it an effective programming tool.

Initially NQC was designed to operate on a command-line. Later, quite a few users created Integrated Development Environments (IDEs) whose aim was to provide a user-friendlier environment for programming with NQC. They developed easy-to-use graphical environments that incorporate the NQC compiler and are enriched with further features, such as an RCX remote control, highlighting the syntax among others (Kaskalis et al., 2001). RCX Command Center (RCXCC), WinNQC, NQCedit, Bricx Command Center and Visual NQC 2001 are such programming environments.

In the lessons we conducted, Bricx Command Center (BricxCC) was used. It is built around NQC Compiler that makes it possible to program RCX. The foundation of BricxCC is a full-featured programmer's editor in which you can write your programs with the integrated ability to compile the programs and download them to the brick. In case of errors, they are reported at the bottom of the editor window so that they can be corrected easily. In addition

to providing an editor, BricxCC contains tools to control your robot directly, to watch what is happening in the brick, to download (new) firmware, etc.

3. Using mechanical models in educational institutions

From their initial appearance in 1998 LM have attracted the attention of numerous educators and researchers. They started being used at all levels of education in order to teach a variety of subjects and fields, such as: basic programming concepts, artificial intelligence, robotics etc (Avanzato, 1998; Barnes, 2002; Genalo, 1997; Goldweber et al., 2001; Lawhead, P. & Bland, 2001; Fagin, 2000; Wolz, 2001). In an attempt to get the most detailed literature review possible regarding the use of LM as a teaching tool, we sought those institutions which have integrated LM in their syllabus. This was perceived to be necessary in order for us to be fully informed which would aid us in the preparation of the series of lessons to be conducted for the teaching of programming to 3rd year junior and 1st year senior high school students in the Greek educational system.

Since it would require a lot of space to give a detailed description of the study programs where LM have been used, we have organized the findings of this literature search into table form and present them the table below. Table 1 includes the type of lesson (incorporated in the existing curriculum, seminar etc), length-content of the lessons and educational level.

Program – Educational Institution	Type of lesson			Lengthα				Educational Level			
	Incorporated in the existing curriculum curriculum	Seminar	Extra-curricula Activities	Less than 6 months	Six months/ 12 months		Over 12 months	Primary Education	Secondary Education	Tertiary Education	Independent of educational Level
Wellesley College, "Robotic Design Studio"	X				X					X	
University of Mississippi, "Java, Event and Robots"	X				X					X	
Indiana University, "Learning with LEGO Robots"		X		X						X	
Penn State Abington College, CMPSC 201	X				X					X	
University of New Jersey, CS1	X				X					X	
Texas A & M University Corpus Christi, "Systems Analysis and Design"	X				X					X	
Colby College, "Exploration with Robots" (CS117)	X				X					X	
US Air Force Academy	X				X					X	
Xavier University	X				X					X	

University of Evansville, "LEGO 101 Project"			X		X					X	
Villanova University, "Introduction to Artificial Intelligence"(CSC4500)	X				X					X	
Case Western Reserve University, "Autonomous LEGO Robotics", BIOL/EECS 375/475	X				X					X	
Uppsala University, "Real time systems & Real time programming"	X				X					X	
National University of Singapore, "CS 3243– Foundations of Artificial Intelligence"	X			X						X	
University of Edinburgh, "Intelligent Autonomous Robotics"	X				X					X	
University of Salzburg, "Embedded Software Engineering Course, CCN 25754"	X				X					X	
Waterford Institute of Technology, "Robotics course for Industrial Computing"	X				X					X	
University of Southern California, CSCI 445 "Introduction to Robotics"	X				X					X	
University of Pennsylvania, "MEAM 247 Mechanical Engineering Laboratory"	X				X					X	
Trinity College," Computer Technology"	X				X					X	
Universität des Saarlandes, "Talking Robots with LEGO MindStorms", 2004	X				X					X	
University of New Mexico	X				X					X	
University of Applied Sciences Hamburg	X				X					X	
Iowa State University, "Toying with Technology"		X		X							X
Carnegie Mellon University, Robotics Academy		X					X				X
Bethany Christian High School	X			X					X		
Brooks-Smith School			X	X				X			
Shady Hill School			X			X			X		
Blake School			X				X	X			
S.P. Waltrip High School			X				X		X		
Davis Creek Elementary			X	X				X			

Table 1. Educational Institutions which use LM

By examining the subjects of the institutions, briefly presented in the above table, we arrived at the following conclusions:

Scientific Field - Level

- The technology of physical models has been integrated more into subjects on robotics. In a number of cases it is considered a basic precondition that students have programming experience (mandatory subject at University), in most cases however, the teaching of robot programming comprises a separate study unit.
- Most applications are implemented in primary or tertiary education.

Length – Content of Lessons

- In tertiary education, the lessons are usually incorporated into the department's curriculum that last for one semester. In contrast, the use of physical models at younger ages usually takes place within the context of extracurricular activities, which can last from a few weeks to up to a year.
- From a study of the curricula, the findings show that at tertiary level some theoretical lessons are first given, whereas for younger ages learning is accomplished exclusively through the students' activities with Legos.
- At all levels of education, lessons in computer laboratories are backed up by related class assignments. University students, were called on to design extra assignments, which lead to a consolidation and expansion of what had been discussed during the lesson, giving stimulation for study and personal involvement in the subject. During the design of assignments, university students were quite active and showed particular interest. In addition, they were more concerned about issues related to the robot's behaviour and the program philosophy rather than the syntax of a particular programming language.
- University student assignments were usually accompanied by written documentation.
- The selection of a physical model, in almost all cases, regardless of educational level, is made by the teachers.
- In addition, in almost all the university institutions, at the end of the semester there is an exhibition of all the physical models which the participants constructed and programmed; a fact which no doubt is an incentive to try harder as well as providing consistency.

Programming environment

- At younger ages preference was for one of the graphical environments (Robolab) for programming robot behaviour, while university students usually program their robots on environments based on more general-purpose languages, such as Java, Basic, C (NQC), Ada.
- The selection of programming environments in each case depends on the level of programming abilities which the student has or should attain. The needs of users who are experienced in programming are met by programming environments that are based on some general-purpose language. In contrast, novice programmers avoid such environments and turn to graphical based ones. In this way, they do not focus their attention on learning the syntax of the language, but rather on basic programming principles.

General comments

- The interest, enthusiasm, and activity shown by students were much greater than in previous years when lessons were conducted in a more traditional manner.

- It is imperative that the teachers who are likely to apply physical mechanical models in their lessons are trained. In order for these training needs to be met quite a few seminars are organized geared to high school teachers to become acquainted with robot technology, learn how it can be incorporated into the existing curriculum and to develop it within the context of the educational process.
- One fairly serious drawback was that in many cases there was a lack of sufficient numbers of robots at the informatics laboratories, which can cause difficulties in conducting the lesson.

4. Description of lessons in the Greek educational system

As stated above, LM are a contemporary tool applied in Informatics lessons in various institutions all over the world. However, few studies have been conducted regarding the effectiveness of these models in teaching in related subjects. The first general impressions in the application of LM in the classroom were encouraging. In spite of this, one of the first formal assessments of Mindstorms as a teaching tool by Fagin and Merkle (2004) showed that LM not only did not have a positive effect on students' performance but actually had a negative one. In a recent research, (Clinburn, 2006) presented projects and courses where LM were effective teaching tools and listed the lessons where LM could be applied.

Taking into consideration the findings related to the cases of LM use presented in the previous section, the research on the effectiveness of LM, the curriculum and the time constraints placed on the teaching of programming for third year junior and first year senior high school students in the Greek educational system, we designed a series of lessons for the subject of Informatics. For the teaching of programming with LM we conducted 3 series of lessons to high school students as already stated The results from the 1st series of lessons were presented in a previous paper (Satratzemi et al., 2005), where we referred to our first attempt to teach programming with the aid of LM. Our latest findings are presented in this section.

Here an analytical description is given of the lessons taught by our research team to two different groups of junior and senior high school students.

In the school year 2005-2006 the environment Robolab was used to teach 84 3rd year junior high school students. The students worked in pairs; each pair having its own PC and RCX processor. Nine lessons were conducted on the following topics:

1st Introduction to programming with LEGO. Presentation of Lego (RCX, infrared transceiver); Introduction to the programming environment Robolab; Loading–Execution of available programs on Robolab.
2nd Presentation of programming with Legos, presentation of Pilot programs levels 1, 2, 3.
3rd Input – Output command, Wait command. Presentation of Inventor programs level 1.
4th Parameters – Modifiers, Inventor level 3.
5th –6th Repeat structures Jump/land, loop
7th –8th Selection structure.
9th Containers - Modifiers

In the school year 2006-2007 the language NQC and the Bricx Command Center environment were used to teach 48 3rd year junior high school students. Likewise, the students did pair-work, and each pair had its own PC and RCX processor. There were 6 lessons on the following topics:

1st. Introduction to programming with LEGO; Presentation of Lego (RCX, infrared transceiver); Introduction to the programming environment BricCC; Loading– Execution of available programs on NQC.

2nd. Output command (Description – motor, lamp, sound); Usefulness of Wait command; Examples of how to apply commands presented.

3rd-4th. Input command (Sensors) – Repeat command; Usefulness of Repeat structure; Repeat command – until command; Examples of how to apply commands presented.

5th-6th. Usefulness of selection structure; If command; Combination of if command with repeat commands; Nested if commands; Examples of how to apply commands presented.

For both series of lessons presentations of the concepts of each lesson were developed along with worksheets, check sheets, brief notes on the basic icons of Robolab and the basic commands of NCQ.

The comments made by the teacher in 2005-2006 were taken into account for the design of the lessons in the following year. Specifically, the alterations made are the following:

1. In 2005-2006 new concepts were introduced by the teacher with the help of digital presentations, following this, the students had to answer questions in writing and they implemented their own programs in order to control the robot's behaviour. According to the teacher's observation, this approach was not particularly effective. Teaching with the use of presentations excluded the students from participating and as a consequence they did not fully comprehend the concepts taught. This resulted in producing misconceptions in terms of the significance of the commands. The commands were learnt at the stage of program development with the students often requesting the teacher's assistance.

   Hence, in 2006-2007 a problem-based approach was applied; that is, the teacher posed the problem for which the students had to develop a code. Through discussion the teacher guided the students to understanding the necessity of the existence of the related commands which would solve the problem. Following this, the teacher presented the commands to the students. The teacher used the board to present the new concepts – commands. Then, exercises were proposed to the students which were based on the implementation of the code. At the end of the lesson students had to answer to True/False questions and the recapitulation of the concepts taught in the current lesson was made through digital presentations.
2. Further, it was observed that in 2005-2006 there were too many exercises set and that they were rather similar as regards the control of the robot's behaviour. This had resulted in the students losing interest in programming as the lessons progressed since there were no challenging new situations of operation to encounter of the robot's behaviour. In addition, the students were forced to work under pressure in terms of time due to the large number of exercises they were required to get through.

   In 2006-2007 in order for student interest to be maintained, fewer exercises were set and thus the time available to solve each problem increased, allowing each group to work at their own pace. In order for the results to be comparable the problems presented below are those that were given both years.

After each exercise tables are given showing the percentages of students who solved the problem correctly, who did not solve it correctly, and who did not attempt the exercise at all. This is followed by another table with data on particular commands of the code which

are considered to be significant in formulating conclusions regarding the degree of understanding and use of the particular data of each programming language by each group of students. In the tables, G1 is the students of 2006-2007 and G2 the students of 2005-2006.
Exercise 1: Program the vehicle you constructed to move forward using the two motors for 6 seconds and then stop.

Groups	correct	Error/ misconception	No answer
G1	75.0%	21.0%	4.0%
G2	57.0%	0.0%	43.0%

Table 2a. Results of 1st Exercise

	Correct answers			
Groups	motors	time	stop	syntax
G1	83.0%	92.0%	83.0%	83.0%
G2	57.0%	57.0%	57.0%	

Table 2b. Data details of the correct answers of the 1st Exercise

Exercise 2: Program your vehicle to move forward using both motors for 4 seconds, to stop for 2 seconds and then to move in the opposite direction for 4 seconds.

Groups	correct	Error/ misconception	No answer
G1	83%	8%	8%
G2	50%	7%	43%

Table 3a. Results of 2nd Exercise

	Correct answers				
Groups	motors	direction	time	stop	syntax
G1	88%	92%	92%	83%	92%
G2	57%	57%	57%	50%	57%

Table 3b. Data details of the correct answers of the 2nd Exercise

Exercise 3: Program the vehicle you constructed to produce an audio signal and then to wait for 1 second. This procedure will be repeated 10 times.

Groups	correct	Error/ misconception	No answer
G1	92%	4%	4%
G2	93%	7%	0%

Table 4a. Results of 3rd Exercise

	Correct answers			
Groups	Repetition structure	Number of repetitions	sec	syntax
G1	96%	96%	96%	92%
G2	100%	100%	86%	

Table 4b. Data details of the correct answers of the 3rd Exercise

Exercise 4: Program the vehicle you constructed to move straight-ahead (using both its motors) until it hits an obstacle. Then to move in the opposite direction and to stop as soon as it hits another obstacle.

Groups	correct	Error/ misconception	No answer
G1	67%	33%	0%
G2	50%	43%	7%

Table 5a. Results of 4th Exercise

	Correct answers				
Groups	Repetition structure	sensor	Stop after until	Condition until	syntax
G1	100%	92%	79%	92%	92%
G2	100%		86%	50%	86%

Table 5b. Data details of the correct answers of the 4th Exercise

Exercise 5: Program the vehicle you constructed to move (using both its motors) with the lamp on. When the light level of the room increases by 5 the following should occur for one second: the lamp turns off and the vehicle stops moving. This procedure will be repeated 4 times. At the end a sound is heard.

Groups	correct	Error/ misconception	No answer
G1	46%	25%	29%
G2	14%	79%	7%

Table 6a. Results of 5th Exercise

	Correct answers					
Groups	Repetition structure	Number of repetitions	Light, motor	Wait for	Stop	sound
G1	67%	71%	58%	63%	58%	67%
G2	79%	79%	71%	21%	64%	71%

Table 6b. Data details of the correct answers of the 5th Exercise

Exercise 6 Program the vehicle you constructed to produce a sound SOUND_DOWN if the touch sensor in the front of the vehicle has been pressed, otherwise, if it has not been pressed, it must move for 3 seconds.

Groups	correct	Error/ misconception	No answer
G1	67%	29%	4%
G2	57%	43%	0%

Table 7a. Results of 6th Exercise

	Correct answers				
Groups	If/Fork	Condition (if/fork)	sound	motor	merge
G1	96%	92%	92%	79%	
G2	57%	57%	100%	-	100%

Table 7b. Data details of the correct answers of the 6th Exercise

Exercise 7: Make a program which will light up the lamp of the vehicle you constructed for 6 seconds, if the touch sensor at the back of the vehicle is pressed. Otherwise it will move the vehicle for 2 seconds in a forward direction.

Groups	correct	Error/ misconception	No answer
G1	63%	25%	13%
G2	36%	57%	7%

Table 8a. Results of 7th Exercise

	Correct answers					
Groups	if	condition	Touch sensor	lamp	motor	syntax
G1	83%	71%	75%	71%	71%	75%
G2	79%	79%	79%	36%	71%	93%

Table 8b. Data details of the correct answers of the 7th Exercise

Exercise 8: Make a program that will produce a sound 4 times, if the light sensor detects a value smaller than 60. Otherwise, the vehicle must move in a forward direction for 4 seconds.

Groups	correct	Error/ misconception	No answer
G1	17%	42%	42%
G2	14%	50%	36%

Table 9a. Results of 8th Exercise

	Correct answers					
Groups	if	condition	Touch sensor	repeat	motor	syntax
G1	86%	50%	93%	79%	79%	100%
G2	14%	57%	57%	36%	43%	64%

Table 9b. Data details of the correct answers of the 8th Exercise

Below we present a sample of the questions given to students, which refer to issues of understanding the commands and locating program errors.

Question 1

List the categories of sensors.

Groups	correct	Error/ misconception	No answer
G1	71%	29%	0%
G2	22%	64%	14%

Table 10a. Results of 1st Question

Groups	touch	light	temperature	rotation
G1	100%	92%	92%	75%
G2	64%	79%	64%	21%

Table 10b. Data details of the correct answers of the 1st Question

Question 2

Where is the mistake? The program must activate motor A for 4 seconds and then stop it.

Find the error in the program below:

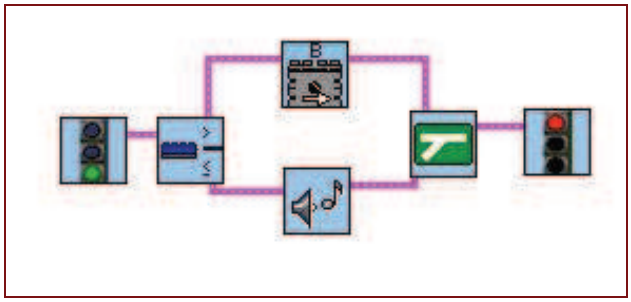
After loading the program the robot must move backwards and stop as soon as it comes up against an obstacle with the help of the touch sensor in port 3. The program must activate motor A for 4 seconds and then stop it.

```
task main()
{
      SetDirection(OUT_A+OUT_C, OUT_REV);
      SetPower(OUT_A+OUT_C, 5);
      SetOutput(OUT_A+OUT_C, OUT_ON);
      until (SENSOR_3==1);
}
```

Groups	correct	Error/ misconception	No answer
G1	79%	0%	21%
G2	86%	0%	14%

Table 11. Results of 2nd Question

Question 3
Find the error in the program below :



```
task main()
{
 SetSensor(SENSOR_1, SENSOR_TOUCH);
 if (SENSOR_1==1)
 {
    SetDirection(OUT_A+OUT_C,OUT_FWD);
    SetOutput(OUT_A+OUT_C, OUT_ON);
    Wait(200);
    SetOutput(OUT_A+OUT_C, OUT_OFF);
 }
 else
 {
    PlaySound(SOUND_CLICK);
 }
}
```

This program must produce a sound when the light sensor detects a value below 50. Otherwise, it must activate motors A and C for 2 seconds in a forward direction. This program must activate motor B when the light sensor detects the light level above 80, otherwise it will produce a sound signal.

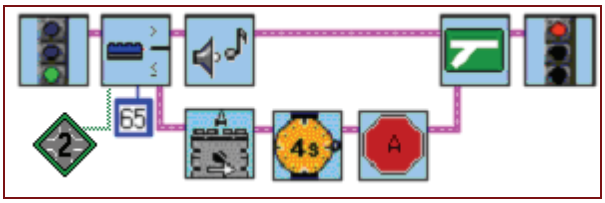
Groups	correct	Error/ misconception	No answer
G1	100%	0%	0%
G2	36%	57%	7%

Table 12a. Results of 3rd Question

Groups	SetSensor / sensor	Condition / expression for 50
G1	100%	100%
G2	36%	93%

Table 12b. Data details of the correct answers of the 3rd Question

Question 4
A) What behaviour do you expect your robot to have?
B) Change your program so that it produces the same result (only for students in NQC).




```
task main()
{
 SetSensor(SENSOR_2, SENSOR_LIGHT);
 SetSensor(SENSOR_1, SENSOR_TOUCH);
 if (SENSOR_2>=20)
 {
    SetDirection(OUT_A+OUT_C,OUT_FWD);
    SetOutput(OUT_A+OUT_C, OUT_ON);
    until (SENSOR_1==1)
    SetOutput(OUT_A+OUT_C, OUT_OFF);
    PlaySound(SOUND_CLICK);
 }
 else
 {
    SetOutput(OUT_B, OUT_ON);
    Wait(300);
    SetOutput(OUT_B, OUT_OFF);
    PlaySound(SOUND_CLICK);
 }
}
```

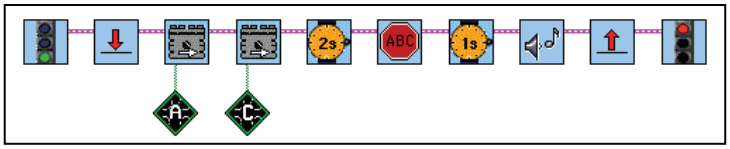
Groups	correct	Error/ misconception	No answer
G1	100%	0%	0%
G2	57%	43%	0%

Table 13a. Results of 4th Question

Groups	Use of if	condition	change
G1	100%	100%	100%
G2	79%	64%	

Table 13b. Data details of the correct answers of the 4th Question

Question 5
A) What behaviour do you expect your robot to have?




```
task main()
{
    repeat(3)
    {
        SetDirection(OUT_A+OUT_C, OUT_REV);
        SetPower(OUT_A+OUT_C, 5);
        SetOutput(OUT_A+OUT_C, OUT_ON);
        Wait(200);
        SetOutput(OUT_A+OUT_C, OUT_OFF);
        Wait(100);
        PlaySound(SOUND_CLICK);
    }
}
```

Groups	correct	Error/ misconception	No answer
G1	58%	25%	17%
G2	100%	0%	0%

Table 14. Results of 5th(A) Question

(B) Change your program so that the sound is produced only once, before the motor is activated.

Groups	correct	Error/ misconception	No answer
G1	46%	21%	33%
G2	57%	36%	0%

Table 15. Results of 5th(B) Question

5. Evaluation of the findings

An analysis of the data of the previous section is made here. A comparative analysis and evaluation of the results for both years of teaching will be attempted. The use of a different programming language and environment give the opportunity to formulate hypotheses in relation to the source of the difficulties, such as problems on account of the environment and the language, problems owing to the mechanical models and lastly to intrinsic problems of programming concepts which appear independently from the language or the environment.
Analysis of results of exercises:
In the majority of the exercises solved with Robolab a large percentage of students forgot to use the stop icon in order to stop the motors or the lamp.

Exercises 1 & 2: The use of the icon which determines waiting for a particular period of time appears to present some difficulty for the students in Robolab where only 57% used the correct icon, in contrast to NQC where 92% used the related command correctly.

Exercise 3: Robolab significantly favours the understanding of the repeat structures since all students with Robolab solved the exercise while a small percentage with NQC developed an incorrect code.

Exercise 4: The use of the repeat structure with Robolab is indirect since the icon that refers to the touch sensor (on/off) is used. It appears that this icon is used effectively, however, almost half the students did not use the modifier to determine the input port between the sensor located in the front and back parts of the robot. The percentage of programs with incorrect syntax was larger with the use of Robolab.

Exercise 5: More students used the repeat structure correctly in Robolab than in NQC (79% and 67% respectively). In Robolab students confused the icon related to the light sensor with the touch sensor and usually did not use the correct icons for the modifiers to control the level of lighting or the equivalent port which the light sensor is connected to. In NQC, the command is wait while in Robolab students must use the icon From the results, this seems to have created difficulties for the students since a mere 21% used the correct icon in conjunction with the correct Modifiers.

Exercise 6: The percentage of students who solved this exercise with Robalab is smaller than with NQC. This exercise is related to the select structure If. It must be noted that although more time was needed in order to explain the if structure in NQC than in Robolab, in the end students made more mistakes in the code with Robolab since they forgot to include in the if/fork icon the modifier that determines which input port the touch sensor is connected to. The icon which represents sound appears to be used more effectively in Robolab than in NQC. As in exercise 5, the majority of students also had problems in this exercise with the syntax of the sound command in NQC than in Robolab.

Exercise 7: In this exercise only 36% of students with Robolab used the correct icon for the operation of the lamp. Often they got the lamp icon confused with that of the sensor. On the contrary with NQC, the percentage of students which used the lamp correctly is practically double (71%), since in their code they did not require a particular command for the lamp. In NQC the commands for the operation of the components which are connected to the output ports are the same, the activation of the motor or the lamp and the determining of their power is defined by the same commands, which are SetOutput & SetPower respectively, as long as, of course, in each one of these commands is included the respective port where each mechanism is connected. Unfortunately, in Robolab different icons are needed for the lamp and the sensor, and in fact the two icons are next to each other, with the result that the students often confused them by selecting the one instead of the other.

Exercise 8: A small percentage of students solved this exercise in both environments. We observe that also in this exercise which refers to the use of if with a nested repeat structure, although the students understood both structures much faster in

Robolab than in NQC in the end they made more mistakes in Robolab. A mere 14% used the correct if icon to control whether the light sensor detected a value smaller than 60. In this case also they confused the light sensor icon of the select structure with that of the lamp. Further, in the repeat command in Robolab 36% used the modifier to state the number of repetitions, whereas 79% in NQC correctly developed the repeat structure. If we compare the percentage of students who developed the repeat structure correctly in exercises 3, 5 and 8, we see that it is 96%, 71% and 79% respectively in NQC, while in Robolab it is 100%, 79%, and 36% respectively. An explanation for this significant reduction in the percentage of correct responses in Robolab, in contrast to the relative consistency which the responses in NQC present, is that when the complexity of the code increases, then the students' code in Robolab presents more errors even for commands which appear to have been comprehended faster and easier in Robolab.

Analysis of the question results:

Question 3: For both years we see that the percentage of incorrect answers is similar and in fact in 2006-2007 slightly higher. It appears that students do not consider the icon or the command related to stopping the operation of the components found at the output ports of the mechanism to be necessary. In other words, the necessity of the stop icon in Robolab or the command SetOutput with the parameter OUT_OFF in NQC, is probably understood by the students as a feature of the mechanism and they do not recognize the equivalent icon or command when these are missing from their programs, but also often they do not include them in the code they develop.

Question 4: In this question, which concerns locating the error in the port that the light sensor is connected to, we observed that the students who worked with Robolab had a very low percentage of correct responses (36%) in relation to those who worked with NQC (100%). In the incorrect Robolab program the students had to add the modifier to the if icon, which determines the port which the light sensor is connected to.

Question 5: 64% of students with Robolab could correctly describe the condition in the control structure. In NQC all students recognized the duplication of the code not only in the if structure but also in the else structure.

Question 6: In the questions regarding the repeat structures students with Robolab achieved a higher percentage than students with NQC. The majority of students with NQC described the body of commands included in the repeat structure correctly but they did not describe the actual command repeat. They probably did not have enough time as it was one of the last questions they were given, or the icon in Robolab which describes the repeat command is quite effective and the students understand the operation of this command quickly without misconceptions, something which does not appear to happen to the same effect in NQC.

General Observations recorded by the teacher:

- The graphical environment of Robolab did not particularly facilitate the syntax of the code. In Robolab students had a problem with connecting the icons together, whereas the same was not observed for NQC.

- A graphical environment helps in the first lessons in terms of time needed to comprehend and develop the code but the use of modifiers counteracts this initial positive start.
- With NQC more time was needed to explain if in relation to Robolab where the graphical representation makes it easier and faster to for this structure to be understood.
- Modifiers are the components of Robolab which produce problems since they involve a large number of different icons, such as regulating the input/output ports, sound, light level, the number of repetitions in the repeat structures etc, which not only did students not apply correctly, but very often they confused them.
- It was ascertained that how the general properties of the "machine" the student learns to control and its relationship to the physical machine are understood, did not produce the same type of ambiguities in both environments. In other words, physical models help students to understand the "mechanism" of program execution and to associate commands of the code with the robot's activities.

6. Conclusions

Below, conclusions are drawn of the findings from the 2 series of lessons; they are related to other studies, and proposals for further research are made.

The analysis of the findings of the two-year study allowed us not only to make a comparison of the teaching methods applied each but also to formulate certain general conclusions in relation to the application of LegoMindstorms robots within the framework of an introduction to programming in secondary education.

Our teaching approaches with the use of physical models present certain features relative to earlier studies:

- Our lessons took place within a specified curriculum at the secondary educational level. This is a significant factor in evaluating the representational value of our results. One consequence of this conditions was, for example, that the participation in the lesson was mandatory thus not all students were highly motivated, as would have been the case in an elective subject. In addition, the lessons occurred within a context, which to some extent determined certain teaching parameters, such as imposed teaching objectives as part of the school syllabus. Furthermore, a relatively large number of students were obliged to participate in the lessons and generally the whole framework within which the lessons were conducted was a typical classroom setting. In some way, therefore, we consider that our lessons correspond more to the average student at secondary education level and subsequently the related findings are fairly representative.
- Our lessons were in the context of an introduction to programming (not robotics or technology). This means that certain aspects of the students' activities were more important than others: for instance, the learning and correct use of common programming structures (such as selection and repeat) were more significant than the constructional features of the robotic systems applied. Thus, while the usual implementation of robotic systems in teaching focuses on construction and "behaviour", we used robots as a means of teaching the basic concepts of programming.
- Our lessons were conducted during two consecutive school years using 2 different programming environments. As previously mentioned, the change of programming

environment in the second year occurred in an attempt to improve the effectiveness of the lessons. For the same reason certain lessons were also altered in that same year. After finalizing the second series of lessons, we made a new set of proposals intended to further improve the teaching methodology. However, these findings will not be discussed here as they are relatively extensive and cannot be incorporated into the scope of the present discussion.

Our general finding could, therefore, be summarized as follows:

Generally, the application of physical models, such as robots, undoubtedly present advantages since they are attractive and thus constitute a strong incentive for students. Nevertheless, as observed by other researchers, their use does not automatically mean that all students fully understand all the new concepts and techniques presented, nor of course to the same degree. Therefore, the use of these environments, in order for them to have teaching value, require a careful design of activities, time etc. In reality, the evaluation of the pros and cons of the use of these environments in teaching is fairly complex, since, as our findings show, they demand explicitly specified teaching conditions in order to achieve positive results. In addition, student results seldom have an "absolute" character; for example, the comparative findings between the two environments used in our study did not clearly favour one or the other environment, since at the initial stages it appeared that one prevailed, while at more advanced stages the other seemed to do so (see below for a detailed analysis). In other circumstances, the students appear to understand certain basic concepts or mechanisms but they make important mistakes when answering the related questions. Consequently, we are not yet in a position to formulate general rules that describe student behaviour in relation to the difficulties they encounter. It is characteristic, therefore that, as referred to in the comparative study in the above section, even the incentives for students must, in some way, be "pre-designed, as often their interest rapidly wanes when progress in the lesson is "slow" or when the proposed activities are not "interesting". In certain cases, it appears that the teaching demands are incompatible or even conflicting with students' incentives.

At another level, as was anticipated, the understanding of the "machine's" general properties that the student learns to control and its relation to the physical model did not produce problems in either environment.

Nevertheless, many of the difficulties encountered in teaching an introduction to programming were present in our lessons as well. In many cases, for example, the requirements for programming the robots comprised a source of difficulty for the students when it is it is different to their daily experience. A characteristic example of this is the command or symbol STOP (or Wait respectively) which is needed to stop the operation of different parts of the mechanism often appears to be ignored by students, as is the use of modifiers for the selection of the correct input port. The students do not include the related symbols in their programs, nor do they indicate the absence of these in pre-prepared programs they have been given to check for mistakes, even though they show that they have an understanding of the role these play and the way they are used. A possible explanation is that students are not familiar with the need for an accurate and explicit formulation of the respective commands, which in their daily experience occur in an indirect manner (for example, finalizing an operation of the mechanism or after some predetermined time span, usually for students means that the mechanism stops without this requiring a specific command in order to do so). Difficulties of this nature are observed in introductory teaching to programming almost irrespective of the environment used.

Our comparative study also enabled us to examine the relative teaching value of the environments which use iconic or "textual" programming languages, an issue which has often preoccupied the international research community.

The use of an iconic language naturally requires the selection of icons and symbols which refer to semantics while at the same time these icons and symbols must be distinctly different to avoid getting them confused (as in the case with the light and touch sensors, or the lamp and light sensor which the students often mix up). The characteristic icons, however, are not always easy to create, since there are elements in the programming of robots which are represented better with words rather than with icons. A typical example of this are modifiers: it is difficult for someone to "guess" the purpose or how they are used by simply looking at the icon. In addition, the large number of icons required even for common programs, could create confusion since the students must choose the appropriate icon among a range of icons that are alike. The weakness of parameterizing the iconic environments seems to produce yet another problem, as for every operation an appropriate icon must be created ad hoc, which refers to it. In contrast, in textual languages the simple existence of parentheses in the syntax of a command refers directly to the need to give a value to the particular parameter. In this way, while in the initial stages in the negotiation of a concept, an iconic environment appears to be better, later, however, this advantage is lost.

A similar thing seems to happen with the repeat structures, since the repeat structure is obvious in the flow chart created by the icons. Students' mistakes, however, are many when the problem requires nested structures and generally a more concise structure and a more complex code than that required for a simple repetition. Lastly, a similar phenomenon was located in the use of the selection commands if.

In some way although written commands are less direct than icons, they permit the creation of relatively complex structures with greater ease.

It is likewise characteristic that while naturally in iconic environments syntactic errors are not an issue, nevertheless, students appear to have difficulty in the "syntax" of their programs, meaning that they found it difficult to connect and correctly locate the icon – commands.

The nature of the problems encountered within the context of the lessons, did not allow for further investigation of students' perception to the lessons and generally to the evaluation of the teaching benefits of these types of environments. It was not possible to examine student strategies on problems which were more complex and algorithmically more interesting. Our findings, however, allow us to conclude that taking into account the conditions described above, teaching with physical models such as robots appears to be enticing to students and helps them to learn elementary programming concepts.

Two adjacent results arising from the lessons conducted, which did not, however, constitute set questions in our research, must nevertheless be mentioned.

The first is perhaps interesting for those who intend to apply physical models in their teaching, like the ones we describe. The use of these models requires a much longer preparation period than that required by using only software. The reasons are quite obvious: since the physical models operate as mechanisms, they must be regularly checked to ensure that they are operating properly (for example, if there is a satisfactory level of energy in the batteries, if the connections are good, etc).

The second is related more to teaching as such rather than the preparation of the material. Since these systems operate under real conditions, they are not governed by the absolute determinism of the software. So, an operational command of a motor, for instance, does not

always cause the vehicle to move for the same distance (it depends on the energy of the battery, ground friction etc). Moreover, certain difficulties encountered by users came precisely from the fact that they were dealing with physical systems, which the students must implement with precision.

Although neither of these points is a serious drawback to teaching, they, nevertheless, do exist and they do affect teaching, even if to a small degree.

The findings so far allow us to formulate the conclusion that physical models comprise a good alternative solution for an introduction to programming, but further research must be conducted, which will determine those conditions with greater accuracy (e.g. selection of programming environment in accordance with teaching aims) which will enable teaching to be more effective.

8. References

Avanzato, R. (1998). Collaborative Mobile Robot Design in an Introductory Programming Course for Engineers, *Proceedings of the ASEE Annual Conference* ", June 1998, Seattle, Washington.

Barnes, D. (2002). Teaching Introductory Java through LEGO Mindstorms Models, *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education,* pp. 50 – 54, Northern Kentucky, Cincinnati, February 2002, ACM.

Becker, B. (2001). Teaching CS1 with Karel the Robot in Java, *Proceedings of the Thirty Second SIGCSE Technical Symposium on Computer Science Education*, pp. 50 – 54, Charlotte, NC USA, February 2001, ACM.

Bergin, J.; Stehlik, M.; Roberts & Pattis J. (1997). *Karel++ - A Gentle Introduction to the Art of Object-Oriented Programming*, *2nd edn. New York, Wiley*.

Brusilovsky, P.; Calabrese, E.; Hvorecky, J.; Kouchnirenko, A. & Miller, P. (1997). Mini-languages: a way to learn programming principles, *Education and Information Technologies. 2*, pp. 65-83.

Brusilovsky, P.; Kouchnirenko, A.; Miller, P. & Tomek, I. (1994). Teaching programming to novices: a review of approaches and tools. *Proceedings of ED-MEDIA '94*, pp. 103-110, Vancouver, British Columbia, Canada, June 1994.

Cliburn, D. (2006). Experiences with the LEGO MindstormsTM throughout the Undergraduate Computer Science Curriculum, *36th ASEE/IEEE Frontiers in Education Conference*, pp. T2F1-6, San Diego, October 2006, CA, IEEE.

Du Boulay, B.; (1989). Some Difficulties Of Learning To Program, *In Studying The Novice Programmer*, Soloway, E., Sprohrer, J. (Ed.), Lawrence Erlbaum Associates, pp. 283-300.

Du Boulay, B.; O'Shea, T. & Monk, J. (1989). The Black Box Inside the Glass Box: Presenting Computing Concepts to Novices, In: *Studying The Novice Programmer*, Soloway, E., Sprohrer, J. (Ed.), pp. 431-446, Lawrence Erlbaum Associates.

Fagin, B.; (2000). Using Ada-Based Robotics to Teach Computer Science, *Proceedings of the 5th Annual Conference on Innovation and Technology in Computer Science Education*, pp. 148-151, June 2000, Helsinki, Finland, ACM.

Fagin, B.; Merkle, L. (2003). Measuring the Effectiveness of Robots for Teaching Computer Science, *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2003)*, pp. 307-311, February 2003, Reno, Nevada, ACM.

Genalo, L. (1997). Toying With Technology: Mobile Robots and High School Interns, *Proceedings of the ASEE Annual Conference*.

Goldweber, M.; Congdon, C.; Fagin, B.; Hwang, D.; Klassner, F. (2001). The Use of Robots in the Undergraduate Curriculum: Experience Reports, *Proceedings of the Thirty Second SIGCSE Technical Symposium on Computer Science Education*, pp.404-405, Charlotte, North Carolina, February 2001, ACM.

Hirst, A.; Johnson, J. ; Petre, M.; Price, B. & Richards, M. (2003). What is the best programming environment / language for teaching robotics using Lego Mindstorms?, *Artificial Life Robotics*, Vol. 7, pp.124-131.

Kaskalis T.; Dagdilelis V.; Evangelidis G. & Margaritis K. (2001). Implementing Applications on Small Robots for Educational Purposes: Programming the LEGO Mindstorms, *Proceedings of the 5th Hellenic – European Conference on Computer Mathematics and its Applications (HERCMA 2001)*, pp. 337 – 341, Athens, Greece, September 2001.

Lawhead, P. & Bland C. (2001). Events, Robots and Programming Using Legos in CS1, SIGCSE Bulletin, Proceedings of the 6th Annual Conference on Innovation and Technology *in Computer Science Education*, Vol. 33, No 3, June 2001, ACM.

Mayer, E. (1989). The Psychology of How Novices Learn Computer Programming. *In Studying The Novice Programmer*, Soloway, E. and Sprohrer, J. (Eds.), Lawrence Erlbaum Associates, pp. 129-160.

Mendelsohn, P.; Green, T.R.G. & Brna, P. (1990). Programming Languages in Education: The Search for an Easy Start. *In Psychology of Programming*, Green H., T., Samurcay, R. & Gilmore, D. (Ed.), Academic Press, pp. 175-200.

Pattis, R. E.; Roberts, J. & Stehlik, M. (1995). *Karel - The Robot, A Gentle Introduction to the Art of Programming*. New York, John Wiley & Sons.

Resnick M. (1993). Behavior Construction Kits, *Communications of the ACM*, Vol. 36, No. 7, pp. 64-71.

Satratzemi M.; Dagdilelis V.; Kagani K. (2005). Teaching Porgramming with robots: A case Study on Greek Secondary Education, P. Bozanis, E.N. Houstis, (Ed.), pp. 502-512, *Lecture Notes in Computer Science (LNCS)*, Vol. 3746.

Wolz U. (2001). Teaching Design and Project Management with Lego RCX Robots, *Proceedings of the Thirty Second SIGCSE Technical Symposium on Computer Science Education*, pp. 95-99, Charlotte, North Carolina, February 2001, ACM.

Xinogalos, S. (2002). *Educational Technology: A Didactic Microworld for an Introduction to Object-Oriented Programming* (in Greek), Ph.D. Thesis, Dept. of Applied Informatics, University of Macedonia.

Service Robot Applications

Edited by Yoshihiko Takahashi

ISBN 978-953-7619-00-8
Hard cover, 400 pages
Publisher InTech
Published online 01, August, 2008
Published in print edition August, 2008

The aim of this book is to provide new ideas, original results and practical experiences regarding service robotics. This book provides only a small example of this research activity, but it covers a great deal of what has been done in the field recently. Furthermore, it works as a valuable resource for researchers interested in this field.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Maya Sartatzemi, Vassilios Dagdilelis and Katerina Kagani (2008). Teaching Introductory Programming Concepts with Lego MindStorms in Greek High Schools: A Two-Year Experience, Service Robot Applications, Yoshihiko Takahashi (Ed.), ISBN: 978-953-7619-00-8, InTech, Available from: http://www.intechopen.com/books/service_robot_applications/teaching_introductory_programming_concepts_with_lego_mindstorms_in_greek_high_schools__a_two-year_ex

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821