A Visual Programming Language for Educational Robotics Based on Constructionist Ideas

Leonardo Cunha de Miranda, *professor@leonardocunha.com.br*
Institute of Computing, University of Campinas (UNICAMP), Campinas, Brazil

Fábio Ferrentini Sampaio, *ffs@nce.ufrj.br*
Electronic Computing Center and Informatics Postgraduate Program, Federal University of Rio de Janeiro (UFRJ), Rio de Janeiro, Brazil

José Antonio dos Santos Borges, *antonio2@nce.ufrj.br*
Electronic Computing Center, Federal University of Rio de Janeiro (UFRJ), Rio de Janeiro, Brazil

Abstract

Educational robotics is a playful and challenging activity that puts an emphasis on education during the creation of hardware and software based solutions. This work presents a visual programming environment built based on constructionist ideas. This computational solution allows students to program and to control electronic components such as LEDs, displays, motors, and light/temperature sensors, connected to different hardware of educational robotics using graphics elements of a visual programming language. One of the main distinguishing factors of this environment is the possibility to visually simulate the implemented logic on the screen before transferring it to the hardware.

Keywords

Visual Programming Environment; Educational Robotics; Hardware and Software; Computers in Education

Introduction

For many years researchers have been debating the different possibilities of using Information Technologies (IT) in educational settings. They seek to establish with these new technologies teaching and learning environments which are rich and motivating for learners.

Among the broad spectrum of ideas and proposals regarding computing artefacts, it is remarkable to note that most of the solutions presented explore predominantly the software. However, the demand for new hardware devices in education is growing, evidenced mainly by the efforts of the academic community to propose the inclusion of robotics for teaching purposes, supported by positive results presented in, e.g., Silva (2009), Alimisis *et al.* (2007), Demo and Marcianó (2007), Norte *et al.* (2005), Alimisis *et al.* (2005), Alves *et al.* (2005), Santos and Menezes (2005), Zilli (2004), Steffen (2002), Chella (2002), d'Abreu *et al.* (2002), and Kouznetsova *et al.* (2001).

Educational robotics is a challenging and fun activity that allows students to create solutions, whether they are composed of hardware or software, aimed at solving a particular problem. Most educational projects that use robotics in the classroom make use of constructionist approaches to support the teaching process, giving to the students a real possibility of knowledge construction while they develop their projects. In other words it is said that in so far as the students are deeply engaged in such activities they also have the opportunity to develop a more accurate understanding of scientific phenomena. Thus, robotics is a new educational tool that is available to the teacher, through which many theoretical concepts, sometimes difficult to understand, can be shown in practice, motivating both the teacher and primarily the student.

According to Zilli (2004), educational robotics can develop the following competencies: logical thinking, manipulative and aesthetic skills, integration of concepts learned in various areas of knowledge for development projects, representation and communication, work with research, problem solving through trial and error, the application of the theories in concrete activities, use of creativity in different situations, and critical thinking related to the topics covered by the project.

One may mention some advantages with the adoption of educational robotics kits on the market in general: 1) hardware and software products targeted to meet specific educational purposes; 2) flexibility to use them in different applications; 3) existence of technical user documentation, including in some cases, teaching materials; and 4) easier to own and operate by users unfamiliar with the technologies involved (electronics and computers).

The focus of this paper is to present a software solution, i.e., the ProgrameFácil[1] environment, implemented based on constructionist ideas during the Master's research of the first author. The choice for the development of ProgrameFácil was driven by the need for a visual programming language – some possible forms of a visual representation are presented and discussed in Chang (1987) – that has a user-friendly interface allowing users to program the hardware of an educational robotics kit, also developed by the same researchers team, called RoboFácil (Miranda, 2006). The process of programming with ProgrameFácil had to be intuitive, devoid of command-line interface, and without the need to know the electronic architecture of the hardware.

This article is organized as follows: the next section gives a general introduction to ProgrameFácil. Following this the reader will find four subsections giving more details about the interface of this environment, its main objects, how it operates, and some examples. Moreover,

[1] The name of this environment in Portuguese – ProgrameFácil – means easy programming.

in a later section, we present some constructionist ideas that provided the basis for this project. In the end, the authors draw some conclusions and present avenues for future work.

The ProgrameFácil Environment

ProgrameFácil is a Visual Programming Language (VPL) based on the manipulation of graphical icons that enables programming electronic and/or electromechanical devices, such as LEDs, displays, light and temperature sensors, and step motors making use of iconic symbols to encapsulate some traditional programming structures such as conditionality and repetition. Initially, the language was designed to control the RoboFácil's hardware, since its original version could only be programmed using assembly and/or C languages.

The conception, design and implementation of ProgrameFácil always took into account the need to create an intuitive environment in order to make it pleasant to use and an efficient resource to control the electronic and the RoboFácil's hardware. In this sense it was designed to present to the user an interactive environment consisting of two hypothetical worlds: the first one, called *MyWorld*, specifies the desired configuration of the hardware – e.g., LEDs, motors and sensors – and presents its behaviour while simulating a program developed by the user. The second, called *MyProgram*, is the place where the user constructs the program which will control the hardware detailed in *MyWorld*. Such worlds are presented in the ProgrameFácil environment through two windows. Figure 1a shows the *MyWorld*'s window and Figure 1b presents *MyProgram*'s window.

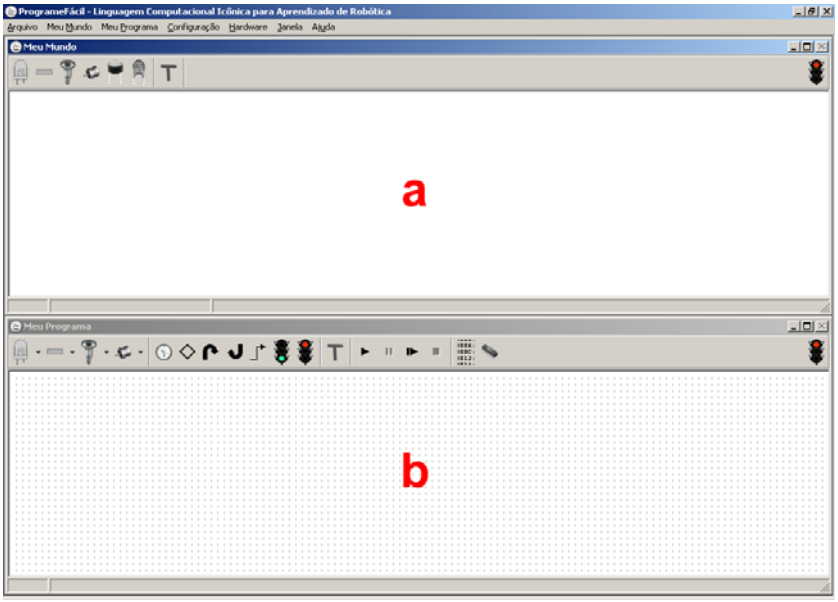

Figure 1. The ProgrameFácil environment presenting MyWorld (a) and MyProgram (b) windows

The adoption of explicit and different stages to draw and run/simulate models – mapped on the environment in different windows – aims to facilitate the investigation of the logic used in each program created by the user. When the user is satisfied with the behaviour of the program in his/her computer he/she can download it to RoboFácil's hardware. In this case a compiler is invoked to convert the icons[2] that make up the program into assembly macro-codes[3]. The

[2] Icons in the context of the ProgrameFácil environment can be defined as graphic symbols representing electronic devices or structures in programming languages.
[3] In this work the term assembly macro-code refers to the hexadecimal code set – bytecode – which represents the virtual assembly of RoboFácil's hardware.

interpreter in RoboFácil's firmware – discussed in detail in Miranda (2006) – in turn then converts programs written in the assembly macro-codes generated automatically by ProgrameFácil into statements that can be implemented in the hardware, such as activating/deactivating an LED, writing messages on the display or moving the step motor, among other possibilities.

MyWorld and *MyProgram* Windows

MyWorld is the window where the user can specify the hypothetical world that represents a hardware configuration of an educational robotic kit. The concrete objects that represent the electronic elements available for selection by the user are displayed in a toolbar of this window, except for the comment-object which applies only to allow the insertion of text in the template.

Figure 2 shows the toolbar of the *MyWorld* window with its hardware-objects: 1) LED, 2) display, 3) lamp, 4) motor, 5) light sensor, 6) temperature sensor, and 7) comments. The hardware-objects presented in *MyWorld* were abstracted from real life. Therefore, to associate them with a physical hardware, it is necessary to know their physical characteristics and actions allowed in reality.

Figure 2. MyWorld's window toolbar

The *MyProgram* window can be defined as the place where the user builds the program that will control the operation of existing objects in *MyWorld*. This process takes place by defining the actions and links between control structures such as conditional and repetition, using iconic symbols to represent them. These symbols are presented in the toolbar of the *MyProgram* window.

Figure 3 shows the toolbar of the *MyProgram* window with the following hardware-objects presented: 1) LED, 2) display, 3) lamp, and 4) motor. Also available here are the programming-objects: 5) timer, 6) IF conditional control structure, 7) looping-start, 8) looping-end, 9) line of programming, 10) program-start, 11) program-end, and 12) comments.

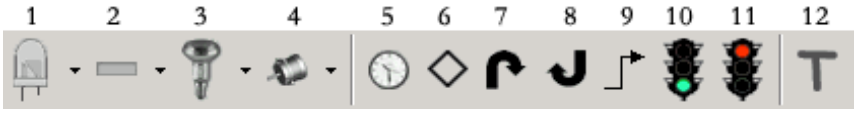

Figure 3. MyProgram's window toolbar

The goal is to make it possible to construct a programming logic between these elements, thus forming what is defined in the context of the ProgrameFácil as the Program of Model. To achieve this purpose the language was built with five rules:

- **1st:** Each object has one or no successor in the logical structure of programming;
- **2nd:** The program-end object (Object 11 of Figure 3) – which represents the end of the program – cannot have successors;
- **3rd:** The object IF (Object 6 of Figure 3) – which represents the IF conditional control structure – will have up to two successors;
- **4th:** Each object can have one or more predecessors in the logical structure of a program;
- **5th:** The program-start object (Object 10 of Figure 3) – which represents the beginning of the program – cannot have predecessors.

The inclusion of new hardware-objects in the *MyWorld* window and, as a consequence, in the *MyProgram* window depends their existence in the hardware.

When a program is simulated the executor starts with the programming-object program-start – represented by a green traffic light – and ends with the programming-object program-end (represented by a red traffic light).

Objects

An object in ProgrameFácil is a graphical representation, similar to an icon, that can be manipulated in both the *MyWorld* and *MyProgram* windows. Objects in ProgrameFácil were divided into three categories to better identify their purposes: hardware-objects, programming-objects, and supportive-objects.

The hardware-objects represent electronic components and were divided into two sub-categories: input-hardware-objects and output-hardware-objects. Programming-objects refer to common structures used in programming languages, e.g., loops, conditionals and delays. In its turn, the supportive-objects are intended exclusively to provide facilities and operational resources to the user, such as the possibility to include comments in the models.

The output-hardware-objects are presented in both windows, but have very different characteristics, e.g., the output-hardware-object LED in *MyWorld* has as property named *Color* to distinguish the color of the LED the user wants to work with. In *MyProgram* this same object has a property called *Set* used to set the LED to be on or off during the execution/simulation of the model.

In practice, to create a program using the ProgrameFácil VPL, you must perform three distinct steps: 1) select the hardware-objects to be used in *MyWorld* window, 2) include in *MyProgram* the representation of hardware-objects selected, and 3) create the flow of programming in the *MyProgram* window by selecting the appropriate icons in the *MyProgram*'s window toolbar.

Simulator

The environment provides a compiler which converts the programs constructed with the ProgrameFácil VPL into assembly macro-codes that can be executed by RoboFácil's hardware. The translation is performed by matching assembly macro-codes for each hardware-object and/or programming-object presented in the program created by the user, which then will be understood by the parser component of the RoboFácil's firmware.

The process of compiling a model is done with a single mouse click on the button corresponding to this functionality. Upon completion of this process the compiler can provide a window stating the result: build successful or performed with compilation errors.

The system also provides two traffic light icons attached in both windows (positioned to the upper right corner). Their function is to indicate the status of a model: under development (red), paused (yellow) or simulating (green).

In *MyWorld*, during a simulation, the objects will change their properties according to the program being executed. As the simulation proceeds, it is also possible to see in *MyProgram* an execution pointer – red rectangle – surrounding the command that is being interpreted. This feature is especially useful for debugging purposes.

Examples

Some examples are given in order to show the implemented features and the possible use of this environment.

Figure 4 shows a simple model constructed in ProgrameFácil. This first model has only one red LED (E1). When running this model in ProgrameFácil or in RoboFácil's hardware, E1 will blink every each second.

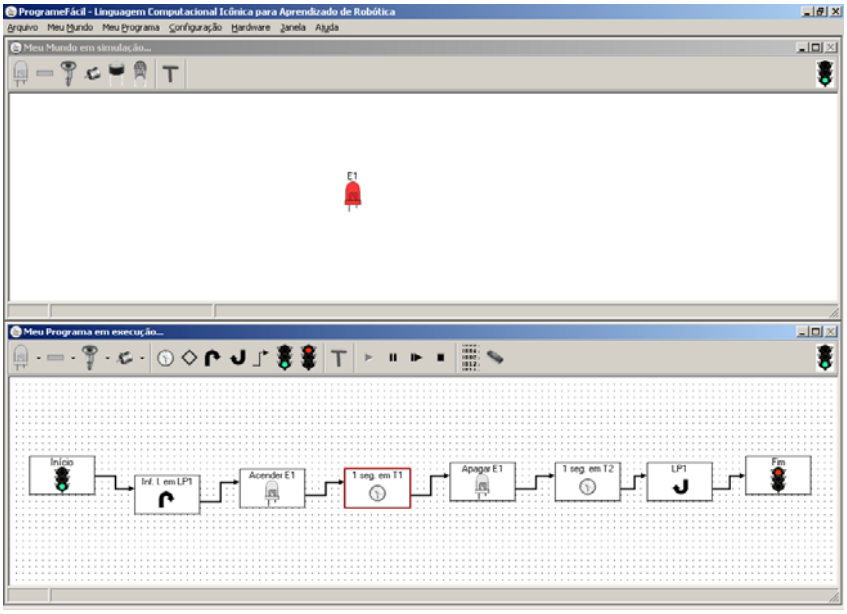

Figure 4. Example of a LED blinking

Figure 5 presents another example with three lamps – L1 (yellow), L2 (green), and L3 (red) – and also a light sensor (SL1). When running this model in ProgrameFácil or in RoboFácil's hardware, L1 will turn on and then SL1 will be tested. In the case it is sensing light around it, L2 will be turned on, otherwise L3 will be on. Note that as L1 is near SL1 (so it is sensing light), the L2 was turned on when this model was simulated in ProgrameFácil.

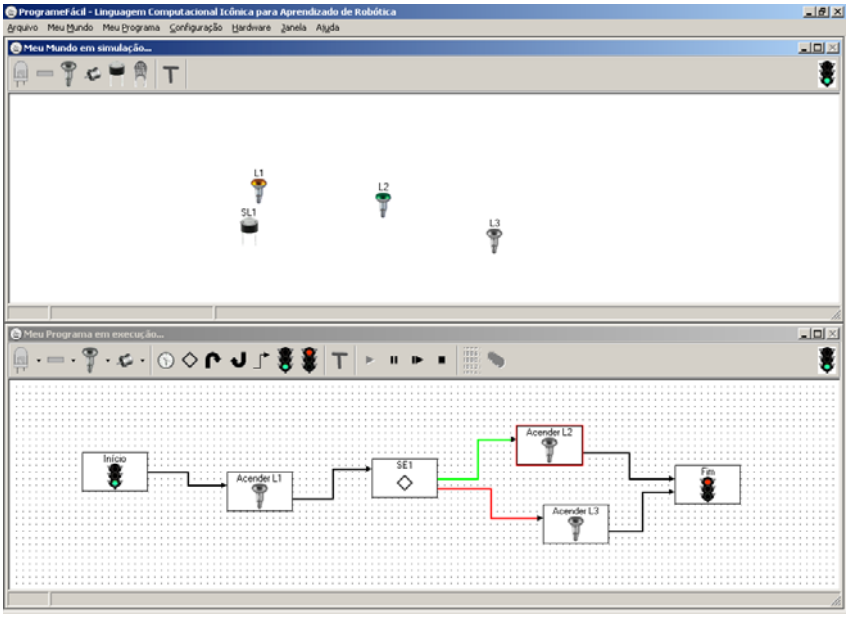

Figure 5. Example with lamps and light sensor

Figure 6 shows a third example of a model constructed with ProgrameFácil. This model has two green LEDs (E1 and E2), and also a light sensor (SL1) and an alphanumeric display with green backlight (D1). The model exemplified here aims to turn on E2, if SL1 is under the "natural" light. When running this model in ProgrameFácil, the message "No light" will appear on D1 for five seconds whenever SL1 is not under "natural" light (this was the condition when this model was simulated).

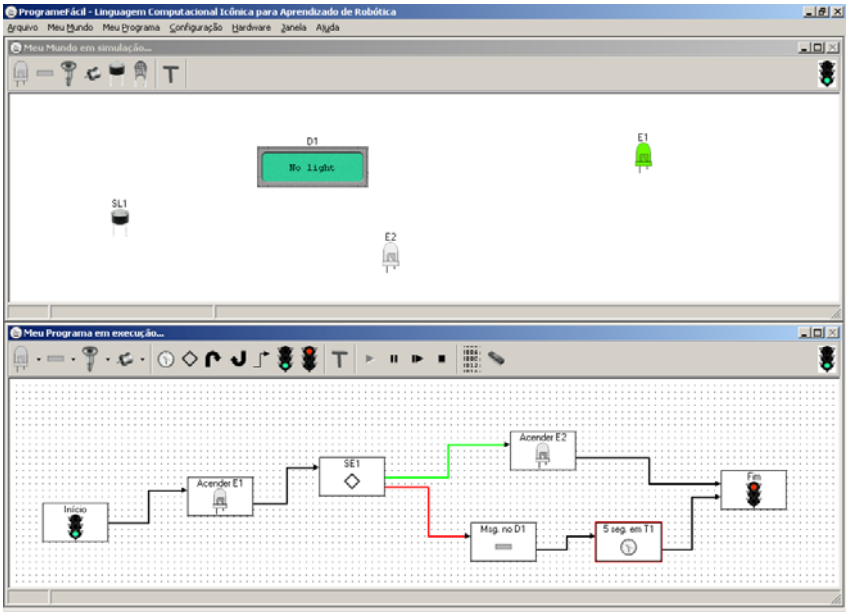

Figure 6. Example with LEDs, light sensor, and display

The last example presents a more complex situation (Figure 7). This model has one yellow lamp (L1), one temperature sensor (ST1), two motors (M1 and M2), and two displays with blue backlight (D1 and D2). In this example, when the ST1 sensor is below its trigger level, i.e., when in ProgrameFácil's model L1 is far from ST1, M1 and M2 will be switched on and this fact will be reported for user through D1 and D2.

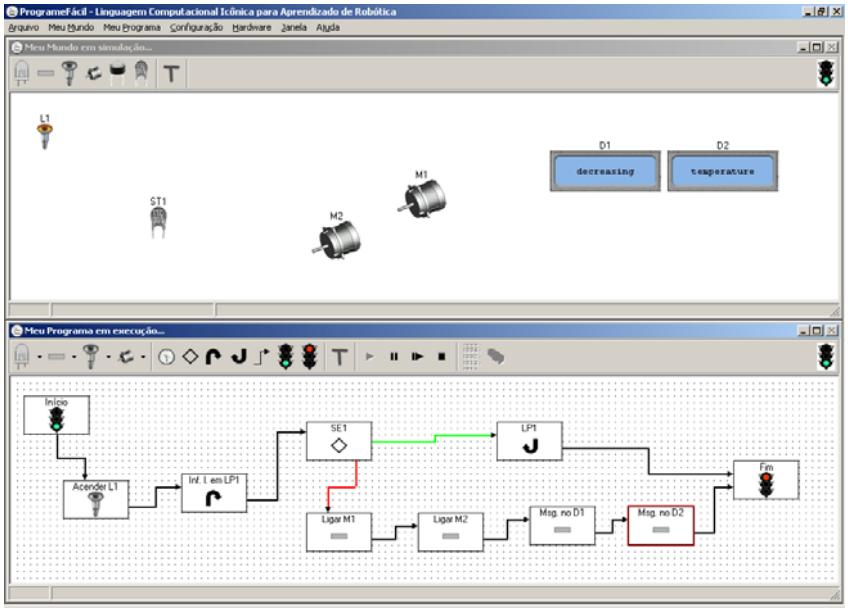

Figure 7. Example with lamp, temperature sensor, motors, and displays

Constructionist Ideas and ProgrameFácil

The ProgrameFácil environment was conceived in line with the ideas of Papert and LOGO (Papert, 1980, 1993). We believe that its graphical interface together with its iconic language allow students to learn to program robotic devices in a more enjoyable way, since before the existence of this environment the process of programming RoboFácil's hardware was done via

assembly and C languages. Moreover, the features that were incorporated into the system, as discussed earlier, enable students to focus on the process of *exploring possible solutions* instead of investing his/her time to program a certain solution.

The process of constructing a solution is made by the students through the manipulation of concrete-abstract objects in the environment. We believe that they are concrete in the sense that they resemble very much the electronics presented in the hardware and they also show on the screen most of the real properties of such hardware.

The possibility of performing a simulation on the screen – in *MyWord* window – allows the students to easily – and quickly – confront his/her ideas initially thought to solve a certain problem with the output of the – hypothetical – hardware. This feature gives them the possibility to visualize and to reflect on how each part of its solution works before downloading it to the – real – hardware.

Conclusions

Educational robotics, although not new, is not yet widespread in Brazilian schools (and possibly the same situation happens in many developing countries). A possible reason for this is the relatively high cost of the necessary hardware for many educational institutions.

The VPL presented here combines theoretical knowledge from different areas such as education, computer science and engineering to provide a feasible alternative for the high cost of robotics toolkits available in the market. Although the ProgrameFácil environment was originally designed to be used with RoboFácil's hardware it can be easily integrated with different hardware designs such as GoGo Board (2006) and Lego Mindstorms (2006).

An important feature of the ProgrameFácil environment is its built-in simulation tool. Among other advantages, this feature minimizes the need to have a specific hardware for each group of students, thereby helping to reduce costs in setting up workshops and laboratories to work with robotics. However, it is important to note that you must have a sufficient number of robotics toolkits for all students participate in the process of constructing and testing their projects.

Experiments initially conducted with undergraduate students in computer science, demonstrated the potential application of the solutions described here. These tests have allowed our research team to obtain a more practical view of the use of digital artefacts in real-world educational scenarios, giving us feedback on some improvements to be implemented.

As future work we propose to carry out pilot studies to develop thoughtful pedagogical proposals that could explore the environment presented in this work. We believe these proposals, as mentioned before, have to be anchored in constructivist ideas in other to explore better the student's potential to work with thought-provoking problems.

References

Alimisis, D., Karatrantou, A. and Tachos, N. (2005) *Technical School Students Design and Develop Robotic Gear-Based Constructions for the Transmission of Motion*. In Proceedings of the 10th European Logo Conference (EuroLogo 2005). pp. 76-86.

Alimisis, D., Moro, M., Arlegui, J., Pina, A., Frangou, S. and Papanikolaou, K. (2007) *Robotics & Constructivism in Education: The TERECoP Project*. In Proceedings of the 11th European Logo Conference (EuroLogo 2007). pp. 1-11.

Alves, A.C., Blikstein, P. and Lopes, R.D. (2005) *Robótica na Periferia? Uso de Tecnologias Digitais na Rede Pública de São Paulo como Ferramentas de Expressão e Inclusão*. In Anais do XXV Congresso da Sociedade Brasileira de Computação (CSBC 2005). pp. 2594-2602.

Chang, S.K. (1987) *Visual Languages: A Tutorial and Survey*. IEEE Software, 4(1), 29-39.

Chella, M.T. (2002) *Ambiente de Robótica para Aplicações Educacionais com SuperLogo*. Master's thesis, University of Campinas.

d'Abreu, J.V.V., Gonçalves, L.M.G., Garcia, M.F. and Garcia, L.T.S. (2002) *Uma Abordagem Prático-Pedagógica para o Ensino de Robótica em Ciência e Engenharia de Computação*. In Anais do XIII Simpósio Brasileiro de Informática na Educação (SBIE 2002). pp. 428-439.

Demo, G.B. and Marcianó, G. (2007) *Contributing to the Development of Linguistic and Logical Abilities through Robotics*. In Proceedings of the 11th European Logo Conference (EuroLogo 2007). pp. 1-10.

GoGo Board (2006) Available at: http://www.gogoboard.org.

Kouznetsova, I., Gorlitskaya, S. and Sidorov, V. (2001) *LOGO&LEGO for Informatical Training of Children with Different Educational Levels*. In Proceedings of the 8th European Logo Conference (EuroLogo 2001). pp. 137-142.

Lego Mindstorms (2006) Available at: http://www.lego.com/eng/education/mindstorms.

Miranda, L.C. (2006) *RoboFácil: Especificação e Implementação de Artefatos de Hardware e Software de Baixo Custo para um Kit de Robótica Educacional*. Master's thesis, Federal University of Rio de Janeiro.

Norte, S., Castilho, N., Condado, P.A. and Lobo, F.G. (2005) *GoGoBoard and Logo Programming for Helping People with Disabilities*. In Proceedings of the 10th European Logo Conference (EuroLogo 2005). pp. 171-178.

Papert, S. (1980) *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, New York.

Papert, S. (1993) *The Children's Machine: Rethinking School in the Age of the Computer*. Basic Books, New York.

Santos, C.F. and Menezes, C.S. (2005) *A Aprendizagem da Física no Ensino Fundamental em um Ambiente de Robótica Educacional*. In Anais do XXV Congresso da Sociedade Brasileira de Computação (CSBC 2005). pp. 2746-2753.

Silva, A.F. (2009) *RoboEduc: Uma Metodologia de Aprendizado com Robótica Educacional*. Ph.D. thesis, Federal University of Rio Grande do Norte.

Steffen, H.H. (2002) *Robótica Pedagógica na Educação: Um Recurso de Comunicação, Regulagem e Cognição*. Master's thesis, University of São Paulo.

Zilli, S.R. (2004) *A Robótica Educacional no Ensino Fundamental: Perspectivas e Prática*. Master's thesis, Federal University of Santa Catarina.