From RoboLab to Aibo: A Behavior-Based Interface for Educational Robotics

Rachel Goldman[1], M.Q. Azhar[2], and Elizabeth Sklar[3]

[1] Google, Inc.
1440 Broadway, New York, NY 10018 USA
rjg@google.com
[2] Dept of Computer Science
Graduate Center, City University of New York
365 5th Avenue, New York, NY 10016 USA
mazhar@gc.cuny.edu
[3] Dept of Computer and Information Science
Brooklyn College, City University of New York
2900 Bedford Avenue, Brooklyn, NY 11210 USA
sklar@sci.brooklyn.cuny.edu

Abstract. This paper describes a framework designed to broaden the entry-level for the use of sophisticated robots as educational platforms. The goal is to create a low-entry, high-ceiling programming environment that, through a graphical behavior-based interface, allows inexperienced users to author control programs for the Sony Aibo four-legged robot. To accomplish this end, we have extended the popular *RoboLab* application, which is a simple, icon-based programming environment originally designed to interface with the LEGO Mindstorms robot. Our extension is in the form of a set of "behavior icons" that users select within RoboLab, which are then converted to low-level commands that can be executed directly on the Aibo. Here, we present the underlying technical aspects of our system and demonstrate its feasibility for use in a classroom.

1 Introduction

Many teachers have an interest in introducing robots into their classrooms for teaching a variety of subjects other than specifically robotics, from traditional technical topics such as programming and mechanical engineering to other areas such as mathematics and physical science, where robots are used to demonstrate the concepts being taught. It has long been recognized that hands-on methods have a powerful impact on student learning and retention [1,2,3]. The growing field of *educational robotics*—the use of robots as a vehicle for teaching subjects other than specifically robotics [4]—has employed that approach, using the *constructionist* [5] process of designing, building, programming and debugging robots, as well as collaboration and teamwork, as powerful means of enlivening education [2,6,7]. Even very young children have been successfully engaged in hands-on learning experiences that expose them to some of the basic principles

G. Lakemeyer et al. (Eds.): RoboCup 2006, LNAI 4434, pp. 122–133, 2007.

of science and engineering, widening their horizons and preparing them for life in a highly automated, technically challenging world.

For the past several years, we have been designing and helping to implement educational robotics curriculum in inner-city primary and middle school classrooms, after-school programs and summer schools, undergraduate introductory programming courses and international robotic competitions [8,9,7,10]. To support these curricula, we have employed RoboLab [11], a widely used graphical programming environment developed at Tufts University, for operation on the LEGO Mindstorms Invention System robot [12]. RoboLab runs on a Mac, Windows or Unix computer. The environment is highly visual and provides a good first experience with procedural programming concepts. Entities such as motors and sensors are represented as rectangular icons on the screen, and users drag and drop them with the mouse onto a canvas to create "code". The icons are strung together using "wires", and all programs are downloaded from RoboLab onto the LEGO robot via a "communication tower", connected to the computer's USB or serial port, that transmits the program to the robot using an infra-red signal. A simple RoboLab program is illustrated in Figure 1.

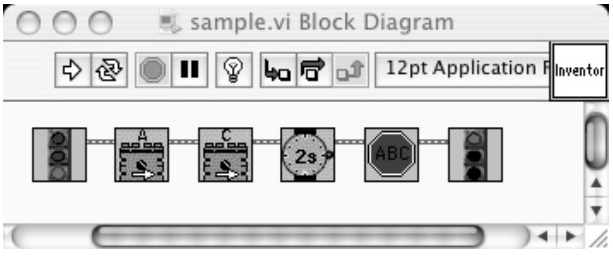


Fig. 1. A basic RoboLab program

If the robot has wheels and motors attached to two of its ports (labeled A and C), this program will make the robot go forward for 2 seconds and then stop.

RoboLab's graphical programming environment tiers the levels of programming, which allow a novice to produce results quickly and acquire skills without having to read a sophisticated text or complicated application manual. Ranging from "Pilot" to "Inventor" levels, RoboLab is broad enough to ensure the success of both beginners and advanced users. The initial Pilot levels are completely graphical so even users who cannot read can be successful. The more advanced "Inventor" levels incorporate advanced programming concepts and features to add power, flexibility and complexity to programs.

Our experiences working with classroom teachers and young students have raised several issues that have motivated us to pursue the development of a behavior-based interface, which abstracts away the low-level motor and sensor commands that often confuse inexperienced programmers or deter techno-phobic students [13]. Our longterm goal is to create a standard middle ground that can

act as a sort of "magic black box", for current and future robotic platforms, following several design criteria:

- *ease of use:* programmers only have to deal with high-level icons—novices will not get discouraged with low-level text-based syntax;
- *disappearing boundaries:* programmers are able to test and run the same behaviors on multiple agent platforms—running the same RoboLab program on a LEGO robot and on an Aibo will help students understand about abstraction and behavior-based control;
- *interoperability:* a standard behavior language is used for multiple platforms—students do not need to learn different languages in order to use a variety of robot platforms, or our simulator (currently under development [14,15]); and
- *flexibility:* students from a wide range of backgrounds and teachers with a broad range of goals can use the system effectively, accommodating different levels, curricular needs, academic subjects and physical environments for instruction.

Because of RoboLab's popularity in the classroom and its icon-based programming style, it is well suited as the front-end for our interface. We have three underlying educational goals, each supporting different pedagogical needs:

- First, we want to produce a low-entry, high-ceiling interface to the Sony Aibo robot [16] (pictured in Figure 2a) that will encourage non-traditional computer science students to learn programming, allowing us to capitalize on the "cuteness factor" associated with Aibo while still providing students with a serious, first adventure in programming.
- Second, we want advanced students to gain an appreciation of the modularity of both a robot's controlling interface and its underlying hardware. Just as Java and JavaScript are platform independent, providing a robot programming environment that is platform independent will give students hands-on experience with *code abstraction*, witnessing the same code executing on both the LEGO robot and the Aibo (and other platforms).
- Third, we want to create a motivating, hands-on environment with which to introduce more advanced students to behavior-based concepts.

This paper is organized as follows. We begin by reviewing several different programming interfaces designed for the Sony Aibo. Then we describe some technical details about RoboLab and explain how our behavior-based programmming interface operates. Finally, we end with a brief summary and mention directions for future work.

2 Background

OPEN-R is the programming interface designed for the Aibo, provided for free via download from Sony's web site [17]. OPEN-R gives programmers the ability

(a) Aibo ERS-7M2/B (b) LEGO Mindstorms

Fig. 2. Robot platforms

to develop software to control the low-level hardware of the robot. Some of the features of OPEN-R include: modularized hardware, modularized software and networking support. Because the hardware is modularized, each module is connected by a high-speed serial bus and can be exchanged for a new module. Each software module in OPEN-R is either a "data object" or a "distributed object" and is implemented in C++[1]. OPEN-R programs are built as a collection of concurrently running OPEN-R objects, which have the ability to communicate with each other via message passing. Due to the modularity of the software, individual objects can be easily replaced and each object is separately loaded from a Sony Memory Stick. Furthermore, OPEN-R supports Wireless LAN and TCP/IP network protocol.

Sony's formalism for describing the working cycle of OPEN-R objects resembles layered *finite state automata* [18]. Each OPEN-R object can have numerous states and must include an IDLE state. At any given point, an object can be in only one state, and objects move from state to state using *transitions*. In order to switch states, an *event* must activate a transition to a new state and the pre-condition of the new state must be satisfied.

The low-level functionality of the robot can be controlled using the OPEN-R API [17]. With the API, programmers can experiment with image processing, sensory feedback information and robot motion in order to develop (original) sets of behaviors for the Aibo. However, the OPEN-R API can be quite hard to use for novice and even intermediate programmers. As a result, several interfaces and abstraction languages have been developed to sit on top of OPEN-R in an attempt to hide the low-level complexity from the inexperienced end-user. These include:

- *R-CODE* [19],
- *YART* (Yet Another R-CODE Tool) [20],
- *Tekkotsu* [21], and
- *URBI* (Universal Robotic Body Interface) [22,23].

[1] C++ objects and OPEN-R objects are not the same and should not be confused with each other [18].

These languages/interfaces vary in power and intricacy, and each has its own goals and features, as described briefly below. OPEN-R is the basis upon which R-CODE, Tekkotsu, and URBI are built. YART goes one step further, abstracting R-CODE one level by providing a basic graphical user interface (GUI) to create and manipulate R-CODE programs.

The R-CODE SDK provides a set of tools that allow users to program the Aibo using the *R-CODE scripting language*, offering higher-level commands than traditional programming languages such as C, C++ and even OPEN-R. The benefits of R-CODE being a scripting language are its simplicity (to both learn and use) and its lack of compilation; however, programmers have less control than with other lower-level languages. With only a few lines of R-CODE, users can program the Aibo to perform complex behaviors such as dancing, kicking or walking. Because R-CODE does not require compilation, it can be written in a plain text file on any operating system and saved directly on a memory stick that has the R-CODE virtual machine pre-loaded on it. R-CODE commands can be viewed as OPEN-R macros, where the degree of precision and control depends solely on the underlying OPEN-R code. Although R-CODE is powered by OPEN-R functions, the developer does not have access to the underlying OPEN-R subroutines. R-CODE is best suited for performing actions and various behavior sequences.

YART [20] is an R-CODE front-end developed by a hobbyist. It provides a text-based GUI with simple drag-and-drop functionality and produces files of R-CODE commands. This tool can also be used to generate customized behaviors via pre-existing YART-compatible Aibo personalities. YART is an easy place to start programming simple behavior patterns. However, unlike RoboLab, YART is a text-based interface, so the objects that the user drags with the mouse are bits of text—whereas in RoboLab, the user drags graphical icons.

Tekkotsu [21], developed at Carnegie Mellon University by Touretzky et al., is an application development framework for intelligent robots that is compatible with the OPEN-R framework [17]. It is based on an object-oriented and event-passing architecture that utilizes some of the key features of C++, like templates and inheritance. Although Tekkotsu was originally created for the Sony Aibo, the current version can be compiled for multiple operating systems. Tekkotsu simplifies robotic application development by supplying basic visual processing, forward and inverse kinematics solvers, remote monitoring, teleoperation tools, and wireless networking support. Tekkotsu handles low-level tasks so the developer can focus on higher-level programming. It provides primitives for sensory processing, smooth control effectors, and event-based communication. Some of the higher-level features include a hierarchical state machine formalism used for control flow management and an automatically maintained world map. Additionally, Tekkotsu includes various housekeeping and utility functions and tools to monitor various aspects of the robot's state.

The Universal Robotics Body Interface (URBI) [22,23], developed at École Nationale Supérieure de Techniques Avancées (ENSTA), is an attempt to provide

a standard way to control the low-level aspects of robots while providing the high-level capabilities of traditional programming languages. URBI is based on a client-server architecture where the server is running on the robot and is typically accessed by the client via TCP/IP. The client can be virtually any system or any other kind of computer, thereby adding flexibility to URBI. The URBI language is a high-level scripting language capable of controlling the joints and accessing the sensors, camera, speakers or other hardware on the robot. URBI has been primarily applied to entertainment robots because they tend to provide the most interesting interfaces and capabilities.

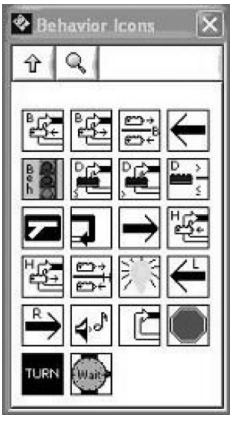
Each of these interfaces have informed our approach. We selected RoboLab as the front-end for several reasons. RoboLab is already quite widely used in classrooms worldwide. Teachers and students are comfortable with the interface and will not feel like they need to learn yet another programming environment in order to expand their use of robot platforms in the classroom. In addition, schools do not need to purchase another software package; our methodology is a free extension to RoboLab. Finally, as detailed below, RoboLab is constructed on top of a framework designed to support a wide range of hardware devices, thus the concept of expanding its use to interface with a range of robot devices is a natural fit.

3 Our Approach

This section describes the approach to our implementation, outlining the steps required for progressing from writing programs in RoboLab to generating code that is executed on the Aibo. Given that one of our objectives with this work is to develop a behavior-based programming interface for controlling Aibo, we have designed a set of generic low-level behaviors that can be graphically represented in RoboLab. As part of this process, we analyzed the main differences between the Aibo platform and the LEGO Mindstorms platform to ensure that our behaviors are suitable for both. The two platforms are physically quite different, not only in terms of processor configuration but also in regard to the types of sensors and effectors provided. The LEGO Mindstorms is typically constructed as a wheeled robot, as depicted in Figure 2b (though legged structures can be built). The kit comes with two touch sensors and a light sensor and has the ability to support numerous other LEGO and commercial sensors. The Aibo, a four-legged robot, comes with a variety of built-in sensors including: multiple touch sensors, distance sensors and a camera; and it cannot support any other sensors (without being dismantled).

With the capabilities of both platforms in mind, we defined a set of prototype behaviors and control structures suitable for our behavior-based "palette" in RoboLab (illustrated in Figure 3). RoboLab is implemented on top of National Instruments' LabVIEW [24]. Individual icons and full programs are saved as "VIs" (virtual instruments). Each icon or program can be seen as imitating an actual instrument [25]. We used RoboLab's built-in feature to create "subVIs" (VI modules or subroutines) in order to construct our customized behaviors.

These behaviors act as macros for sets of lower-level RoboLab commands. Although the ability to expand RoboLab's current set of icons is a remarkably powerful feature, we are constrained by the necessity to use a set of pre-defined icons as the underlying basis for each new icon. For example, as illustrated in Figure 4, our forward behavior icon is in reality a macro for the set of icons: motor A forward, motor C forward, and wait for (some amount of time).



The behaviors icons, shown in the palette to the left, can be used just like any other RoboLab icon. As with basic RoboLab function icons, wiring together a sequence of behavior icons creates a well-formed RoboLab program. The meanings of the first eight icons are (from top left to right): loop while back sensors are pressed, loop while back sensors are not pressed, branch according to state of back sensor, move backwards, begin behavior, loop while distance sensor is less than or equal to parameter, loop while distance sensor is greater than parameter, branch according to state of distance sensor. See [13] for a complete and detailed description of the behavior icons.

Fig. 3. Behavior Icon Palette

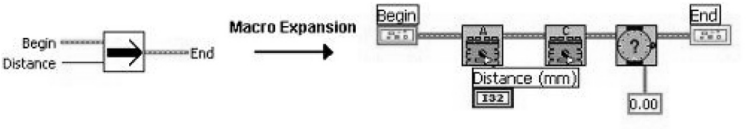


Fig. 4. Forward behavior icon and its underlying set of basic RoboLab icons

The first step in our approach is for a user to write a program in RoboLab and save the contents of the program to a file, in a manner that will preserve its functionality while allowing translation to Aibo commands to occur outside of RoboLab. The default output from RoboLab is *LASM*, or LEGO Assembly Language. We use this default, saving the LASM commands in an output file — whereas normally, users send the LASM commands directly to the LEGO robot via the communication tower. Once the LASM file is saved, we invoke our translation program, called `lasm2aibo`, that converts the LASM commands into Aibo commands. After examining the Aibo languages and interfaces discussed in section 2, we decided to use R-CODE to implement Aibo commands in our system because R-CODE is easy to use and does not require compilation, and because R-CODE offers a more natural mapping to RoboLab. With R-CODE,numerous behaviors can be prototyped quickly and tested efficiently.

Both Tekkotsu and URBI are better suited for applications that require complex solutions and greater computational power.

A key challenge in designing `lasm2aibo` was to determine which LASM command(s) and what parameters are generated for each RoboLab icon. Our behavior icons are macros comprised of multiple low-level built-in RoboLab icons, which complicates the translation process, as detailed below. Taking a file of LASM commands as input, our translater recognizes tokens that match relevant LASM commands, numbers, white space, and delineators (commas and new line characters), and "compiles" (or translates) these into R-CODE sequences. We designed and implemented our translator using the UNIX tools Lex [26] and Yacc [27].

Both Lex and Yacc greatly simplify compiler writing, or translating between two programming language representations. Lex generates the C code for building a lexical analyzer or lexer. Any given lexer takes an arbitrary input stream and divides it into a sequence of tokens based on a set of regular expression patterns. The Lex specification refers to the set of regular expressions that Lex matches against the input. A deterministic finite state automaton generated by Lex performs the recognition of the expressions. Lex allows for ambiguous specifications and will always choose the longest match at each input point. Each time one of the patterns is matched, the lexer invokes user-specified C code to perform some action with the matched token. Yacc is responsible for generating C code for a syntax analyzer or a parser. Yacc uses the grammar rules to recognize syntactically valid inputs and to create a syntax tree from the corresponding lexer tokens. A syntax tree imposes a hierarchical structure on tokens by taking into account elements such as operator precedence and associativity. When one of the rules has been recognized, then the user-provided code for this rule (an "action") is invoked. Yacc generates a bottom-up parser based on shift-reduce parsing. When there is a conflict, Yacc has a set of default actions. For a shift-reduce conflict, Yacc will shift. For reduce-reduce conflicts, Yacc will use the earlier rule in the specification.

In our case, Yacc reads the grammar description and the token declarations from `lasm2aibo.y` and generates a parser function `yyparse()` in the file `y.tab.c`. Running Yacc with the `-d` option causes Yacc to generate definitions for the tokens in the file `y.tab.h`. Lex generates a lexical analyzer function `yylex()` in the file `lex.yy.c` by reading the pattern descriptions from `lasm2aibo.l` and including the header file `y.tab.h`. The lexer and parser are compiled and linked together to form the executable `lasm2aibo`. From the `main()` function in `lasm2aibo`, the `yyparse()` function is called, which in turn calls the `yylex()` function to obtain each token.

Our Yacc parser is generated from the `lasm2aibo` grammar specification. Through a series of grammar productions, the parser attempts to group sequences of tokens into syntactically correct statements. Associated with each production is a set of semantic actions. Given that many of the LASM command sequences for varying groups of behaviors are the same, often behavior

identification becomes part of the semantic analysis. For instance, all the motion behaviors including forward, backward, right and left produce the same LASM command output. Furthermore, there is no direct way to distinguish between activating an LED or a motor because they are both viewed as output devices. Only by examining the values of some of the parameters and by making certain assumptions can they be differentiated. Having to perform specific behavior recognition during the semantic analysis is one of the limitations involved in using LASM as our source program. Ideally, each behavior should be represented by a unique syntax, allowing for quicker and cleaner parsing.

When a behavior is recognized, the semantic actions involve calling the corresponding behavior function. These pre-defined behavior functions are used to generate the equivalent behavior in R-CODE and help flag the R-CODE subroutines that need to be included in the R-CODE source file. The final output of the translator includes the file `R-CODE.R`, the R-CODE program that matches the original RoboLab program, and a `behaviors.txt` file used for debugging purposes to ensure that behaviors, sensors, and control structures are appropriately classified. Running the `lasm2aibo` executable file with a LASM text file as input generates the file `R-CODE.R`. This file should be placed on a R-CODE-ready memory stick in the `OPEN-R/APP/PC/AMS` directory [28]. To execute the program, insert the memory into Aibo, turn on the power and watch Aibo come to life!

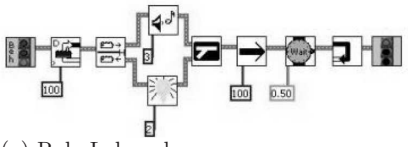
Throughout the process, there are multiple places where errors could occur; this includes lexical, syntactic and semantic errors. The `lasm2aibo` translator attempts to handle errors gracefully. If the error is fatal, a blank `R-CODE.R` file will be generated; however, if the error is non-fatal then the translator will continue to process the source file. In both cases, a descriptive error message and the line number where the error occurred is displayed on the console via standard output.

3.1 Example

A detailed example follows. First, the RoboLab behavior-based program (shown in Figure 5a) is constructed. The resulting LASM is contained in Figure 5b. Second, our `lasm2aibo` module is executed, taking the LASM file as input and creating an R-CODE equivalent, illustrated in Figure 5c. This gets written to a Sony memory stick, which also contains the R-CODE virtual environment, and is ready to be executed on the Aibo.

4 Summary

We have presented the design and implementation of our prototype framework for providing a simple, graphical, behavior-based interface to the Sony Aibo robot. Built into the popular RoboLab graphical programming environment and connected to the Aibo via a translater, `lasm2aibo`, this framework represents the proof-of-concept for a longterm project aimed at bringing educational robotics into a broad range of classrooms. The successful implementation of the direct



(a) RoboLab code

```
delt 0

task 0
sent 0,3
senm 0,4,0
sent 1,1
senm 1,1,0
Label1002:
chkl 2,100,1,9,0,Label1003
chkl 2,1,0,9,1,Label1004
plays 2
wait 2,100
wait 2,0
jmpl Label1005
Label1004:
pwr 2,2,2
dir 0,2
out 2,2
wait 2,0
Label1005:
pwr 1,2,100
dir 2,1
out 2,1
pwr 4,2,7
dir 2,4
out 2,4
wait 2,0
wait 2,50
ping
jmpl Label1002
Label1003:
endt

plays 5
```

(b) LASM code

```
:Start
PLAY:ACTION:STAND
WAIT
WHILE:100:<:Distance
IF:1:>:Head_ON:THEN
PUSH:2471
CALL:Play_Sound:1
PUSH:1000
CALL:Wait_For_Time:1
WAIT
ELSE
PUSH:2260
CALL:Display_LED:1
WAIT
ENDIF
CLR:SENSORS
PUSH:100
CALL:Forward:1
WAIT
PUSH:500
CALL:Wait_For_Time:1
WEND
CLR:SENSORS
EXIT

*************************************************
*                  BEHAVIOR FUNCTIONS           *
*************************************************

/*********************************************
 * walk forward for a specified distance in mm
 *********************************************/
 :Forward //pass a distance
  ARG:distance
  PLAY:ACTION:WALK:0:distance
  RETURN

/*********************************************
 * wait for a specified amount of time
 *********************************************/
 :Wait_For_Time
  ARG:time
  WAIT:time
  RETURN

/*********************************************
 * plays a specified sound
 *********************************************/
 :Play_Sound
  ARG:sound
  PLAY:MWCID:sound
  RETURN

/*********************************************
 * plays a specified led pattern
 *********************************************/
 :Display_LED
  ARG:pattern
  PLAY:MWCID:pattern
  RETURN
```

(c) R-CODE

Fig. 5. Example

This program will test whether the head sensor is pressed or not. If the head sensor is released, it will play sound number 3, otherwise if the head sensor is pressed it will display LED pattern number 2. The Aibo will then move forward 100mm and wait half asecond. This process will repeat while the distance sensor reads a value greater than100mm.

translation from RoboLab to Aibo demonstrates the feasibility and viability of the process. A user study is planned for Summer 2006.

Although our direct translation is appropriate for small-scale solutions, current work on this project is exploring a more abstract, generalized framework for linking RoboLab (or other graphical programming interfaces) to a variety of robot platforms [15]. Recent press releases have revealed that the LEGO Mindstorms will be succeeded in August 2006 by a more sophisticated platform called NXT [29], and Sony has announced that production of Aibo halted in March 2006 [30]. Given the changing face of consumer robotics, a flexible framework such as ours will help classrooms ease transitions from one robot platform to another by providing teachers and students with a familiar interface and an intuitive behavior-based methodology for programming, no matter what hardware lies beneath.

References

1. Piaget, J.: To Understand Is To Invent. The Viking Press, Inc., New York (1972)
2. Papert, S.: Mindstorms: Children, Computers, and Powerful Ideas. BasicBooks (1980)
3. Resnick, M.: Technologies for lifelong kindergarten. Educational Technology Research and Development 46(4) (1998)
4. Sklar, E., Parsons, S.: RoboCupJunior: a vehicle for enhancing technical literacy. In: Proceedings of the AAAI-02 Mobile Robot Workshop (2002)
5. Papert, S.: Situating constructionism. Constructionism (1991)
6. Slavin, R.: When and why does cooperative learning increase achievement? theoretical and empirical perspectives. In: Hertz-Lazarowitz, R., Miller, N. (eds.) Interaction in cooperative groups: The theoretical anatomy of group learning, pp. 145–173. Cambridge University Press, Cambridge (1992)
7. Sklar, E., Eguchi, A., Johnson, J.: RoboCupJunior: Learning with Educational Robotics. In: Kaminka, G.A., Lima, P.U., Rojas, R. (eds.) RoboCup 2002. LNCS (LNAI), vol. 2752, pp. 238–253. Springer, Heidelberg (2002)
8. Goldman, R., Eguchi, A., Sklar, E.: Using educational robotics to engage inner-city students with technology. In: Kafai, Y., Sandoval, W., Enyedy, N., Nixon, A.S., Herrera, F. (eds.) Proceedings of the Sixth International Conference of the Learning Sciences (ICLS), pp. 214–221 (2004)
9. Sklar, E., Eguchi, A.: RoboCupJunior – Four Year Later. In: Nardi, D., Riedmiller, M., Sammut, C., Santos-Victor, J. (eds.) RoboCup 2004. LNCS (LNAI), vol. 3276, Springer, Heidelberg (2004)
10. Sklar, E., Parsons, S., Stone, P.: RoboCup in Higher Education: A preliminary report. In: Polani, D., Browning, B., Bonarini, A., Yoshida, K. (eds.) RoboCup 2003. LNCS (LNAI), vol. 3020, Springer, Heidelberg (2004)
11. Tufts University: RoboLab (accessed January 16, 2006), `http://www.ceeo.tufts.edu/robolabatceeo/`
12. LEGO: Mindstorms robotics invention kit (accessed February 1, 2006), `http://www.legomindstorms.com/`
13. Goldman, R.: From RoboLab to Aibo: Capturing Agent Behavior. Master's thesis, Department of Computer Science, Columbia University (2005)

14. Chu, K.H., Goldman, R., Sklar, E.: Roboxap: an agent-based educational robotics simulator. In: Agent-based Systems for Human Learning Workshop at AAMAS-2005 (2005)
15. Azhar, M.Q., Goldman, R., Sklar, E.: An agent-oriented behavior-based interface framework for educationa robotics. In: Agent-Based Systems for Human Learning (ABSHL) Workshop at Autonomous Agents and MultiAgent Systems (AAMAS-2006) (2006)
16. Sony: AIBO (accessed January 16, 2006), http://www.us.aibo.com/
17. OPEN-R: SDE (accessed January 16, 2006), http://openr.aibo.com/
18. Serra, F., Baillie, J.C.: Aibo Programming Using OPEN-R SDK Tutorial (2003), http://www.cs.lth.se/DAT125/docs/tutorial_OPENR_ENSTA-1.0.pdf
19. R-CODE: SDK (accessed January 16, 2006), http://openr.aibo.com/openr/eng/no_perm/faq_rcode.php4
20. YART: Yet Another R-CODE Tool (accessed January 16, 2006), http://www.aibohack.com/rcode/yart.htm
21. Touretzky, D.S., Tira-Thompson, E.J.: Tekkotsu: A framework for AIBO cognitive robotics. In: Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05), Menlo Park, CA, AAAI Press, Stanford (2005)
22. Baille, J.C.: URBI: Towards A Universal Robotic Body Interface. In: Proceedings of the IEEE/RSJ International Conference on Humanoid Robots, Santa Monica, CA USA (2004)
23. Baille, J.C.: URBI: Towards a Universal Robotic Low-Level Programming Language. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Edmonton, Canada (2005)
24. National Instruments: LabVIEW (accessed January 16, 2006), http://www.ni.com/labview/
25. LabVIEW: User Manual (accessed January 16, 2006), http://www.ni.com/pdf/manuals/320999e.pdf
26. Lesk, M.E., Schmidt, E.: Lex – A Lexical Analysis Generator. Bell Laboratories, Murray Hill, NJ (1975)
27. Johnson, S.C.: Yacc – Yet Another Compiler-Compiler. Bell Laboratories, Murray Hill, NJ (1975)
28. Téllez, R.: R-CODE SDK Tutorial (v1.2) (2004)
29. LEGO: What's NXT? LEGO Group Unveils LEGO MINDSTORMS NXT Robotics Toolset at Consumer Electronics Show (January 4, 2006), http://www.lego.com/eng/info/default.asp?page=pressdetail&contentid=17278&countrycode=2057&yearcode=&archive=false&bhcp=1
30. Duffy, J.: What happened to the Robot Age? BBC News Magazine (January 27, 2006)