What is the Best Environment-Language for Teaching Robotics Using Lego MindStorms?

Anthony Hirst[1], Jeffrey Johnson[2], Marian Petre[3], Blaine A. Price[3], Mike Richards[3]
Departments of Telematics[1], Design and Innovation[2], and Computing[3]
The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK
robofesta@open.ac.uk

Abstract

We aim to produce a range of educational materials to teach robotics to a variety of audiences using the LEGO Mindstorms Robotics Invention System™. We briefly review the programming environments currently available and consider their appropriateness for our candidate audiences. There is the usual trade-off between ease of use and power. It is suggested that no single programming environment is suitable for all audiences. Instead, a *progression* of environments from microworlds, through graphical programming environments, to textual languages seems to provide the best way to develop our teaching. In this paper we synthesise our thoughts, and present them for constructive criticism by the robotics community.

1. Introduction

Robotics has been shown by a number of researchers to be motivating and beneficial in teaching science and technology (Beer *et al*., 1999). We believe that robots are a powerful way to motivate learning. The construction and programming of robots uses a wide range of scientific and engineering principles – key skills in the modern technological economy (Wasserman, 2002). This range of skills necessitates teamwork, planning and record keeping.

We have taught subjects related to robotics for many years, and we are beginning to formulate a new robotics curriculum. In collaboration with the international RoboFesta[1] and RoboCup[2] movements, we plan a large programme to teach robotics in schools as well as in our university. Previous experience with Lego-based teaching materials has made us well disposed toward the Lego MindStorms[3] Robotics Invention System™ as a possible hardware platform for robotics, engineering, and computing courses at first, second, and third levels. The inevitable question is:

What are the best environment and language for teaching robotics using Lego MindStorms?

[1] www.robofesta.net, www.robofesta-uk.org
[2] www.robocup.org
[3] http://mindstorms.lego.com

Given the depth and breadth of things that we intend to teach using MindStorms, from simple programming to engineering principles and simulation; and given the range of audiences we intend to serve, from young children to mature university students, the language issue is both complex and crucial. Because the large-scale production of good quality teaching materials is expensive, the issue has economic as well as pedagogic ramifications.

In this paper, we are not concerned with the division between environment and language, and we give both the terms language and environment a wide interpretation. For example, we treat a drop-and-drag environment for creating code as a 'language' in the same way a conventional textual language within an editing environment.

This paper is a synthesis of our research and analysis to date. We do not attempt to give a definitive answer to the question at this stage, and we invite readers to contribute to the discourse.

2. What are we teaching to whom and why?

There is currently a widespread appeal of robotics to adults and children of both sexes. This is evident in the success of television programmes featuring robots, and the growing number of robot competitions. We have broad educational aspirations, and would like to harness the interest and enthusiasm of all groups in this audience for wider educational purposes. The programming environment-language choice must accommodate those we are teaching, what we are trying to teach them, and our deeper educational aims.

2.1 To whom are we trying to teach?

- young children, less than ten years
- school children, 10 – 18 years
- university students, 18+
- adults – life-long learning

- teachers, learning to support students

The breadth of this list complicates the choice of environment and language. Although, we assume that some students will commence our courses as novices to robotics, the assumptions we can make about existing skills, speed of learning, and appropriate conceptual level will differ among groups. The needs of newly literate children are different from those of highly literate university students, which are different again from the needs of mature students returning to education. This suggests that there is no one perfect programming environment. Our goal must be pragmatic: to serve as many students as possible while making the best use of our resources.

2.2 What are we trying to teach and why?

Our plan is twofold:

- to teach robotics *per se*;
- to use robotics as a springboard to further to motivate learning.

Robotics itself is multi-disciplinary, encompassing subjects such as mechanical engineering, electronics, control, communication, vision, real-time parallel computing, and systems design. All these are relevant in our teaching.

Robotics is also a vehicle for developing key skills (e.g., teamwork, critical thinking, planning, scientific observation and record keeping); for reinforcing skills in elementary physics, mathematics, and numeracy; and for introducing advanced concepts in simulation, Artificial Intelligence (AI), and cognition.

Furthermore, robots raise profound questions about our relationship with advanced technologies and their potential that allow us to address ethical and social issues surrounding technology use.

2.3 Using robots to bridge between concept and practice

Traditional methods of teaching computing tend to be abstract, and students often have difficulty reasoning about program behaviour and recognising the relevance of their activities. The trouble is that general-purpose languages are complex, in order to afford necessary richness to the programmer. Unfortunately for the novice, this often means: '*you need to know a lot to do a little*'.

Many languages require the users to type in a large amount of code to produce relatively trivial results. Either students have to learn the syntax before they can write any programs (which is frustrating), or they have to enter code that is effectively meaningless to them. An alternative approach is to use a graphical programming environment.

Programming with robots using a tailored environment that provide strong visual cues and supports syntactic correctness:

- is concrete: students program things they can handle, to behave in ways they can observe in the physical world
- is incremental
- is creative
- admits many solutions
- allows manipulation within a constrained context
- provides immediate feedback
- has behaviour (and thus encourages anthropomorphisation)
- uses a variety of skills
- allows complete novices to create interesting outcomes (e.g*., "go collect a tennis ball"* rather than "print *'Hello, world.'*)

Our experience so far is that programming with robots helps learners to bridge between concept and practice – and to derive principles for themselves from their own experience.

2.4 Robots are appealing

The appeal of robots is evident in the success of television programmes featuring robots, such as *RobotWars* and *TechnoGames* in the UK, that attract large audiences across a wide range of ages. For over 75 years robots have been a staple of popular culture. Recent films such Steven Spielberg's *A.I.* have stimulated popular debate about the potential of robotics, and the debut of the Sony AIBO has attracted substantial media attention. Competitions involving robots are popular with participants and audiences alike. Robots are attractive to adults and children of both sexes.

2.5 How will students study what we teach?

- supported distance learning
- classroom lesson
- self-help group
- independent exploration

Multiple disciplines, multiple audiences, multiple learning modes – all of these mean that our choice of programming environment is sufficiently complex that there is unlikely to be a single solution. Instead, we might ask: What is the best *progression* of environments and languages for teaching Robotics using Lego MindStorms?

3. The System Context

3.1 The RCX Brick

Programming the MindStorms processor *brick* requires a standalone computer where code is composed, edited and compiled. The compiled code is downloaded to the brick where it executes using a small operating system implemented as the brick's *firmware* (Fig. 1).

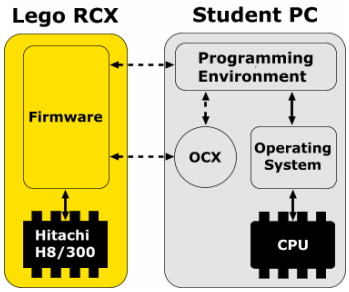

Figure 1. system concept

MindStorms is shipped with three integrated software components:

- **firmware** that can be downloaded to the micro-controller at the heart of the brick; this firmware implements a virtual machine that will run bytecode downloaded from a host machine[4];
- **an ActiveX control** (the Spirit OCX) that can be used as component-ware on an external host machine to write programs that can be downloaded to run on the brick, as well as sending direct commands to the brick running the Lego firmware[5].
- **a graphical programming language-environment** (RCX code) that uses a Lego block metaphor to construct programs out of small functional units.

For the educational market, Lego produce a more flexible environment (RoboLab) in which a wider range of functional units can be wired together rather than plugged together. The programming environments runs on the user's computer. They allow users to compose, edit, and compile code which they then download to the brick to run on the firmware. The firmware shipped with the brick imposes limitations on the types of commands that may be executed and on the number of variables, but replacement firmware can be downloaded to provide different functionality. Hence, choosing a particular programming environment may require downloading new firmware.

3.2 Hardware and operating system choice

The OU specifies the so-called Wintel machine for its students. For better or worse, this policy is based primarily and pragmatically on the fact that some ninety percent of our students have this hardware-software platform, and it is easier to support a single platform from a generic helpdesk servicing hundreds of thousands of students world-wide.

Given this hardware default, the operating system is virtually a *fait accompli*. The obvious contenders are variants of Windows and Unix (Linux or Macintosh System X). The OU's commitment to being as *open* and *inclusive* as possible contradicts a one-platform approach. Therefore, a language solution that is platform or OS 'agnostic', such as Java, would receive special consideration**.**

4. Choosing a programming environment

Our experience in teaching computing (Griffiths et al., 1999, Woodman et al., 1998), and the current trends in software engineering and AI, give us some general guidance in terms of desirable characteristics for programming environments/languages.
An object-based approach would support and integrate with our existing curriculum and is now considered the basis of sound software engineering. Object Oriented programming also makes it easy to represent and present complex behaviours to novices (Griffiths et al., 1999).

We emphasise the importance of providing software suitable for novices. Any programming environment for novices must be robust – it should behave reliably and consistently, and it must not crash. Errors (if they appear at all), must be meaningful.

The human-computer interaction, end-user programming, and visual programming

[4] A disassembly of the firmware is available at http://graphics.stanford.edu/~kekoa/rcx/.
[5] Technical documentation released by Lego as SDK1 (http://www.legomindstorms.com/sdk/) describes the functions provided by the Spirit OCX.

literatures give us some guidance about relevant concepts in language selection, as follows.

4.1 Separation of domain manipulation from programming *per se*.

Microworlds are an educational tool originally developed by the MIT Logo Group that allow students to explore and manipulate a domain in a controlled way (Pappert, 1980). The user can manipulate data or phenomena in the microworld through GUI devices such as push buttons and fill-out boxes and see the subsequent changes reflected on the screen.

In effect, users are 'programming' the microworld – albeit only to the extent of combining operations and manipulating program parameters – but the syntax and structure of the language are hidden under the interface. Hence the implementation is hidden, and users can concentrate on the domain concepts, independent of the implementation language. Moreover, users can learn fundamental programming concepts that generalise across languages without having to learn language syntax (cf. Soloway's (1986) environment designed to allow high school students to program by combining conceptual units or 'plans' rather than in a programming language.)

The sorts of concepts that can be learned from such an environment include:

- that algorithms can be used to solve problems
- that solutions can be decomposed into relatively small components
- that most tasks can be accomplished by using sequence, iteration, choice
- object concepts

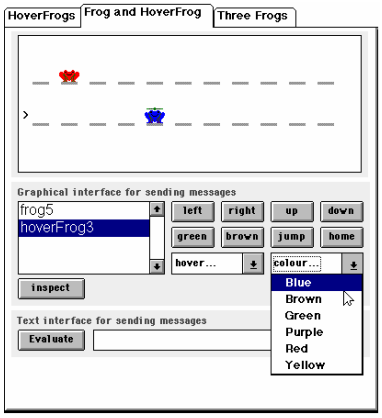

Figure 1: frogWorld. A microworld used to teach Open University students about objects, messages and inheritance

Microworlds have been used on the entry-level Open University course *Computing: An Object-Oriented Approach* to teach the concepts behind object-oriented (OO) technology. In an early example, the students are able to send messages to an on-screen frog - telling it to hop left, right and up and down, setting its colour, and so on. In later lessons they create subclasses of frogs with some inherited properties and some novel properties particular to the subclass.

4.2 Simulation: separation of control logic from physical control

Simulation is a method commonplace in the field of autonomous mobile robots for working out and testing control strategies in isolation from the physical system.

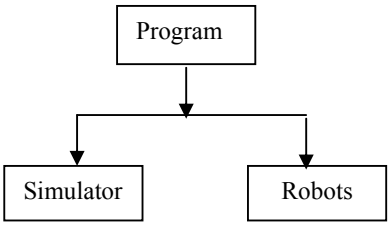

Figure 2: Ideally, the same program can drive

Figure 2 illustrates the ideal in which the same program drives both the simulator and the robots. Although simulations are often different from real systems, simulators allow ideas to be tested, and they are good for detecting bugs when the vagaries of real machines in real environments are not present. This is pertinent to MindStorms where the performance of individual sensors and motors may vary. The effects of physical variation can be addressed when the logic of the program and its implementation are correct.

Although a various of RCX simulators are available, we do not feel that they are stable enough for student use at the current time.

4.3 Direct manipulation

An important characteristic of the microworlds approach is the direct manipulation of screen objects, without imposition of linguistic devices or explicit syntax. Hutchins, Hollan, and Norman (1986) attribute to direct manipulation that novices can learn basic functionality quickly, experts can work extremely rapidly to achieve complex ends, and users can see immediately if their actions are furthering goals. Hence, direct manipulation is seen as highly desirable, characterised by the provision of rapid, incremental, reversible operations whose impact on the object of interest is immediately evident (Shneiderman, 1982).

4.4 Layering, progressive disclosure

A generalisation of the microworlds approach are the 'direct manipulation programming environments' (e.g., The Alternative Reality Kit: Smith, 1987; LabView, http://www.natinst.com/labview) which provide both a domain-level representation (e.g., a microworld or a control surface) and an underlying code representation. A key advantage of layering is that it is possible for the user to build their conceptual model through interaction with the microworld (i.e., in a controlled environment), and hence not get near the underlying syntax until they have a well-established model of the domain.

This sort of 'layered' approach, providing a gradual revelation of functionality so that the user can have the simplest environment that meets immediate needs but expose more functionality as needed has long been espoused (Carroll and Carrithers, 1984; Carroll, 1987). It has been incorporated into some of the most effective programming environments for novices and young users, such as Repenning's AgentSheets® (Repenning, 2000), a system which also allows users to move from a simple, accessible graphical environment to a textual environment when more sophistication and precision is required.

In fact, AgentSheets was used to create a rule based programming environment - LEGOsheets - for a forerunner to the Lego RCX brick, MIT's Programmable brick; as far as the authors are aware, a version of LEGOsheets has not been produced for the RCX brick (Grindling, *et al*, 1995).

Layering is also supported to a limited extent by the RCX SDK2, which introduced the Mindscript language. From the SDK2 release notes, the intention behind this language was to allow users to see a script language version of the programme produced using the graphical RCX language.

Students using our 'frogWorld' are only introduced to the implementation language (in this case Smalltalk) after fully exploring the microworld. By then, they should have a firm grounding in the concepts and can see how they are applied in a more conventional programming interface (Griffiths et al., 1999).

4.5 Readership

Graphical environments are seen as accessible and fun, and direct manipulation potentially reduces the need for text generation, which may be problematic for newly literate children. Yet graphical environments have associated issues of readership (Petre, 1995), such as:

- significant limits on the number of elements that fit on a screen;
- discriminability of graphical elements
- the need to develop effective reading or inspection strategies
- the difficulty of indexing into the code, of searching for and identifying desired graphical entities;
- scalability;
- the importance of an effective graphical editor.

5. Criteria for choice

We derived a list of criteria for language selection. Our primary concern has been an entry-level university course. However, we also wish to re-use materials for use in schools and to support students in competitions such as RoboFesta and RoboCup. Hence, the detailed decisions refer to university level, but the higher level decisions (e.g., OO, layering, multi-mode environments) are meant to generalise across our diverse audience.

Relevant criteria for selecting a language include:

- ease of understanding and use (and suitability for novices)
- rapid development
- scalability (from simple programs to complex systems)

- general-purpose programming
- convenient control of physical devices
- robustness
- support for maintenance
- cost
- compatibility with existing course and curriculum decisions
- ease and cost of updating
- longevity

5.1 Comparison of RCX Programming Environments

From its first release, Lego MindStorms proved very popular with the technically sophisticated hobbyist community. Faced with the limited power of the standard RCX

Summary table							○ Partially applicable ● Applicable	
Package	**Language type**	**Spirit OCX**	**LEGO fw**	**Novice**	**Low cost**	**CS**	**Power**	**Development environment**
RCX language	Custom graphical (Lego)	●	●	●	●			Drag-and-drop, plug together program blocks
Robolab	Custom graphical (Labview)	●	●	●		○	●	Drag-and-drop, wire together program blocks, supports communication between bricks
MindScript	Script language	●	●		●		●	Text editor
LASM	Byte-code	●	●		●			Text editor
Brick Command	Spirit OCX commands	●	●		●			Syntax checking text editor
Gordon's Brick Programmer	Spirit OCX commands	●	●		●			Drag-and-drop editor
BotCode	Resembles Spirit OCX commands	●	●		●			Syntax checking text editor
Pro-Bot	Resembles Spirit OCX commands	●	●		●	○		Text editor
Finite State Machine	Resembles Spirit OCX commands	●	●		●	●		Dialogue
Visual Basic	Visual Basic (using ActiveX control)	●	●	○			●	Microsoft Visual Studio
Visual C	Visual C (using ActiveX control)	●	●			○	●	Microsoft Visual Studio
JavaScript	(using embedded ActiveX control)	●	●		●			Preferred editor
Bot-Kit	Dolphin Smalltalk	●	●	○	●	●	○	Language sensitive text editor
nqc	C-like		●		●		●	Language sensitive, visual editor available (Bricxcc)
Ada	Ada*		●		●	●	●	Language sensitive editor
legOS	C				●		●	Preferred editor
librcx	C				●			Preferred editor
leJOS	Java				●	●	●	Preferred editor (visual interface available)
pbForth	Forth			●	●			Console
MIT YBL	Logo							Console
* requires nqc								
Legend								
Spirit OCX	Does the programming language use the LEGO Spirit OCX componentware?							
Lego fw	Does the programming language use the LEGO firmware?							
Novice	Is the language suitable for novice users, incorporating direct-manipulation, layered functionality, multi-mode environment (graphical and textual), robustness?							
Low cost	Is the programming language cheap to buy?							
CS	Is the language suitable for teaching principles of computer science							
Power	Is the language powerful enough for advanced students to create complex systems?							

Table 1: A comparison of MindStorms programming environments

Package	URL
RCX language	mindstorms.lego.com/sdk
Robolab	www.ceeo.tufts.edu/graphics/robolab.html.
MindScript	mindstorms.lego.com/sdk2
LASM	mindstorms.lego.com/sdk2
Brick Command	www.geocities.com/Area51/Nebula/8488/lego.html
Gordon's Brick Programmer	www.umbra.demon.co.uk/gbp.html
BotCode	www.iddgroup.com/products/botcode.html
Pro-Bot	prelude.psy.umontreal.ca/~cousined/lego/4-RCX/PRO-BOT/index.html
Finite State Machine	www.idi.ntnu.no/~petrovic/fsm
Bot-Kit	www.object-arts.com/Bower/Bot-Kit/Bot-Kit.htm
nqc	www.enteract.com/~dbaum/nqc/index.html
Ada	www.usafa.af.mil/dfcs/adamindstorms.htm
legOS	legos.sourceforge.net
librcx	graphics.stanford.edu/~kekoa/rcx/tools.html#Librcx
leJOS	lejos.sourceforge.net
pbForth	www.hempeldesigngroup.com/lego/pbFORTH/index.html
MIT YBL	el.www.media.mit.edu/projects/ybl

Table 2: Sources of MindStorms programming environments

programming environment described above, several people created their own. Many made use of the ActiveX component and the Lego provided firmware, but some approaches led to the creation of new firmware in the form of software libraries that could be linked in to 'traditional' programming languages.

Table 1 gives a summary of the most popular community-sourced programming environments, and Table 2 gives their availability.

6. Conclusions

We believe that robotics is a suitable vehicle for teaching a wide range of students, no matter what their age or background. The Lego MindStorms kit is an appropriate low-cost solution. Even though our work comparing programming environments/languages for MindStorms is incomplete, the investigations to date allow us to draw provisional conclusions.

First, MindStorms robotics provides an opportunity to offer a microworld that bridges between computing abstractions and real-world activity. Well-designed microworlds and simulations are useful teaching methods, providing a low-risk, controlled environment in which to learn and develop a firm footing for further learning. Using such systems fosters confidence in using skills as well as teaching those skills.

More advanced microworlds, in which the user can see genuine program code being constructed and executed, are excellent primers to advanced computer programming with integrated development environments.

Second, although a wide range of programming environments has been created for the MindStorms brick*, none meets fully our requirements for an introductory course*. With the exception of RoboLab, none of the graphical environments is powerful enough for students to continue to advanced work. The minimalist textual environments (text editors and command line compilers) are not robust or supportive enough for novice – especially young novice – use.

Finally, we conclude that we need to take a progressive approach, starting with a custom-built, graphical, microworld-based system and later moving to a more sophisticated programming environment.

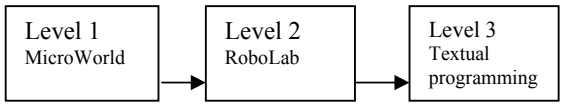

Figure 4. A progression of environments for an introductory course

The microworld-based system would introduce concepts and simple programming in a language-independent, object-based methodology; would use progressive disclosure (e.g., a pseudo-code view linked to the microworld view) to help students map between domain actions and code; and would serve as a bridge to a more traditional programming environment such as one of those reviewed.

References

[1] Beer, R.D., Chiel, H.J, and Drushel, R.F. (1999) Using Autonomous Robotics to Teach Science and Engineering. *Communications ACM*, 42 (6), 85-99.

[2] Carroll, J.M. (1987). Minimalist Design for Active Users (Enhancing System Usability), *Readings in HCI: A Multi-Disciplinary Approach*, 621-626.

[3] Carroll, J.M., and Carrithers, C. (1984) Training Wheels in a User Interface, Communications of the ACM, 27 (8).

[4] Griffiths R., Holland S., Woodman M., Macgregor M., Robinson H. (1999) Separable UI Architectures in Teaching Object Technology. *Proceedings of the 30th International Conference on Technology of Object-Oriented Languages and Systems,* Tools USA '99, Santa Barbara.

[5] Gindling, J., Ioannidou, A., Loh, J., Lokkebo, O., and Repenning, A. 'LEGOsheets: A Rule-Based Programming, Simulation and Manipulation Environment for the LEGO Programmable Brick'. *Proc. of Visual Languages Conference*, (Darmstadt), 172-179. IEEE Computer Society Press. 1995.

[6] Hutchins, E., Hollan, J., and Norman, D. (1986) Direct Manipulation Interfaces, *User Centred System Design,* D. Norman & S. Draper (Eds) Lawrence Erlbaum Assoc., 87-119

[7] Johnson, J. H., 'The "can-you trust it problem?" of simulation science', *Complexity*, 2001.

[8] Johnson, H. J., 'Children, Robotics, and Education', Proc. AROB-7, 16-18 Jan 2002.

[9] Johnson, J.H., Hirst, A. J., The RoboFesta - Blue Peter Robot Design Competition, Research Report 2, RoboFesta Edutainment and Robotics Education Research Centre, Faculty of Technology, The Open University, Milton Keynes, MK7 6AA, UK.

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

[11] Petre, M. (1995) Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming, *Communications of the ACM*, 36 (6), 33-44.

[12] Repenning, A.,(2000) AgentSheets®: An Interactive Simulation Environment with End-User Programmable Agents, *Interaction 2000*, Tokyo, Japan.

[13] Schneiderman, B. (1982*) Designing the User Interface*. (2nd Ed) Addison Wesley.

[14] Smith, R. (1987) Experiences with the Alternate Reality Kit: An Example of the Tension between Literalism and Magic. *Proceedings CHI + GI 87*, 61-67.

[15] Sklar, E., Johnson, J., Lund, H., 'Children learning from team robotics', Research Report 1, RoboFesta Edutainment and Robotics Education Research Centre, Faculty of Technology, The Open University, Milton Keynes, MK7 6AA, UK.

[16] Soloway, E. (1986). Learning to Program = Learning to Construct Mechanisms and Explanations. *Communications of the ACM*, 29 (9), 850-858.

[17] Wasserman, E. (2002) Why Industry Giants Are Playing with Legos, *Fortune*, 144(10), 101-106.

[18] Woodman M., Griffiths R., Robinson H., Holland S. (1998) An Object-Oriented Approach to Computing, *Proceedings of the ACM Conference on Object-oriented Programming, Systems and Languages*, OOPSLA '98, Vancouver.

[19] www.robofesta.net

[20] www.robofesta-uk.org

[21] www.robocup.org