Teaching Computer Science With Robotics Using Ada/Mindstorms 2.0

Barry S. Fagin
Department of Computer Science
US Air Force Academy
Colorado Springs, CO 80840
719-333-3340
barry.fagin@usafa.af.mil

Laurence D. Merkle
Department of Computer Science
US Air Force Academy
Colorado Springs, CO 80840
719-333-7671
larry.merkle@usafa.af.mil

Thomas W. Eggers
Department of Computer Science
US Air Force Academy
Colorado Springs, CO 80840
719-333-3590
tom.eggers@usafa.af.mil

1. ABSTRACT

We present one approach to teaching basic computer science concepts with robotics, using an Ada interface to Lego Mindstorms™[1]. We show simple problems put to students with no programming experience, discuss the solutions, and for each concept explain the advantages of using robots to teach it.

1.1 Keywords

Computer science education, Lego Mindstorms, Robotics

2. INTRODUCTION

The potential use of robots to teach computing principles has long captured the attention of computer science educators. As far back as twenty years ago, one popular computer science text first proposed a robot programming model to teach introductory programming [11], under the assumption that getting the student to infuse a device with behavior of their own design would be a powerful aid to learning. That book remains in print today, and its goal is still shared: to provide a more experiential, "learning by doing" approach to computer science [6,12].

When the book first came out, this goal was difficult to attain, because the robot itself was strictly virtual. The technology of programmable robots at the time made real devices far too costly for ordinary classroom use.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page.
To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGAda 2001 09/01 Bloomington, MN, USA
© 2001 ACM 1-58113-392-8/01/0009...$5.00

[1] Mindstorms is a registered trademark of Lego Corporation.

Fortunately, the remarkable cost/performance advances that have become routine in the computing industry have changed the picture completely. The recent emergence of the Lego Mindstorms programmable RCX "brick" is a case in point. Developed as a joint project by Lego and MIT's Media Laboratory [9], it was released two years ago as part of a larger kit for building programmable robots. The kit includes hundreds of lego pieces, wheels, input sensors of various kinds, and a visual programming environment, all of which permit the construction of programmable robots with remarkably sophisticated behavior.

While originally designed for bright children, the kit has also attracted considerable interest among adults. There are dozens of unofficial books and internet resources available for Lego Mindstorms, maintained by professional engineers who have reverse engineered the RCX hardware, determined the programming and communication protocols, and developed a variety of alternative Mindstorms programming environments for a variety of platforms[2]. One such environment, NQC [1], plays an important role in the research described here.

These alternative environments arose because the visual programming environment developed by Lego is better suited for small children learning programming for the first time as opposed to serious programmers who want to program robots using high level languages[3]. They all suffer, however, from drawbacks that make them poor choices for introductory programming courses. These include unsophisticated error handling, confusing naming conventions, and a failure to abstract away technical issues that could provide unnecessary stumbling blocks to students in an introductory computer science course.

To address this problem, we have developed an interface to the Lego Mindstorms RCX based on a subset of Ada, known as Ada/Mindstorms 2.0. The authors used Ada/Mindstorms in special sections of the USAFA core

[2] See for example [1,7,8].

[3] The programming environment that ships with Mindstorms, for example, does not support variables.

computing course, as part of an experiment in determining the effectiveness of using robots to teach computer science. Our hope was to provide an environment that provides the rigor and introductory programming experience of a high level language with a forgiving and well-behaved development environment appropriate for students with no programming experience.

The technical details of Ada/Mindstorms 2.0 have been presented elsewhere [4,5]; accordingly we do not focus on them here. Instead, this paper presents our experiences with using a high level language and robots to teach some basic computer science concepts. We discuss how robots might be employed to teach a particular idea, show the problems we put to students in the lab, and discuss their solutions. Where robots provide immediate, experiential feedback on whether or not a concept has been grasped, we discuss that as well. We conclude with our plans to evaluate the effectiveness of robots for teaching, and scheduled features for Ada/Mindstorms 2.0.

3. TEACHING SEQUENTIAL CONTROL FLOW, VARIABLES, AND CONSTANTS WITH ROBOTICS

The first introductory programming exercise in our computing course introduces very simple programming concepts, explained to students more or less as described below:

1) *Sequential control flow:* you can give a computer a series of instructions in a particular order that start somewhere, proceed from top to bottom, and then stop.
2) *Variables:* it is very convenient when getting a computer to solve a problem to have quantities that change while the program is running, and to give these quantities a name that suggests their purpose.
3) *Constants:* some quantities don't change at all while the program is running, but it's both convenient and good programming practice to have names for them as well.

3.1 Sequential Control Flow

Sequential control flow is easy to demonstrate with a robot. Any series of commands long enough to appear as a connected list of instructions drives the point home. For Ada/Mindstorms, this could take the form of getting the robot with two motors (connected to RCX outputs A and C) to move forward for 2 seconds, play a sound, go forward again for 1 second, and stop:

```
--sequential control flow example
Output_On_For(Output => Output_A,
   Hundredths_Of_A_Second => 200);
Output_On_For(Output => Output_C,
   Hundredths_Of_A_Second => 200);
Play_Sound(Sound_To_Play => Up);
Output_On_For(Output => Output_A,
   Hundredths_Of_A_Second => 100);
Output_On_For(Output => Output_C,
   Hundredths_Of_A_Second => 100);
Output_Off(Output => Output_A);
Output_Off(Output => Output_C);
```

The sequence of robot actions matches the sequential progression of code on the page. This provides an experiential encounter with sequential control flow, and may seem clearer than more traditional sequential examples that, say, accept a number from the user, go through a series of calculations with no visible results, and produce a final number on the screen.

3.2 Variables

Teaching variables with Ada/Mindstorms requires a robot to work with a quantity that changes while it is running. We believe that the most effective way to demonstrate the concept of a variable is for the robot change its behavior in a way directly related to the quantity in question. This can be accomplished by having the amount of time that the robot travels change by a numeric calculation:

```
--an integer variable
Time_Forward : Integer := 500;

Output_On_For(Output => Output_A,
   Hundredths_Of_A_Second =>Time_Forward);
Output_On_For(Output => Output_C,
   Hundredths_Of_A_Second => Time_Forward);
Time_Forward := Time_Forward*3/4;

--now the robot goes forward for ¾ as long
Output_On_For(Output => Output_A,
   Hundredths_Of_A_Second => Time_Forward);
Output_On_For(Output => Output_C,
       Hundredths_Of_A_Second              =>
Time_Forward);
```

3.3 Constants

To demonstrate the notion of a constant, the robot requires a quantity that does not change while the program is running. We chose a problem that required the robot to make a 90° turn. This is accomplished by turning one motor on, and either leaving the other motor off or rotating it in the opposite direction for a specific amount of time. The amount of time required for an accurate right turn is naturally represented as a constant:

```
Turn_Duration : constant integer := 250;
Output_On(Output => Output_A);
Output_Off(Output => Output_C);
Wait(Hundredths_Of_A_Second =>
   Turn_Duration);
Output_On(Output => Output_C);
```

The need for "tweaking" the turn value in the laboratory when students see that their robot doesn't turn at a right angle also

provides a nice demonstration of the use of constants.[4]

4. TEACHING PROCEDURES WITH ROBOTICS

Procedures and problem decomposition are normally considered more complex topics than those of the previous section; we believe it is unusual to cover them in the first laboratory assignment of an introductory programming course[5]. Robotics, however, provide a very natural way to teach procedures: *a procedure is something you want your robot to do.*

Accordingly, when students are assigned a robot behavior that requires a series of smaller tasks, they see very quickly that these subtasks should be written as procedures. For example, the previous example can be encapsulated into a "Turn_Right" procedure that can be used in later assignments as robot behavior becomes more complex:

```
Turn_Duration : constant integer := 250;
procedure Turn_Right is
begin
   Output_On(Output => Output_A);
   Output_Off(Output => Output_C);
   Wait(Hundredths_Of_A_Second =>
       Turn_Duration);
   Output_On(Output => Output_C);
end Turn_Right;
```

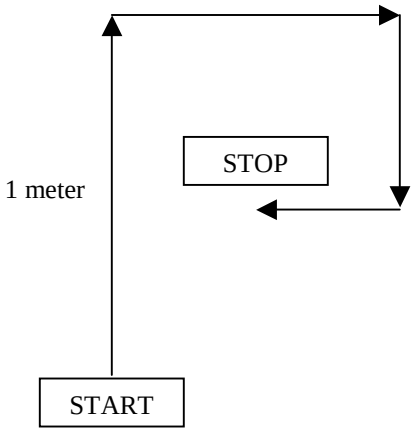
This promotes code reuse, an obvious advantage of problem decomposition that we want students to learn.

There is also immediate feedback with robotics in learning the distinction between writing a procedure and calling it. On more than one occasion, we observed student teams adding the code for a new procedure to their program, then downloading the program to their robot and not seeing any change. Looking through the sequential control flow of their program as the robot goes through its paces, they immediately see that they didn't tell the robot to do its new behavior.

4.1 Putting It All Together: the First Robotics Programming Assignment

We tie in all these concepts in the first programming assignment of the course. The problem statement is as follows:

Make your robot trace a 4-sided spiral path on the floor of the lab, with each side ¾ of the length of the previous one. It should look like this:



The visibly shortening length of each side reinforces the concept of a variable, the need for turning right the same way each time shows how constants are used, the successive nature of the required commands demonstrates sequential control flow, and the use of repeated subtasks to go forward and turn right illustrate procedures. The solution is available from the authors.

5. TEACHING SELECTION AND BOOLEAN EXPRESSIONS WITH ROBOTICS

Much of the power of a general purpose computer lies in its ability to make decisions. Students are often first exposed to this concept by writing simple programs that require the computer to take different courses of action based on a simple input, usually numeric, from the user. In the robot world, however, selectional control flow manifests itself in the much more interesting ability of a student's robot to react to its environment.

Mindstorms robots can receive input in a variety of forms, including light intensity, temperature, and a pulse when an input sensor is pressed. We chose the latter approach for simplicity, and provided students with a two-motor robot equipped with a bumper that sets an input sensor to 1 when pressed. The robot is then tasked to engage in a specific behavior when it bumps into something. Thus selectional control flow can be captured in Ada/Mindstorms with:

```
if Get_Sensor_Value(Sensor => Sensor_1) = 1
    then
begin
    --code for desired behavior here
end;
```

A logical choice for the desired behavior is backing up, turning, and going forward again. The power of decision making in computing is dramatically illustrated by downloading one program that causes a robot to blindly stumble into a wall, and then another that has it back up and scurry off elsewhere.

Understanding the wide range of possible boolean expressions is an important part of selectional control flow, and can be taught by

[4] Another advantage of constants is the ability to only have to change one line of code when it is used in multiple places. This is demonstrated in the second programming assignment.

[5] But see [10] for an alternative view; this topic is still debated today.

equipping the robot with the ability to remember something of its previous encounters and changing its behavior based on what it has done before. Our problem statement for the second Ada/Mindstorms programming exercise specifies a non-trivial set of behaviors for the robot to accomplish. The complete solution is available from the authors.

6. TEACHING ARRAYS WITH ROBOTICS

Recent modifications to NQC permit the use of arrays, albeit in a somewhat restricted fashion. Only integer arrays are supported, and array elements cannot be used as parameters to subroutines. Nonetheless, these changes were enough to encourage us to add support for arrays to Ada/Mindstorms and to construct a laboratory exercise around them.

We introduce the notion of arrays as structured data by challenging students to capture a sequence of numbers input to the robot through a combination of touch sensor and bumper presses. Once the sequence is captured, their program then "plays back" the sequence by examining each number in order and having the robot do a predefined maneuver based on the number read. For example, the sequence 1-2-1-3-4 causes the robot to move forward, turn right, move forward, turn left, and stop. Sample code from our solution looks like:

```
Turns : Integer_Array (1 ..Max_Index);
    --see [5]
Turn_Value : Integer;
Num_Points : Integer := 0;

begin
   Initialize_Robot;
   for Index in 1..Max_Index loop
      Get_Touch_Count(Touches =>
        Turn_Value);
      Turns(Index) := Turn_Value;
      Num_Points := Num_Points + 1;
      exit when Turn_Value <= 0;
   end loop;
   Play_Sound(Sound_To_Play => Down);
   Wait(Hundredths_Of_A_Second =>
       200);
   for Index in 1..Num_Points loop
      Turn_Value := Turns(Index);
      Beep_A_Digit(Digit =>
        Turn_Value);
      Wait(Hundredths_Of_A_Second =>
        50);
      if Turn_Value = Forward then …
```

The effect of an assignment like this is to give a robot programability, a task that at first glance might seem beyond the scope of an introductory programming course.

7. CONCLUSIONS AND THE CHALLENGE OF ASSESSMENT

The use of robots can provide a visceral, "hands on" learning experience for students who have never programmed before. The design-write-run-redesign feedback loop is very fast, and students truly enjoy it. Our anecdotal experience, having taught this course both with and without robots, is that the "fun factor" is extraordinarily high when robots are used.

Robots provide a programming experience that resonates very strongly with certain basic computer science constructs. Constants, variables, procedures, and different types of control flow are all excellent examples of concepts that can be taught very effectively with robots.

As much fun as teaching with robots is, however, our true interest is in improving computer science education. The question is not whether or not robots in the classroom are more fun, but whether or not they are more *effective*[6]. If they are, how much more? For which concepts?

The authors and other instructors taught about 180 students using Ada/Mindstorms, while about 800 students in the same class year learned programming using the course as it was originally structured. Both sets of students were tested with written exams and individual-effort, timed, non-robotics programming assignments. Thus the 2000-2001 academic year provided us with a large data set and a good experimental opportunity to shed some light on these questions. Our results should be available by the summer of 2001.

Future plans for the next release of Ada/Mindstorms include support for separate compilation, tasking, and a simulator. We hope to have these features available by summer 2001 as well.

8. RESOURCES AND ACKNOWLEDGEMENTS

Funding for this project is provided by the Institute for Information Technology and Applications, whose support is gratefully acknowledged.

Ada/Mindstorms 2.0 for Windows is available online at http://www.usafa.af.mil/dfcs/adamindstorms. htm. Source code is also available on the site.

9. REFERENCES

[1] Baum, D. The NQC Web site. Available WWW: http:// www.enteract.com/~dbaum/nqc/

[2] Burks, A.W., Goldstine, H.H. and von Neumann, J. Preliminary discussion of the logical design of an electronic computing instrument, *Papers of John von Neumann*, W. Aspray and A.Burks eds, MIT Press, Cambridge MA, pp 97-146.

[3] Carlisle, M., Graphics for free, *ACM SIGCSE Bulletin*, vol. 31, no. 2 (June 1999), pp 65-68. Available WWW: http://www.usafa.af.mil/dfcs/papers/mcc/sigcsebull99.html

[4] Fagin, B. Using ada-based robotics to teach computer science, Proceedings of the *5th ITICSE Conference*,

[6] Of course, if they are more fun, then they will probably more effective, but the ultimate goal remains effectiveness.

July 2000, Helsinki Finland. Available WWW: http://home.rmi.net/~fagin/Papers/ITICSEWeb/using_ada. htm

[5] Fagin, B. An ada interface for lego mindstorms, *Ada Letters*, vol. 21, no. 2, September 2000. Available WWW: http://home.rmi.net/~fagin/Papers/AdaLetters.htm

[6] Herrman, N. and Popyack, J. Creating an authentic learning experience in introductory programming courses, *ACM SIGCSE Bulletin*, Vol 27, No 1, pp 199-203.

[7] Knudsen, The unofficial guide to lego mindstorms robots , O'Reilly & Associates, ISBN: 1565926927.

[8] Noga, The LegOS Operating System web site. Available WWW: http://www.noga.de/legOS/

[9] Oakes, C. Lego My Ego, Wired News Network, Sep. 23rd, 1998. Available WWW: http://www.wired.com/news/culture/0,1284,15171,00.html.

[10] Pattis, R.E., The "procedures early" approach in CS 1: a heresy, *ACM SIGCSE Bulletin*, vol. 25, no. 1, pp 122-126.

[11] Pattis, R.E., Karel the robot: a gentle introduction to the art of programming, 2nd ed., Wiley 1995.

[12] Urban-Lurain, M. and Weinshank, D. I do and I understand: mastery model learning for a large ,non-major course", *ACM SIGCSE Bulletin*, vol. 31, no. 1 (March 1999), pp 150-154.