

# Robotran: A programming environment for novices using LEGO Mindstorms robots

R. Mark Meyer and Debra T. Burhans

Canisius College  
Computer Science Department, 2001 Main Street WTC 207, Buffalo, NY 14208  
meyer|burhansd@canisius.edu

## Abstract

The LEGO Mindstorms robots provide an excellent introductory platform for students to learn computer programming. However, a major drawback is students' lack of access to robots outside of lab. Our system includes a 2-D graphical simulator and a translator for a simple block-structured language we call Robolang that produces Lejos code. Students can program directly in either Robolang or Lejos, so the system has a longer curricular lifespan, allowing beginners to start with a simpler language and progress to a "real" language, all the while using the simulator to test their programs before downloading to a real robot. This solves the problem of students not having access to their own LEGO Mindstorms kit and enables them to complete assignments outside of lab.

## Introduction

LEGO Mindstorms robots are inexpensive yet versatile programmable toy robots that have been used to teach introductory computer science concepts and programming for a number of years (Burhans 2006, Danyluk 2004, Mataric 2004, Miller 2004, Schumacher 2001, van Lent 2004). Introduced by LEGO in 1998, the Mindstorms features a Hitachi H8 computer enclosed in a plastic "brick" that has bumps and holes for attaching plastic LEGO toy parts. Sensors and motors can be plugged into the brick and programmed.

Robots are typically used to increase student interest, interaction and retention (Bergin 2006.) While there have been some counterclaims, and even some studies that indicate that using the robots does not result in better student learning, interest in the robots continues to grow. (Cliburn 2006, Fagin 2003) are among many who claim that robots increase the fun factor of learning computer programming. LEGO's introduction in 2006 of the successor, the NXT Invention system, is similar enough in price and concept to make future curricular use feasible.

Programming is the primary interest for computer science educators, who have developed a number of languages and environments for LEGO robots. LEGO's original Robolab is a visual programming environment that is limited in important ways, such as not having variables, but fits in with LEGO's original concept of the robot as a toy for grade school children.

A dazzling array of languages have been adapted to the LEGO Mindstorms, including C (NQC), Java (Lejos), Ada, Lisp and Forth. In many cases, variants and subsets of the language were created to accommodate the small size of the robot's memory (32K total) and to add robot control features. Some new languages were invented, such as Drizzle (Ernest 2005), which is a simple object-oriented scripting language.

One problem with traditional languages such as C is that the novice programmer has to grapple with syntax issues that are often frustrating. Our solution to this was to create Robolang, a simple, block-structured programming language that is text-based. An issue with graphical languages such as Robolab and Alice, used to teach general programming in the context of animations and storyboarding, is that they are so different from traditional programming languages that the leap is very far for the students continuing on in the major. Finding the right point of similarity with languages that students might use later with ease of use is difficult (Kelleher 2005.)

Programming environments (IDEs) were developed alongside some of these languages, such as DIODE for Drizzle and a complete simulator system for Jago, a Java variant (Schumacher 2001.)

We have developed an IDE, Robotran, that includes a graphical user interface for editing and downloading as well as an editor where students can write Robolang programs which can be translated by Robotran into Lejos. CS 0 students, who may never program again, use Robolang exclusively. CS 1 students who are majoring in computer science or a related field and who need to learn computer programming use Lejos within Robotran, which provides facilities for compiling and downloading Lejos code.

Common concerns with the use of robots in course assignments are cost and availability (Walker 2004, McNally 2006). We have addressed this problem through the development of a graphical 2-D simulator to accompany Robotran. Our simulator embodies a focus on algorithm development for simple robot behaviors. Other LEGO Mindstorms simulators include Jago, which provides a subset of Java that can be simulated directly or translated to NQC (Not Quite C) for downloading. The [lejos.sourceforge.net](http://lejos.sourceforge.net) web repository has four emulators, two of which have been in development for years. One of them uses Java 3-D graphics and looks appealing but is difficult to use. Our simulator is simple, easy to use, and is focused on pedagogical issues of algorithm development and testing.

To simplify assignments, we use a standard robot structure which we call a penbot (Figures 1 and 2) and which we build in advance for CS0 and CS1 student labs. Our simulator currently displays only the penbot.

### Penbot structure

Building upon a simple roverbot architecture we added a moveable pen to create a robot that can be used in a number of different programming assignments. The sensor and motor port assignments are as follows:

- 1 touch sensor connected to a left front bumper (port 1)
- 1 touch sensor connected to a right front bumper (port 2)
- 1 light sensor mountable in front or back (port 3)
- 1 motor connected to left wheels (port A)
- 1 motor mounted on top and connected to an arm (port B)
- 1 motor connected to right wheels (port C)

Since the LEGO Mindstorms robot is limited to three sensor ports and three motor ports, this represents a "full house" in terms of basic construction.

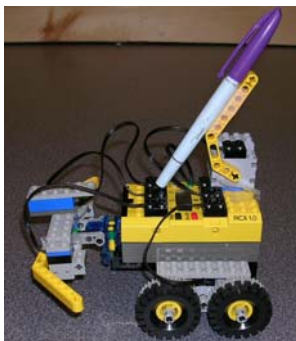


Figure 1: Penbot with light sensor in front

Figure 1 shows the penbot with the light sensor mounted in front so the bot can follow someone holding a flashlight. The pen is in the up, non-writing position.



Figure 2: Penbot with pen in drawing position

Figure 2 shows the penbot with the light sensor mounted in back, and the pen in the down position so it will "write" when it moves.

The light sensor can be mounted in front pointing forward, enabling the robot to hunt for and follow a light. For example, a person holding a flashlight pointed at the bot could cause it to follow the light around. Alternately, the light sensor could be pointed down at the floor to enable line following.

By separating the touch sensors into right and left, the penbot can sense objects or barriers in its mini-world and know on which side they hit, thereby taking evasive action by turning in the direction opposite of the bump. Separating the motors into right and left allows the robot to turn.

The placement of the pen motor and attached technic arm, to which we taped a pen, was tricky due to the penbot's turning radius. A number of alternate configurations were tried but the penbot shown above used the fewest parts, was the smallest in overall size, and alleviated many of the problems with turns. Accommodating a turning radius that is a little too large makes it more challenging to draw desired shapes, providing an opportunity to discuss the difficulty of programming real-world machines.

Sensor configurations can be altered on the penbot: touch sensors and light sensors can be attached to any of the three input ports. It is also possible to connect two touch sensors to one single input port by putting one on top of the other. Whenever either touch sensor is activated by a bump, the program will receive a signal without being able to tell which one was activated. Though the penbot would lose directionality of touch in this configuration, it would gain an open spot for another sensor, such as a second light sensor.

### Robolang

Robolang is the programming language embedded in Robotran. It is a block structured language that is

translated directly into Lejos, making it easy to use objects and expressions from Lejos. The Lejos code that is created from a Robolang program is exposed for students to view and possibly modify. More advanced students can dispense with Robolang and use Lejos. The simulator works with both Robolang and Lejos programs.

There are several types of statements in Robolang:

1. control statements
2. robot control commands
3. native Lejos code that is not changed during translation

Native Lejos statements, if desired, are placed inside a block comment that begins with `/*#java` and ends with `*/`.

The following is a brief exposition of Robolang. A nearly complete Robolang program can be seen in Figure 3.

Assignment of values to variables is done by means of a `let` or `set` statement, which have no semantic difference and exist only to allow programmers to pick the terminology they find most comfortable. Variables are local unless declared global explicitly by putting `global` in front of their names in a separate line. Global variables are needed in when blocks (handlers) that set a boolean variable or an integer variable when buttons are pressed.

The control statements that Robolang provides include `loop`, `while`, `repeat`, `for`, `if then`, `if then else`, `when` and `define`. The first four keywords signal loops, which are similar to Java. Parentheses are not needed for the `while` condition. All of these constructs are delineated with a single `end` statement on a line by itself. The `end` line can also contain the name of the construct, such as `end while`. Where possible, Robolang is insensitive to minor syntactic variants in order to decrease frustration of novice programmers.

The decision statements are limited to `if then` and `if then else`. Since the `end` keyword must always follow the statements that are controlled by the construct, no dangling `else` ambiguity is present. Omitting the `then` word from an `if` statement is often confusing to students of Java and C, so Robotran permits but does not require `then` to appear at the end of the header line of an `if` construct. No parentheses around the condition are required, however, the conditional expression itself must follow Java's syntax rules, with two exceptions. One of these is that the single equal sign may be used to express equality instead of `==`, which is often confusing for novice programmers. Additionally, the words `and`, `or` and `not` may be used instead of the Java operators `&&`, `||` and `!`.

Unlike Python, indentation is not a required feature of the enclosed statements nor does it signal enclosure, although a code beautifier automatically indents the Robolang program for the user.

In the LEGO Mindstorms world, at least as implemented by Lejos, there is a difference between the way sensor and button events affect program execution. Sensors are sampled explicitly by the program code when it copies values from the associated sensor ports to variables. For instance, assigning `S1` in Robolang to a variable is equivalent to invoking the Lejos method `Sensor.S1.readValue()`. Sensors cannot interrupt the program or queue up unread values.

Sensor assignments to the various ports that do not reflect the default penbot construction can be made in a Robolang program. For example, if an extra light sensor replaces the left touch sensor, the program should include the line `S1 is a light sensor`.

By contrast, a button press causes an event to be created and sent to the Lejos system, which then calls the `buttonPressed()` method. Thus, buttons asynchronously generate interrupts. This is reflected in Robolang: sensor ports are polled when a program wants to see what their current values are, and when buttons are pressed it triggers execution of a section of code identified by the `when` keyword. For instance, the following code handles VIEW button presses. In addition to controlling the robot, global variables may be accessed in `when` blocks.

```
when VIEW button is pressed
  stop
  turn right 90 degrees
  let numturns = numturns + 1
end
```

Only three of the RCX's buttons are programmable (VIEW, PRGM, RUN). The ON/OFF button cannot be programmed, lest it become impossible to turn off the robot without removing its batteries.

The following are some robot control statements, with obvious meanings:

```
go forward
go backward
go forward 1 inch
turn right
turn left 45 degrees
turn sharp right 45 degrees
```

Two statements, `continue` and `continue until bump` cause the translator to emit a while loop. The first is an infinite loop and is used when the programmer wants to turn on robot motors until someone pushes the ON/OFF button. The `continue until bump` statement is implemented as a while loop that repeatedly polls the two

touch sensors and exits when either sensor has the value of one.

Robolang also has subroutines similar to JavaScript's functions. Parameters are typeless and consist of names only. Subroutines can be procedure-type, which change global variables or issue a robot action, or function-type, which return a value. The keyword `do` appears in front of the subroutine call when it is procedure-type.

All movement distances and angles are implemented by turning on the motors for a set amount of time. For instance, to turn a full 360°, one method is to turn on the A motor governing the left wheels while making sure the C motor is stopped, and letting this motor run for 9700 milliseconds. The exact number is determined by a calibration statement that equates 1 circle to a stated number of seconds.

The `turn` statement allows the programmer to swivel more quickly by turning on the wheels on one side of the robot in the forward direction as the other motor is turned on in reverse. The keyword `sharp` is added to the `turn` statement to accomplish this.

Full details of Robolang and downloadable versions of Robotran are available at our website: <http://cs.canisius.edu/~robotics>.

## Curricular use

We have a variety of robots at Canisius College that are used in courses ranging in level from incoming freshmen who are taking a course as a general studies math/computing course to juniors and seniors. The LEGO penbots are used in three early courses: CSC 109 "A Robotics Introduction to Computer Science" (alternative to CS 0), CSC 110, "Introduction to Computing" (CS 0) and CSC 111 "Introduction to Programming", (CS 1). In only the first course (CSC 109) are the robots themselves the focus of study. The second course (CSC 110) is a typical breadth first introduction to CS. The third course is a true CS 1 course whose focus is introductory Java programming.

For CSC 109 and CSC 110, Robotran and the Robolang language are used to program the robots directly. The process of translation into another language, and indeed the greater complexity of the equivalent Lejos program, can be used to motivate discussion of programming language concepts in Computer Science for these early students. CSC 111 employs the Robolang IDE and the simulator but not the algorithmic language. Students learn programming concepts via the Java language so they program directly in Lejos. The translator can be used to check their Lejos code

against stored code examples that allow them to compare what they wrote to the recommended Lejos program.

## Programming assignments

The following is a partial list of assignments that have been given in the three early courses mentioned above:

1. The robot moves around, avoiding obstacles when bumping into them by backing up and choosing a different direction.
2. The robot draws various letters by putting the pen down and moving around.
3. The robot follows a light source by turning when it loses the light until found and then moves forward again.
4. The robot receives input from the user in the form of button presses or touch sensor events. The input may be binary or unary. A combination of button presses and touch sensor events may be used.
5. The robot is taught a path by remembering touch sensor presses and the times between them. The path is stored in two arrays: one containing the time until the next bump, and the other which direction to turn.
6. The robot follows a path on its rolling surface by using its light sensor pointed down.

Following a flashlight provides an excellent way to progress from very simple programs to much more complicated and seemingly intelligent ones. In the first iteration, the robots motors turn on when the light sensor receives a signal close to 100%, indicating a light is pointing into it. The robot stops when the light turns off. In the next version, students are asked to consider how to make a turn when the light source is lost, which simulates the robot seeking the light source rather than merely stopping and waiting. Algorithms to seek for the light start out simple, with the robot turning in a circle until the light is seen, whereupon it goes forward again. This algorithm makes the robot always turn in one direction, for example to the right. If the light source moved to the left, the robot will have to make an almost full circle to find it, by which time the light may have moved. A third algorithm has the robot turn to the left and then to the right in order to find the light. Finally a fourth version has the robot "remember" which direction it successfully turned last time so it can turn first in that direction when it loses the light. Such a strategy makes the robot act as though it were being led around on an invisible leash. Students are encouraged to embellish these strategies to make it seem as though the robot were truly intelligent. If intelligence is in part a measure of how flexible an entity's strategies are when it tries to achieve its goals, such robot programs can be said to progress in the direction of greater intelligence.

```

program lightFollower
global constant BLACK = 42
global var turnedLeft = 1
calibrate 1 circle = 9.7 seconds
loop
  var lightvalue = S3
  if lightvalue < BLACK then
    do findLight
  else
    go forward
  end
end
// end of main program, subroutines follow
define huntLeft() returns number
  turn left sharp 25 degrees
  var lightvalue = S3
  if lightvalue > BLACK then
    return 1
  else
    turn right sharp 25 degrees
    return 0
  end
end
define huntRight() returns number
  // similar to above
end
define findLight
  var result
  if turnedLeft = 1 then
    let result = huntLeft()
    if result = 1 then
      return
    end
    let result = huntRight()
    if result = 1 then
      let turnedLeft = 0
      return
    end
  end
  if turnedLeft = 0 then
    if huntRight() = 1 then
      return
    end
    if huntLeft() = 1 then
      let turnedLeft = 1
      return
    end
  end
end
end

```

Figure 3: Light Follower Robot

Figure 3 shows most of the Robolang code for an implementation of the final light following algorithm, where the direction of the last turn is recorded in a variable.

### The simulator

A simulator enables students to test their code outside of lab time. This is particularly important in CS 1. Our simulator is geared to the penbot architecture that our robots embody, displaying a diagram of the penbot seen from above. The simulated robot is controlled by either a Robolang or a Lejos program. The simulator is implemented by substituting calls to the RCX's ROM with method calls to a software object, which then triggers

repainting of the simulated penbot on the screen. Development of the simulator required investigation of the many different ROM calls and how they were represented in Lejos, and uncovered a few errors in the Lejos implementation. One benefit of this approach is that the simulator is not tied to Robolang directly and any changes to the RCX, such as substitution of the NXT robot, will only require remapping the ROM calls. Also, the verisimilitude of the simulated robot should be quite high, if the simulated robot object is repainted correctly and often enough.

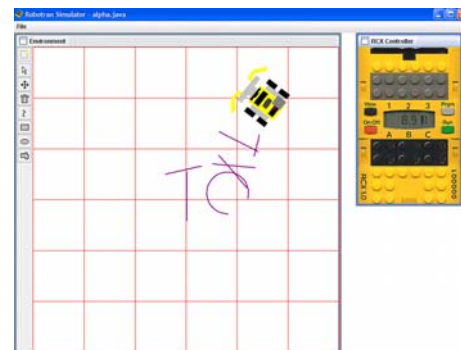


Figure 4: Simulator pane for Robotran

Figure 4 shows the simulator pane. The penbot image that moves is the smaller robot image that has drawn the letters T, O, X, and L. The larger picture of the RCX brick to the right is interactive: users may click on the four buttons of the brick to simulate pressing the real buttons. The small LCD on the real RCX brick is simulated in the larger image. When the same Lejos program was run in the simulator and downloaded to a real brick, the letters that the real brick drew were almost identical to the simulator's output shown in the figure. This reinforces the utility of the simulator program and is exciting for students to see.

## Results

As we move towards incorporating more robotics into our courses we are collecting quantitative and qualitative data from students about the impact of robotics on their attitudes and learning experiences. Overall reaction has been positive. We surveyed three CSC 110 classes with a total of 56 students in the fall of 2005. The survey results are shown in Table 1.

Overall, 59% agreed that working with robots helped them to better understand algorithms, 64% said working with robots helped them to better understand programming, and 71% said we should include more robotics material in the course. In addition to the survey shown above, we asked students (1) did they think our plans to offer a robotics focused version of CSC 110 was a good idea, and (2) would they consider taking the "all-robotics" course. 82%

Question	SA	A	N	D	SD
I enjoyed working with/seeing the robots	14	45	13	4	5
The robots helped me understand algorithms better	13	46	25	11	5
The robots helped me understand programming better	16	48	23	9	4
I would like to do more with robots in the course	38	34	14	7	7
I think the robots should not be part of the class	7	5	14	39	34
The robots helped me to better understand computing	13	41	32	11	4
The robots were confusing	9	7	25	39	20
The robots were boring	5	11	14	34	36

Table 1. Survey Results CSC 110 Fall 2005 (SA=strongly agree; A= agree; N=neutral; D=disagree; SD=strongly disagree)

of students said they thought offering an “all-robotics” course was a good idea, and 48% indicated they would take such a course with another 30% undecided, primarily due to uncertainty about whether the course would count towards a program requirement for their major in the same manner as CSC 110. We also solicited free text comments from students. Responses included, “seeing the visuals really helped me understand topics in Computer Science” and “Robots are fun and make the class more interesting.” Other comments convey overall student excitement and enthusiasm for the robots.

### Future work

Currently, buttons and sensors use a different model of interaction: buttons cause interrupts for which we must write handler code, while sensors are polled explicitly. If buttons could be polled like sensors, or if sensors could generate interrupts, then both modes of input to the RCX brick could be unified and handled in a similar fashion.

Robolang is not object oriented. Statements such as `go forward` are more procedural than their Lejos counterparts, which invoke methods on static objects. For example, `Motor.A.forward();` is a statement that will start the motor attached to port A going forward. The authors listened to objections raised against using LEGO Mindstorms robots to teach CS 1 at a recent national meeting, making this very point (McNally 2006). Clearly, Robolang does not provide much of an introduction to object orientation which may need to be addressed if the students continue on to a true Java or other OOP language programming course.

Finally, the process of student learning has not been studied with regard to specific concepts or specific

programming structures. Deciding what parts of Robotran or Robolang aid or impede student comprehension is an interesting and important challenge that could be addressed in the future, particularly with wider use of the system.

### References

- Bergin, J., Lister, R., Owens B. B., McNally, M. 2006. The First Programming Course: Ideas to End the Enrollment Decline, *Proceedings of ITiCSE'06*, 301-302, ACM Press.
- Burhans, D.T., Meyer, R.M., VanVerth, P., Puehn, D., Steck, V., and Wiejaczka, J.P. 2006. Introductory Computer Science with Robots, *Proceedings of AAI '06*, AAAI Press.
- Cliburn, D.C. 2006. Experiences with LEGO Mindstorms throughout the Undergraduate Computer Science Curriculum, *Proceedings of ASEE/IEEE Frontiers in Education Conference 2006*, T2F-1 – T2F-6, IEEE Computer Society.
- Danyluk, A. P. 2004. Using Robotics to Motivate Learning in an AI Course for Non-Majors. *AAAI Spring Symposium TR SS-04-01*, 93-96, AAAI Press.
- Ernest, J.C., Boswer, A.S., Ghula, S., Sudireddy, S., Porter, J.P., Talbert, D. A., and Kosa, M.J. 2005, Weathering Mindstorms with Drizzle and DIODE in CS0, *Proceedings of ITiCSE '05*, 253, ACM Press.
- Fagin, B. 2000. Using Ada-Based Robotics to Teach Computer Science, *Proceedings of ITiCSE '00*, 148-151, ACM Press.
- Fagin, B. and Merkle, L. 2003, Measuring the Effectiveness of Robots for Teaching Computer Science, *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education, SIGCSE '03*, 307-311, ACM Press.
- Kelleher, C. and Pausch, R. 2005, Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers, *ACM Computing Surveys* 37(2):83-137.
- Mataric, M. J. 2004. Robotics Education for All, *AAAI Spring Symposium TR SS-04-01*, 14-16, AAAI Press.
- McNally, M. 2006, Do Lego Mindstorms Robots have a Future in CS Education? *Proceedings of SIGCSE '06*, 61-62, ACM Press.
- Miller, D.P. 2004. Using Robotics to Teach Computer Programming & AI Concepts to Engineering Students, *AAAI Spring Symposium TR SS-04-01*, 115-117, AAAI Press.
- Schumacher, J., Goldweber, M., Fagin, B., and Klassner, F. 2001. Teaching Introductory Programming, Problem Solving and Information Technology with Robots at West Point, *Proceedings of ASEE/IEEE Frontiers in Education Conference 2001*, F1B2-F1B7, IEEE Computer Society.
- van Lent, C.E. 2004. Using Robot Platforms to Enhance Concept Learning in Introductory Computer Science Courses, *AAAI Spring Symposium TR SS-04-01*, 115-117, AAAI Press.
- Walker, E.L. 2004. Lego Mindstorms Robotics in a (Very) Small Liberal Arts College, *AAAI Spring Symposium TR SS-04-01*, 115-117, AAAI Press.