Using a primary-school challenge in a third-year IT course

Simon
School of Design, Communication, and Information Technology
University of Newcastle, Australia
simon@newcastle.edu.au

Abstract

Programmable Lego Mindstorms robots are used as a challenge activity in competitions for schools, and in various capacities in university-level computing courses. We describe an assignment that we use in a third-year IT course, part of which is identical to one of the school-level challenge tasks. We explore the benefits of this assignment in our university course, and explain why it is legitimate to 'challenge' our final-year students with an exercise undertaken by children in primary school.

Keywords: Lego Mindstorms, robotics, programming.

1 Introduction

Robotics has long been a fascinating topic area in computer science, artificial intelligence, and other related areas. Until the last decade or so, however, most university courses in robotics have been purely theoretical, as the price of actual working robots has been well beyond the reach of most departments. A typical robotics textbook of the early 1990s suggests that industrial robots cost tens or hundreds of thousands of dollars, while personal robots cost only thousands or tens of thousands, are more experimental than functional, and are more use in education than as personal household servants (McKerrow 1991).

When cheaper robots did arrive on the scene their uptake was still fairly slow, because they were perceived as toys rather than real robots. Lego’s Mindstorms robots are a case in point. Costing only a few hundred dollars, they are on first impression simply a Lego construction kit.

What sets them aside, what makes them robots, is the RCX™ ‘brick’ (Figure 1), a large Lego block incorporating an eight-bit programmable processor with 32Kb of memory. Three ports accept input from a variety of simple sensors; another three direct output to motors or lamps; a simple screen displays a handful of characters at a time; and four buttons provide a certain degree of direct control. These features together make the RCX a programmable autonomous device capable of sensing its environment and effecting action – a robot.

Programs for the RCX are written and compiled on a standard computer, then downloaded to the RCX by way of an infra-red link. A single button press on the RCX then tells it to begin execution, and it executes the program until either the program ends or another button press tells it to stop. It is important to recognise that while executing the program, the robot is not under any form of remote control – it is simply carrying out its programmed instructions.

The RCX is not the only device of this kind. Engineering departments have been making their own for many years, and there are several other commercially available kits. This one is undoubtedly popular, though, perhaps in part because of the marketing power that lies behind it. More than a million units have been sold (Devine 2008).

One attraction of the Lego Mindstorms kit is the vast variety of robots that can be constructed from it. A typical kit contains close to a thousand pieces, including a dozen different kinds of wheel, caterpillar tracks, an impressive variety of gears, and hundreds of associated rods, axles, swivels, and other challenges to the imagination.

2 Lego robotics at school level

Early marketing of Mindstorms was to schools, for educational purposes. RoboLab, the programming language sold with the kits, is a purely graphical...
language. Icons representing programming concepts are dragged and dropped onto a form and linked with a ‘wiring tool’ to determine the flow of control. The language incorporates variables, subroutines, branching, looping, and most of the features expected in a procedural programming language, although of course there are limitations and bugs. Figure 2 shows a small segment of a RoboLab program.

Mindstorms robotics appears to have been used generally as a challenge activity for brighter students rather than a core part of the school syllabus, and various competitions have sprung up to support this use. RoboCupJunior in the USA was joined by RoboCup Junior Australia and RoboCup Junior New Zealand. In these competitions, teams of school students compete in robotic dance, rescue, or soccer tasks. The first and third of these are reasonably self-explanatory. In the rescue competitions, robots are expected to follow a line of some sort in terrain of varying difficulty with the goal of finding and rescuing one or more ‘victims’.

For some years the RoboCup Junior Australia rescue competition used the field illustrated in Figure 3. Robots started at the end of the black line near the top left of the image, followed the line to its other end (taking the light-coloured shortcut if desired), then entered the ‘swamp’ and rescued the victim by pushing her out of it. Any similarity between the swamp and Australia is to be taken as purely coincidental, and the competition is definitely not a simulation of Australia’s policy to refugees under the recent conservative government – notwithstanding that both that government and the rescue field were replaced at about the same time.

3 Lego robotics at university level

The use of Lego robotics at university level appears to have been sparked in part by the introduction of text-based alternatives to the graphical programming language. Supplements to existing mainstream programming languages include LeJos for Java (SourceForge 2009), Not-Quite-C (NQC 2007), and Visual Studio for Mindstorms (Mindstorms 2007). We speculate that programming in a graphical language was considered too trivial a task for university students, although in fact this is far from being the case.

Barnes (2002) reported on the use of Mindstorms robots as an approach to teaching Java. Later in the same year an ITiCSE working group (Lawhead et al 2002) set out a road map for teaching introductory programming using Mindstorms robots, discussing the concepts that could be taught in this way and proposing some simple assignments. A poster at the same ITiCSE conference (Garcia 2002) describes the use of Mindstorms robots in a software analysis and design course. All of the teaching reported in these papers used Java and LeJos.

With the flurry of interest in using these cheap and simple robots as teaching tools, it was appropriate to ask whether they offered any advantage to students who used them. Fagin and Merkle (2002) conducted an experiment with
more than 800 students and reported that the students who had used robots actually performed worse in subsequent tests than those who hadn’t. They suggest that one reason for this is that their students had access to the robots only during timetabled classes, depriving them of the facility to continue working on their programs in their own time.

More recent publications explore the use of Mindstorms for teaching courses other than introductory programming. McNally (2006) uses them in a data structures course, again with LeJos, to traverse a grid and determine which squares of the grid are occupied by objects. Jipping et al (2007) use the robots to teach Java bytecode in a computer organisation course.

A number of interesting assignments emerge from the literature, but few educators seem to have decided to make use of the well thought out challenge tasks of the various RoboCup Junior contests. Sklar et al (2004) are an exception: they clearly document their choice to use the RoboCup Junior tasks as assignments in their courses on robotics, artificial intelligence, and autonomous multi-agent systems.

## 4 A course in IT Applications

The course in which we use Mindstorms robots is a final-year undergraduate course called Information Technology Applications, which has been offered at four campuses of the University of Newcastle. Over a single semester the course dips into three quite different applications of IT – currently cryptography, robotics, and computational modelling – taking a brief look at the theory and algorithms of each. While it would never have been possible to justify the purchase of more expensive robots for a few weeks in a single course, the Lego Mindstorms kits fall well within the typical budget of a computing department.

The main point of the IT Applications course is to expose students to just a few applications that they do not encounter in any other courses, in the hope of helping them to realise that there are countless other applications in the real world, and that they should not leave university believing that they know all there is to know about computing.

A specific benefit of the robotics application is that it is perhaps the only course in which students see the interface between their programs and the physical world. They discover, much to their dismay, that the same program cannot be relied upon to produce the same output each time in response to the same input. They discover that the amount of charge in the robot’s batteries can have a dramatic effect on its performance. They discover that a minute difference in the robot’s initial location can lead it to take an entirely different route. They discover that light sensors will give different readings for the same light source when the ambient light conditions are different. They learn the hard way that a debugged and working computer program is not as predictable as they had always assumed.

Robotics also provides a simple introduction to real-time programming. It is not enough that the program respond in a particular way to particular circumstances; it must do so within an appropriate time-frame, or the circumstances will change and the response will no longer be appropriate.

Yet another important lesson is the interaction between hardware and software. There is no such thing as a successful program or a successful robot; rather, there is a successful combination of robot and program. A change, even an apparently minor one, to the robot’s structure can result in a completely different execution of the same program.

Finally, of course, there is the fun. While working with robots can be deeply frustrating, it can also be thoroughly enjoyable, and most of our students do see both sides of it.

## 5 The robotics assignment

The robotics assignment in IT Applications is in two parts, a maze task and a rescue task. The tasks require different programs and different robot structures, so they are assessed in two consecutive weeks.

The marking scheme for the assignment required serious consideration. Programming assignments are generally criterion-referenced: fully satisfying the criteria earns full marks. It was felt, though, that students would perform better under the pressure of competition, so aspects of the competition were incorporated into the marking. In broad terms, teams are awarded a mark for successfully completing the task, a mark for programming style and documentation, and a mark based on their rank in the competition. The first time the assignment was run, each of these items earned a third of the available marks. In subsequent offerings the mark for successful completion was raised to 50%, the mark for style and documentation was lowered to 30%, and the competition mark was lowered to 20%. This change meant that any team successfully completing the task would earn at least half marks.

The kits are issued to groups on loan for the duration of the assignment, getting round the problem of restricted access reported by Fagin and Merkle (2002). Students sign an undertaking to return the kit in good condition, and so far there have been no problems with this arrangement. Students who know young children are encouraged to involve them in the design and construction of the robots.

Groups are permitted to program the robots in whichever language they wish. Those more familiar with Java tend to prefer LeJos, but many others choose PBrick, a language that comes with Visual Studio for Mindstorms. We thought long and hard before deciding to permit them to use RoboLab, as intuition suggests that this is an easier option. In practice, though, it isn’t. Students familiar with text-based programming would spend a long time learning the graphical language and its environment, and more time discovering and working around the limitations and bugs in the language. Besides, this choice
would make it difficult for them to work at home, as RoboLab is a licensed product for which our site licence does not extend to students’ computers.

5.1 The rescue component

The rescue task of the assignment is modelled closely on the RoboCup Junior Australia rescue task, using the field illustrated in Figure 3. The following aspects are taken directly from that task:

- robots must fit within a vertical cylinder 18cm in diameter;
- competitors have five attempts at the rescue;
- for each attempt the referee places the victim in a different part of the swamp;
- each attempt is limited to 90 seconds;
- if the robot leaves the road during an attempt, it must be restarted at the beginning of the road while the timer continues;
- the score for a successful rescue is the number of seconds taken for that rescue;
- the score for a failed attempt is 100;
- the score for the whole activity is the sum of the team’s best four scores from the five attempts;
- teams are ranked according to the scores, the lowest score winning.

While all of the points above determine a team’s rank, and thus the competitive 20% of the score, there is also a gradation in the 50% score for success. There are four measures of partial success, the first three being to reach specified points along the road and the fourth being to reach the swamp. A team’s success is based not on an aggregate of its five attempts but on the best of the five. Thus a team that rescues the victim just once in five attempts still scores full marks for success, and a team that never even reaches the swamp can still earn up to 60% of the success mark.

Following the road is in principle quite a simple task. Each Mindstorms kit comes with two light sensors, which shine an infra-red light and measure the intensity of its reflection. The three colours on the field (black, green, and yellow) reflect different amounts of light, and thus return clearly distinct sensor readings. So, for example, a robot can be programmed to straddle the road by instructing it to turn left when the left sensor reads black, to turn right when the right sensor reads black, and to go straight ahead when neither sensor reads black.

But even this simple task has challenges. The road has some very tight turns, especially towards the swamp end, and it is easy for a robot to be misled into leaving the road on such a turn. The problem is exacerbated if the robot is long, limiting the tightness of its turns, or if the wheels are too far from the sensors, limiting its responsiveness. The quest for more speed, while potentially offering a better score, also increases the chance of overshooting a tight bend.

On reaching the swamp the robot will generally move to a different phase of its program, one in which it systematically or randomly criss-crosses the swamp in the hope of eventually ‘finding’ the victim and pushing her to safety. This phase has its own problems, such as trying to ensure a good coverage of the swamp in a reasonably short time.

In the five years that we have used this assignment we have seldom seen a robot succeed on all five rescue attempts, and we have often seen teams fail on all five. What seems a relatively simple task is in fact a worthy challenge to groups of three or four students.

5.2 The maze component

While the rescue task is based on a well-known competition for school students, the maze task is one that we have never seen elsewhere. Other people (Sklar et al 2004, Jipping et al 2007) have used mazes in their assignments, but these are mazes in which the robot has a line to follow, in a similar manner to the rescue task.

Our maze is built of white timber walls 10cm high, with no passage less than 20cm wide. Robots are placed at a referee-specified location inside the maze and must find their way to and out of the single exit (Figure 4). The maze is reconfigurable, so the students do not know its structure until the competition begins. Their programs must be general maze solvers, not specific to a known maze.

The standard way of solving a maze is wall-following: place a hand, for argument’s sake the left hand, on a wall, proceed in whatever direction one must to keep that hand on the wall, and come eventually to the exit. The choice of wall can have a dramatic effect on the distance to be travelled – consider the distance to be travelled by the robot in the lower left corner of Figure 4 depending on which wall it follows – but should have no effect on the eventual outcome.

However, wall-following is guaranteed to succeed only if there are no islands in the maze. Consider the maze in Figure 5, and the time that a robot could spend following a wall if that wall were on one of the islands.

Figure 4: a simple timber maze
In our assignment students are given two mazes, one without islands and one with islands, and are to solve both mazes with the same program. Wall-following clearly makes sense for the first maze, and some sense for the second, but the program must have some way of deviating from this pattern. Some groups consider trying to deduce when the robot is touring an island; for example, they might note if there have been four consecutive turns in the same direction. However, islands are not necessarily rectangular, which makes this an unreliable measure. Worse, the robot’s wall-following movement invariably involves small alternating left and right turns, so the program would have to distinguish between these turns and the bigger turn resulting from a junction between walls. In the end most groups program something more arbitrary, such as a 1% chance of deliberately turning away from the wall at any time.

Wall-following can be implemented with light sensors, touch sensors, or both. Because the walls are a uniform colour, the amount of reflected light received by a light sensor can be used to measure its distance from a wall, and the robot can be programmed to maintain a more or less fixed distance from the wall as it proceeds. A second sensor, pointing forward, can detect a wall blocking the path. Alternatively, a blocking wall can be detected by a touch sensor as the robot actually strikes the wall.

Robots are given ten minutes to solve each maze. Within that time they can be restarted as often as the team wishes. As with the rescue competition, there are partial success marks for reaching various points of the maze, and a competition mark based on the lowest time from (re)starting to exiting the maze. Some robots will exit the maze several times in the ten minutes, while others will not once find the exit.

However the wall is followed, careful decisions must be made as regards the robot’s means of progress. The robot in Figure 4 is in fact stuck in a corner. It repeatedly senses the wall on its left and turns to the right, then senses the wall in front of it and turns back to the left. Some robots wedge themselves so tightly into corners that they cease to move at all. On one memorable occasion a robot, having finally reached the exit after some eight minutes, became stuck halfway out. A restart is generally the best response to becoming stuck, but there are robots that simply return to exactly the same spot and become stuck in exactly the same way.

6 Assignment outcomes

As indicated in section 5, there are many problems associated with the rescue and maze tasks, not least of which are those related to the physical nature of the robots, and there are robots that simply fail to complete one task or the other. Even so, there are few teams that score less than 50% for the assignment. On each task, a reasonably written and documented program and a partial success tend to ensure this sort of mark even for the team that comes last in the competition; and it is seldom the case that the same team comes last in both rescue and

Figure 5: a maze with islands to confound wall-following

Figure 6: a selection of robots ready to compete in rescue
maze.

At the other end of the scale, many teams score high marks on this assignment, notwithstanding the fact that they cannot all come first in either or both competitions.

Despite its many frustrations, the students generally enjoy the experience of designing, building, and programming the robots, of which a selection are shown in Figure 6. Some even consider buying their own kits once the course is over, though it’s not clear what use they would put them to. Others, not surprisingly, are pleased to see the back of the robots, and resolve not to play with Lego for many years to come.

Almost without exception, though, the students agree that programming takes on a new dimension when it involves control of a physical device; this important lesson would itself be reason enough to use robots in an assignment.

7 Discussion

At first glance it seems unlikely that a task used in primary schools can be legitimately used as an assignment in a final-year computing course. Of course there are many differences between the two uses. For example:

- RoboCup Junior tends to involve the brightest few school students; our assignment is for all of our students;
- RoboCup Junior teams tend to work all semester or even all year on the task; our students have just a few weeks to complete it;
- RoboCup Junior teams tend to work on just one task of the three in the competition; our students have to work simultaneously on two;
- RoboCup Junior teams tend to be given a great deal of help and guidance by their teachers and mentors as they work on the task; our students are expected to be far more self-reliant;
- RoboCup Junior teams tend to be new to programming; our students are highly familiar with programming, but not with programming to control physical devices in the physical world;
- RoboCup Junior teams have the principal goal of winning the competition; our students are aiming to score good marks in an assignment, which is not necessarily the same thing.

In summary, we share the opinion of Sklar et al (2004) that some of the tasks developed for school-level competitions make excellent university-level assignments, and we heartily recommend this approach to other educators.

8 Postscript: the NXT

The NXT (Mindstorms 2009) is the next generation Lego robotics product. Will it take over from the RCX for our assignments? Certainly not in the near future, and possibly not until the RCX kits cease to function.

At least in the Education kit, the NXT has far fewer wheels, gears, and general construction pieces than the RCX. Along with a less Lego-like structure, this seriously limits the basic shapes and structures of NXT robots, constraining them to be all of the same mould.

Our first experience programming the NXT, in the new graphical language that is sold with it, suggests that its movement is far less precise than that of the RCX. This might be an artefact of our own programming or of the language, but we had difficulty programming the NXT for either the old-style rescue task (as mentioned in section 2, RoboCup Junior Australia now uses a different rescue task) or our own maze task. Even the NXT’s size works against it for these tasks – it simply takes up more space.

Text-based programming languages are now becoming available for the NXT, and it is possible that these will offer some relief from the imprecision, but we remain concerned that it might be a feature of the physical architecture.

Finally, of course, there is the timeless argument against obsolescence. So long as the RCX kits provide a worthwhile experience for our students, it would be environmentally irresponsible to discard them in favour of a new product, even if that new product were somewhat better.

9 References


