Innovation in Teaching and Learning in Information and Computer Sciences

ISSN: (Print) 1473-7507 (Online) Journal homepage: http://www.tandfonline.com/loi/rhep14

Introductory Problem Solving and Programming: Robotics Versus Traditional Approaches

Amanda Oddie, Paul Hazlewood, Stewart Blakeway & Alma Whitfield

To cite this article: Amanda Oddie, Paul Hazlewood, Stewart Blakeway & Alma Whitfield (2010) Introductory Problem Solving and Programming: Robotics Versus Traditional Approaches, Innovation in Teaching and Learning in Information and Computer Sciences, 9:2, 1-11

To link to this article: http://dx.doi.org/10.11120/ital.2010.09020011

INTRODUCTORY PROBLEM SOLVING AND PROGRAMMING: ROBOTICS VERSUS TRADITIONAL APPROACHES

Amanda Oddie, Paul Hazlewood, Stewart Blakeway, Alma Whitfield
Department of Computer Science
Liverpool Hope University
Hope Park, Liverpool, L16 9JD
oddiea@hope.ac.uk, hazlewp@hope.ac.uk, blakews@hope.ac.uk, whitfia@hope.ac.uk

ABSTRACT

The ability to solve problems is one of the key skills that computing students require in order to learn programming. However they find this a challenge. The Computer Science Department at Liverpool Hope recruits students to two programmes, BSc. Information Technology and BSc Computing. Both programmes have used bespoke software and text books to guide the teaching and learning of programming. This paper examines how innovative technologies such as the use of robotics can facilitate the students' understanding and application of problem solving and programming. It discusses and builds on previous initiatives undertaken at Liverpool Hope.

Keywords
Problem solving skills, robotics, teaching programming, Flowcode, innovative, programming text books, constructivist, instructional design, blended learning.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the contribution of Years 1 and 2 students (Academic Year 2006-2007 and 2009-2010) in response to the questionnaire on the learning and teaching of programming. Student comments remain anonymous and verbatim, and permissions have been given to publish. The authors also acknowledge the contribution made by Mark Barrett-Baxendale in the development of the research questionnaire.

1. INTRODUCTION

Students now entering university to undertake undergraduate Computing and Information Technology (IT) programmes are a more diverse group (Jenkins and Davy, 2001). This could be attributed to various factors such as:

- the decline in mathematical ability (Herterich, 2004) and symbolic reasoning;
- the fact that some IT degree students do not view programming as being central to their programme;
- the clear differentiation between A level Computing and Information Technology qualifications means that some students entering undergraduate courses may not even see themselves as programmers;
- students consider themselves to be "digital natives" and feel that they are already computer literate due to their exposure to computers, the Internet and social networking sites;
- many students have no experience of problem solving or programming and find these activities difficult.

1.1 Programming

Problem solving and programming concepts are seen as difficult and abstract skills to learn with little to relate them to things familiar to most students. Programming languages taught are seen as obstacles to engagement and interest in computing subjects (McCracken et al*.*, 2001). It has been argued that students have difficulties with programming if they lack confidence in their ability to use symbolic reasoning (McDermott et al*.*, 2007).

Traditional approaches to learning programming include identifying and understanding the syntax and structural elements of the chosen language then trying to combine these with techniques and understanding of structured problem solving. This methodology is fairly successful if students have had experience of problem solving and symbolic reasoning (usually obtained through exposure to mathematics). However as indicated above students entering into undergraduate Computing and IT courses are a more diverse group. Students at Liverpool Hope entering on to the BSc Computing programme are required to have a mathematics qualification (a minimum of GCSE grade C) whereas those entering on to Information Technology Programme are not.

1.2 Problem Solving

Problem solving is regarded as an important stage in the development of a program. It is often considered a difficult skill to learn. Jenkins (2002) states that problem solving is a complex process which requires various skills. The problem has to be identified and understood. Whitfield et al*.* (2007) state that problem solving is a complex process which requires various skills. Programming is a product of the solution to a particular problem. In order to be able to program, the problem first must be understood and often dissected. In computing, students need to translate the algorithm for the solution into code. Dijkstra (1989) discusses programming with its abstract nature as not being easily related to the familiar or existing knowledge. Oldehoeft and Roman (1977) consider that students need to understand problem solving methodologies at the basic level if they are to progress as programmers. Beaumont and Fox (2003) also suggest problem solving is a non trivial process that requires many skills including the identification of the central issues; recognition of relationships, familiar situations and patterns.

Before trying to teach programming to students it is important for them to be able to identify a problem and understand the steps to be taken to solve it. If these steps are ignored it can lead to poorly defined solutions or programs that are created through trial and error – “hacking”. A traditional approach is taken to lead students through the steps of structured problem solving, flow charts and trace tables identifying sequence, selection and repetition, in order to create a solution that can be formalised as an algorithm presented in pseudo code. The algorithm can then be translated into a program (Burton and Bruhn, 2003).

Although the theory is important, research and ongoing developments show that the use of gaming (Chang, 2008), robotics (e.g. Flowers, 2003; Turner and Hill, 2007; Gandy, 2010), and visual programming tools such as Alice (Cooper et al. 2003; Alice, 2010) to “hide” the mechanics/technical aspects of problem solving, and programming has the potential to successfully engage learners through interesting activities. Although often focussed on the engagement in programming tasks, the research does suggest that being able to visualise (and predict) what is happening encourages a better understanding of, and creativity in, problem solving. One could say that the technology is being used as a “mindtool”. According to Jonassen (2000), when students are using technology as mindtools they become creators of the knowledge that they are learning (in line with constructivist principles). This in turn requires them to think harder about the knowledge being learned. Importantly, students are not as reliant on the teacher to interpret problems for them and so are constructing their understanding and views within the framework of the course and the tasks set.

1.3 Liverpool Hope Students

Learning to program is a challenge that all Computer Science students face; the difficulties being with syntax and the strict structure of the language. An additional constraint is the prior knowledge that the students have when beginning University as well as their expectations. Computer Science students at Liverpool Hope are equipped with at least a grade C in GCSE mathematics. This is seen as important because, although not related to programming, mathematics does begin to address the issues associated with problem solving skills. This is backed up by Herterich (2004) who suggests that the decline in mathematical ability, specifically in analytical and logical reasoning, problem solving skills and algebraic manipulation, has had adverse implications in Computing disciplines and that this has led to problems with formal manipulation of symbols (Programming) and multi-step processes (Algorithmic Design and Data Structures).

Liverpool Hope University has a strong Education degree programme and many students apply for QTS (Qualified Teaching Status). Although these students are studying to become teachers they are also required to study in another subject, one of the choices they have being Information Technology (IT). These students are committed to their Education degree as that is their chosen career, however the commitment on the IT course is sometimes less prevalent. Similarities to this situation are described by Anderson et al. (2003) in relation to Liberal Arts students. Where students do not see programming as an essential core of their studies they will not be motivated enough to devote sufficient time to the study. The IT course is not restricted to Education students and is often more popular in numbers than the Computer Science degree. Thus, at Liverpool Hope we have to meet the learning requirements of students who (a) expect programming to be part of their Computer Science degree, (b) take IT in a combination of their primary degree (QTS), or (c) those who do not meet the requirements of the Computer Science degree but are offered a place on the IT degree, all of whom perceive programming with different importance, have different expectations, different experience and different commitments.

Previous research carried out at Liverpool Hope University (Whitfield et al., 2007) discussed how the use of bespoke software and tailored course materials was used to support students who were new to problem solving and programming. This was grounded in a traditional understanding of problem solving principles and looked at moving to a visual and more constructivist approach to learning. Although successful there were still some who struggled when errors were discovered in their solutions, had difficulty identifying other possible solutions and struggled when presented with new problem scenarios. This paper builds on these experiences and explores the use and efficacy of robotics in the first year programming curriculum.

2. ROBOTICS

Research in this area has demonstrated that the use of robotics to teach programming can be very effective.

Lawhead et al. (2003) stated that robots "…provide entry level programming students with a physical model to visually demonstrate concepts" and "the most important benefit of using ROBOTS in teaching introductory courses is the focus provided on learning language independent, persistent truths about programming and programming techniques". Turner and Hill (2007) found that when using robotics there was an improvement in grades and that there was a positive response from students stemming from the benefit that the use of robots provided a more tangible way of visualising the outcome of their programs. This is supported by Doswell (2006) who stated that "robotics provides students with the opportunity to test the results of abstract design concepts through concrete, hands-on robotic manipulation". Gandy (2010) suggested that "the robot enables quite complicated and interesting actions to be performed using very basic code structures. It was felt that this makes it an ideal vehicle for assisting in the teaching of basic programming".

Generally research has shown that robotics is beneficial for the teaching of programming in a number of ways. It can:

- engage all students in core tasks whilst providing opportunities for more advanced students;
- help to mitigate the abstract nature of programming (Dijkstra 1989) by providing a richer tangible real world environment for seeing feedback;
- encourage experimentation and exploration e.g. use of sensors;
- provide visualisations of implementations of solutions to problems;
- offer a real world environment for learning about programming concepts and designing more robust code e.g. do not want robot to fall off the work table, collision with walls;
- learning vocabulary of programming in an interesting way.

3. TEACHING MATERIALS

Various textbooks, software and resources are used to support the programming courses. The Computing Fundamentals course (BSc Computing programme) uses the same core structure for teaching materials as the Computer Concepts course (BSc Information Technology programme) with some changes and variations that are discussed below. These changes were made to take into account the higher ability level of these students and the different programme requirements.

3.1 Text Books

Many courses at Liverpool Hope adopt textbooks that are required for study as part of a course. While textbooks can be invaluable, it has been observed that often they are not entirely specific for the purpose of a course. This is especially apparent for first year courses at Liverpool Hope. Therefore, textbooks were created internally so that they were specifically tailored for the first year course. Bennedsen et al. (2005) describe the usefulness of published text books as limited to the delivery of the finished product but not for the development process. Programming text books tend to be easy to understand at the beginning, developing very quickly into a complexity that demands a much higher level of understanding from students.

The advantages in this approach are that with in-house publications students only pay for the copying of the book and not the high price of a commercial publication. Students studying courses at Liverpool Hope felt that text books did not focus on their specific learning process. This has been incorporated into the internally created books where the pace and complexity for the recruited student is better considered. Such internally created books are freely available on the website and as problems are encountered with the texts, they can be immediately updated. There are, however, disadvantages, for example, the amount of time required by staff in order to create and review the in-house text books.

Computing Fundamentals made use of both in house textbooks that were created for the course which were specifically tailored to focus on the C programming techniques and concepts the students were required to learn. It was recognised that some programming *text books* did not address the requirement of the students. However, a programming *reference* book, “The C Programming Language, 2nd ed.,” (Brian W. Kernighan and Dennis Ritchie), was considered to be a useful resource for supplementing and furthering understanding of the fundamental concepts presented in the in-house materials. Computer Concepts relied wholly on in-house text books.

3.2 Online Support

The courses delivered by the Computer Science Department are supported using Moodle, the University’s virtual learning environment (VLE). The VLE contains all the lectures, seminar activities, assessment details, course schedule, software and any other information required for that course. The VLE provides functionality, for example, online submission, discussion forums and message facilities. A Moodle course was set up for Computing Fundamentals and Computer Concepts.

3.3 Software

Software used by our Computer Concepts students initially consisted of JavaTrainer and was discussed in detail in Whitfield et al. (2007). It was decided that commercial Integrated Development Environments (IDEs) such as NetBeans were initially too overwhelming in appearance, number of instruction sets (functions) and complexity. Liverpool Hope IT students are introduced to JavaTrainer, an in-house development, which was designed to allow students to understand programming in an uncomplicated forum.

The students experience JavaTrainer during the first part of their course and are introduced to a traditional approach to programming. In the second part of the course they are introduced to object oriented concepts; classes, objects, signatures, parameters, message passing, attributes and protocols. The students begin to implement these concepts by programming in a freely available IDE, in this case BlueJ.

In the Computing Fundamentals course programs were implemented in C. Students developed their programs in the Linux environment using a text editor and compiled their code at the command line using the gcc compiler. It was a minimalist environment but it allowed students to focus on content and debugging with little or no effort being expended on learning an IDE. Compiling from the command line presented the student with the opportunity to understand the stages in the compilation process. (Memorising compilation commands did not present a barrier as one could use the history feature of the CLI to recall the command.) The students later used Dev C++ to give them experience of writing C programs in a Windows environment using an IDE.

The techniques described above are similar for both courses and depend upon teaching programming using traditional methodologies. A new element was introduced into the Computing Fundamentals course which made use of robotic buggies and Formula Flowcode.

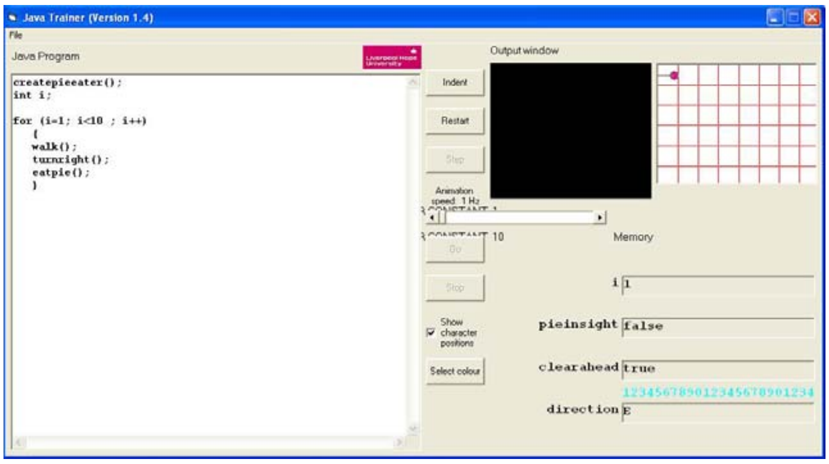
4. DELIVERY OF THE COURSES

This section discusses how innovative methods have been developed to teach problem solving on the Computing Concepts course and then discusses how the approach taken with Computing Fundamentals builds upon this work in particular with the introduction of robotics.

4.1 Computing Concepts

Several approaches have been adopted to teach programming skills to the IT students. Students build upon previous knowledge using the constructivist view of learning, as problem solving techniques are applied to everyday problems. The examples adopted of baking a cake, charging cars to park for periods of time or knitting a jumper which were introduced early in the course are discussed in Whitfield et al. (2007). The team felt that basic examples were required to start the students on the journey of “logical reasoning and algorithmic expression”. (Whitfield et al. 2007). Students are introduced to the three constructs of sequence, selection and repetition and use conventional flowcharts in order to help them express their solution in structured English. When the students begin to examine the components of a problem they understand, even before they start coding, how to continue to refine their solution until they understand how this can be adapted to high level programming language. At this stage the students are ready to begin tracing through algorithms for testing and debugging, leading to the recording of the values of conditional statements and the values of any variables within their solution. The first programming language that the students encounter is Java which they experience through an in-house designed environment (Java Trainer). This was designed to enable students to take the first steps on what will become more complex as they progress on to environments such as BlueJ.

Java Trainer, although of limited functionality, is an application implemented in Visual Basic and written specifically to teach the students the three problem solving constructs of sequence, selection and repetition expressed in Java syntax. Students are introduced to loops, selection criteria and sequences of instruction through visual representations instead of text based solutions. Experience has demonstrated that students react better to visual representations of programming than text based solutions. Java Trainer has the advantage of visual representation of programming similar to Pooples (Culwin, 2005) but with more flexibility. The functions the students engage with include the facility to create a creature called PieEater which obeys basic commands such as walk, turnleft, turnright, eatpie, penup, pendown.



```
createpieeater();
int i;

for (i=1; i<10 ; i++)
   {
   walk();
   turnright();
   eatpie();
   }
```

```
i           1
pieinsight  false
clearahead  true
            123456789012345678901234
direction   E
```

Figure 1 Java Trainer showing code and grid with PieEater.

The student sets instructions which are reflected in the visual representation as the PieEater carries out the instructions. Students learn to manoeuvre the creature, e.g. have the creature walk, turn, show a pen line with the instructions pendown. Students experience loops, repetition and sequence, as well as predefined variables which are also introduced at this stage, one example being clearahead which is true when PieEater can walk forwards and false when it is facing a boundary wall.

As discussed in the earlier work presented by Whitfield et al. (2007), the Computing Concepts course has tried to develop a more constructivist and visual approach to learning as can be seen with the PieEater approach where students can create programs and instantly receive feedback watching the PieEater move around the screen. This software also encourages the students to explore their solutions to investigate what happens to the PieEater. However the students still struggle with some of the problem solving and programming concepts. Although the in-house software is beneficial as a learning tool it is still limited to visual feedback as there is only interaction with the computer and may be seen by some students as an IDE. The remainder of this section highlights how robotics was introduced on the Computing Fundamentals course.

4.2 Computing Fundamentals

Traditionally the Computing Fundamentals course is based around C and in the past has been delivered using an instructional design approach. This approach involved teaching the core concepts to the students, getting the students to practice the task and then the students applying the techniques to various problems. However this year it was decided to introduce the use of robotics at the start of the course using Flowcode Buggies (Figure 2) and Formula Flowcode software (Matrix Multimedia, 2010). The rationale for this was three-fold:

1. It was seen as an ideal way of introducing the initial stages of programming; that is, problems solving, algorithmic design, flow charts, and the basic components of sequence, selection and repetition. These are all vital components and concepts that students must understand before becoming too involved with the syntax and grammar of the language. The use of Flowcode supported this as it is a graphical programming language.
2. To engage the students, as programming is seen as a difficult task by students.
3. To change the pedagogical approach of the course to provide a more constructivist environment in which to learn. It was a perceptible shift from a predominately instructional approach to one that includes more constructivism.

It can be argued that the course was adopting more of a “mindtools” approach with the use of robotic buggies and Flowcode. This meant a move away from learning from technology as with the Computing Concepts approach toward learning with the technology (the robotic buggies).

Flowcode 4 is an advanced graphical programming language for microcontrollers which currently supports PICmicro, AVR and ARM series microcontrollers. The advantage of Flowcode is that it allows those with little or no programming experience to create programs in minutes (Matrix Multimedia, 2010). The Computing Fundamentals course used the Flowcode 4 in conjunction with formulae Flowcode robot vehicles. These vehicles use the PICmicro microcontroller and are used to teach robotics and, in the case of Computer Fundamentals, to deliver flow charting and problem solving techniques.

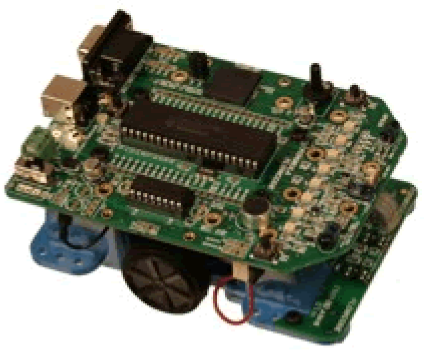


Figure 2 Formula Flowcode robotic vehicle (Matrix Multimedia, 2010)

The two wheeled robot uses the Flowcode graphical programming software (Figure 3). The advantage of Formula Flowcode is that it is a visual language that allows programs to be built by dragging and dropping icons which represent program structures and functions. This is beneficial to novice programmers as it allows them to focus on the problem to be solved rather than struggle with the syntax of a language e.g. dragging a block on the screen to represent a for loop is easier for many students than learning the structure and syntax in C. It helps them to become more disciplined as it encourages the use of a simple develop, test, and try model. Students develop the program, simulate its functionality on-screen and then click on a button to download the program to the buggy via USB connection.

The buggy uses an advanced PICmicro 18 series microcontroller with internal precision motor controller circuitry, has 3 infrared distance sensors, line following sensors on a separate circuit board, a buzzer, audio level sensor, light sensor, two spare switch inputs and 8 user programmable LEDs (Formula Flowcode Datasheet, 2007). These features were considered when designing the exercises to provide a range of interesting problems ranging from the straight forward (e.g. flashing the LEDs) to the more complex (e.g. maze solving problems that made use of the sensors).

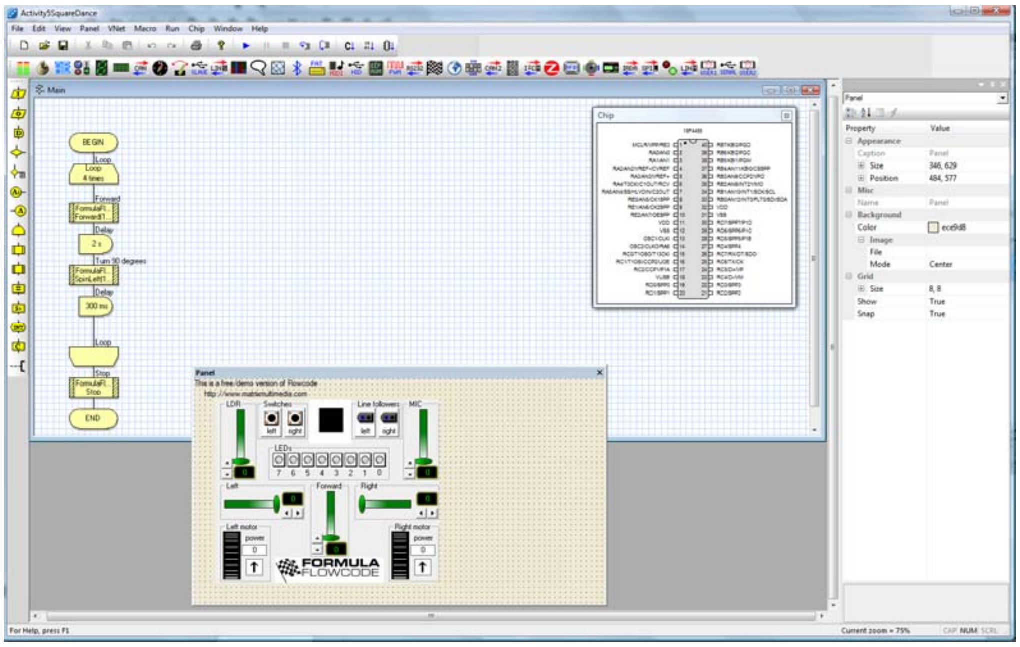


Figure 3 Flowcode Interface

Programming is inherently difficult for many students to understand. As discussed, it was decided to use the Flowcode to allow students to focus on the problem solving, algorithmic design and flow charting aspects of programming before worrying about the syntax and grammar of the C language. During the first session of the course the students were taught about algorithms and flowcharts. This provided the students with an understanding of problem solving techniques. During the following session the students were introduced to Flowcode and given a demonstration of the interface, how to construct a program (represented as a flow chart), how to run this on the simulator and finally how to transfer the program to the buggy for testing. The Flowcode simulator is illustrated in Figure 4.

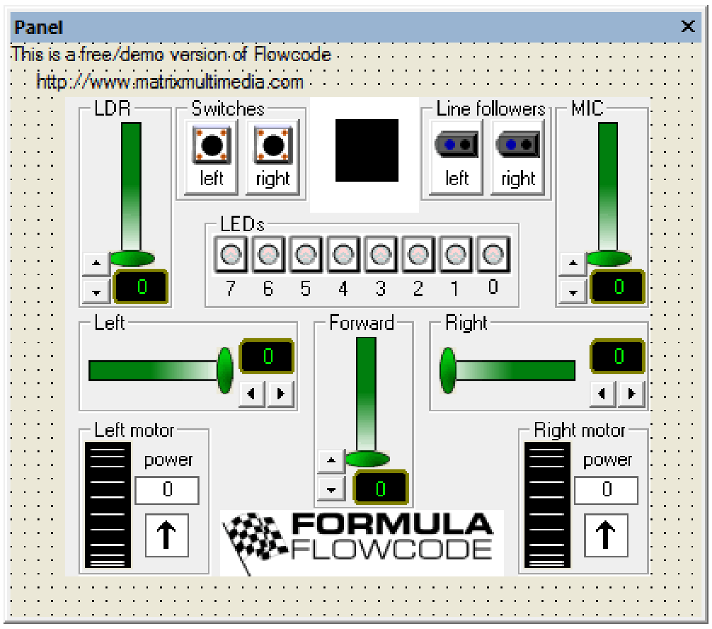


Figure 4 Flowcode Simulator

After these initial sessions the students were presented with a series of portfolio questions and given three weeks to complete them. The activities comprised of a set of compulsory core activities designed to provide an understanding of flowcharting, problem solving and the core programming constructs of sequence, selection and repetition along with some additional optional activities designed to be more complex and develop a deeper understanding of problem solving and programming. A sample solution to one of the portfolio questions is shown in Figure 5.

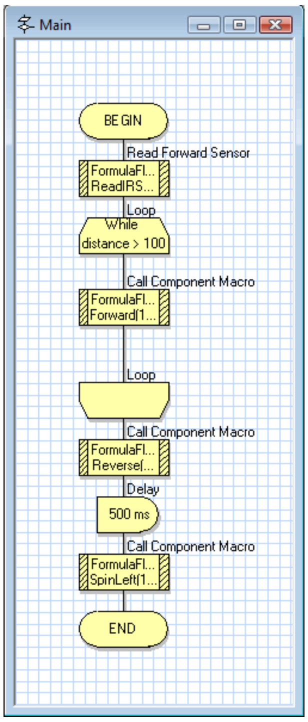


Figure 5 A Flowcode program

The philosophy of the course at this point was to employ a more constructivist approach to learning, allowing the students to tackle the problems in different ways, to see what happened when the programs were tested on the buggies and to explore and generate their own problems that they wished to solve. It was hoped that this approach would engage the students and deepen their learning experience. According to Gagnon and Collay (unknown) learners construct their own knowledge on the basis of interaction with their environment in line with the following principles of constructivist learning:

- knowledge is physically constructed by learners who are involved in active learning;
- knowledge is symbolically constructed by learners who are making their own representations of action;
- knowledge is socially constructed by learners who convey their meaning to others;
- knowledge is theoretically constructed by learners who try to explain things they do not completely understand. (Gagnon and Collay, unknown).

The students were to use this experience to help with the remainder of the course which was delivered using the more traditional instructional approaches of lectures on core programming theory, followed by activities to practice and appropriate assessments.

5. ANALYSIS

Feedback and evidence was gathered during the course using questionnaires, observations and staff student liaison. This was used to assess the impact of the use of the Flowcode buggies.

5.1 Questionnaires

The questionnaires comprised of two sections. Section A was an area for students to write comments about the use of buggies on the course with Section B set out as a series of statements against a 5 point scale ranging from strongly disagree to strongly agree. Below are some of comments expressed in section A:

- "… very enjoyable and a good introduction to programming. Should definitely be carried on in future years."
- "I thought the buggy programming was a good introduction into how to set up code and how to give instructions to a computer."
- "It was a fun and interesting way to start learning programming."
- "I enjoyed using the buggies. It helped me to gain some understanding of programming in a particular way."
- "Using the buggies was an interesting introduction into programming which wasn't too difficult to understand. All in all it was a useful and enjoyable start to the course which allowed us to actually program without prior knowledge and to learn the format of programming in the process."
- "I really enjoyed using the buggies. I found them a simple and more interesting way of introducing us to programming. I understood how to work the drag and drop functions and testing programs out. Using the actual buggies was all in all a fun way to program. I think using these next year would be great for next year's students because if they have never programmed before they may be able to understand the logic of programming by using buggies and Flowcode."
- "It would beneficial to have more buggies available."

Below are the responses to the statements used in Section B. The percentages show the students who selected agree or strongly agree to the statements. None of the students ticked disagree or strongly disagree for any of the statements.

The buggies were a good introduction to programming	91%
I enjoyed using the buggies	100%
The software was easy to use	100%
The buggies helped me understand the basics of programming	82%
I would have like to do more with the buggies	64%
The Flowcode activities help me to get to know people in class	73%
This is a good way to learn programming	91%

Figure 5 Table of questionnaire results

It is evident from the comments quoted from Section A and the responses from Section B that the students found the use of the robots to be a positive experience.

5.2 Student Feedback

Staff student liaison meetings were held throughout the year for all courses. The feedback received from the Computing Fundamentals students was positive about the use of robotics. In particular they commented on how it improved engagement and group collaboration. They indicated that they would like to see them used for a larger part of the course next year.

5.3 Staff Observations

From staff observations this proved to be a successful approach for a variety of reasons. As students tackled the portfolio questions they were able to see instantly from the buggy's actions whether their solution was correct or not. It allowed them to play with various parameters and options to see what affect this would have on the solution. Working with the robotic buggies and the Flowcode software encouraged collaboration as students would work together to come up with different solutions to more complex problems such as making the buggy negotiate a maze. It encouraged them to formulate their own problems that they wanted to solve, e.g. negotiating paths of different shapes and lengths whilst avoiding objects. It proved to be a very successful approach at motivating and engaging the students.

The process also allowed the students to focus on the problem solving, flow charting and the algorithmic design aspects of programming as the Flowcode interface presented the students with a graphical interface where icons can be dragged onto the screen to construct a flowchart that solves the problem. This was very useful in avoiding the complexities and "hang ups" with students worrying about grammar and syntax. It also allowed students to explore the logical aspects of programming in a visual way. If the Flowcode programme ran and did not have syntax errors but rather logical errors then the buggy would behave in an unexpected way. This reinforced the logical approach needed when constructing a solution to a problem.

Below is a summary of some of the benefits that using the Flowcode buggies had:

- Engaged all students in core tasks whilst providing opportunities for more advanced students to work on advanced exercises or even to formulate their own problems.
- Helped to mitigate the abstract nature of programming (Dijkstra, 1989) by providing a richer tangible real world environment for seeing feedback. The PieEater graphical user interface does provide feedback but is limited in its interaction to mouse and keyboard inputs and the output is restricted to visuals on a monitor.
- Encouraged experimentation and exploration e.g. use of sensors. In the activities with the robot buggies this ranged from simply providing a correct solution to exploring modifications to their (correct solutions) due to environmental factors (e.g. surface friction, light affecting the robot's sensors).
- Provided visualisation of implementations of solutions to problems.
- Provided a real world environment for learning about programming concepts and designing more robust code e.g. do not want robot to fall off the work table, collision with walls.
- Assisted learning vocabulary of programming in an interesting way.

Although the experience proved to be positive for the students there are some practicalities that need to be considered. These are summarised below:

- To use the robotic buggies on a wide scale you need to purchase a sufficient number of buggies and batteries.
- Time must be set aside for preparation e.g. charging and fitting the batteries.
- Space is required in the classroom to utilise the buggies in a meaningful way.
- There is the possibility for more things to go wrong with the buggies and the connections from the computer to the buggies. This is not a problem with a purely software based solution.

6. CONCLUSION

Previous research (Whitfield et al., 2007) suggested that students with lower mathematical skills can learn problem solving and programming if provided with appropriate materials and the use of less complex visual tools (e.g. Java Trainer) before using more complex IDEs. The Flowcode IDE and buggies are more complex than those discussed in Whitfield et al. (2007) e.g. the Flowcode IDE uses appropriate technical language. This may demonstrate that it is not necessarily the complexity of the tool used but rather the engagement and learning approach taken that is important.

This research demonstrates that the introduction of robotics encourages a constructivist environment in which to learn, it encourages engagement, collaboration and exploration. It motivated the students to go beyond the tasks set for the course as they wanted to formulate problems to explore “what if”. It enabled the students to take control of their own learning and ask questions of themselves and their peers rather than immediately seek solutions from the tutors. The main incentive for the students is the visual and tangible output of their solutions. They are able to receive instant feedback and interact with the buggy. The solutions that they create are influenced by real world factors e.g. friction, space.

This research was conducted with a small group of students. However initial findings were positive both from the students’ perspective and from their results. Their approaches to problem solving seemed to improve along with their overall performance. It is proposed that the use of robotics be used more widely in this course and other programming courses within the department. The practicalities listed above would need to be taken into consideration to ensure the smooth delivery of classes, particularly with large cohorts of students

We acknowledge all registered trademarks within this paper.

7. REFERENCES

Anderson P B, Bennedsen J, Brandorff S, Caspersen and M E, Mosegaard J, (2003), Teaching Programming to Liberal Arts Students – a Narrative Media Approach, *ITiCSE’03*, June 30 – July 2003, Thessaloniki, Greece.

Alice, (2010), Alice Project, http://www.alice.org/ (date accessed 8.Jun.2010).

Beaumont C, and Fox C, (2003), Learning Programming: Enhancing Quality through Problem-based Learning *LTSN-ICS* conference paper, August.

Bennedsen J, Caspersen M E, (2005) Revealing the Programming Process *SIGCSE’05*, February 23–27, 2005, St. Louis, Missouri, USA.

Burton P J, Bruhn R E (2003) Teaching Programming in the OOP Era*, ACM SIGCSE Bulletin*, Reviewed Paper, Volume 35, Number 2 (June 2003) ACM Press.

Cooper, S, Dann W, and Pausch R, (2003), Teaching Objects-first in Introductory Computer Science, *Proc. 34th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE ’03, 191-195.

Culwin F, Adeboye K, Campbell P, (2005) POOPLE (Pre-Object Oriented Programming Learning Environment) prototypes (unpublished).

Dijkstra E W, (1989), On the Cruelty of Really Teaching Computing Science, Comm. *ACM 32*, pp 1398-1404.

Doswell J T, Mosley P H, An Innovative Approach to Teaching Robotics (2006), *Proc. IEEE International Conference on Advanced Learning Technologies*, Sixth IEEE International Conference on Advanced Learning Technologies, ICALT'06, 1121-1122.

Formula Flowcode Datasheet, (2007), Matrix Multimedia, http://www.matrixmultimedia.com/datasheets/HP794-60-1.pdf, (date accessed 8.Jun.2010).

Flowers T R, and Gossett K A, (2003), Using Robots and Simulation to Teach Problem Solving in an Introductory Course in Computing and Information Technology, *Proceedings of the Advanced Simulation Technologies Conference*, 2003, Orlando, Florida, March 30 - April 3.

Gagnon G W Jr, and Collay M, (unknown), Constructivist Learning Design,

http://www.prainbow.com/cld/cldp.html, (date accessed 10.Jun.2010).

Gandy E G, (2010), The use of LEGO Mindstorms NXT Robots in the Teaching of Introductory Java Programming to Undergraduate Students, ITALICS Volume 9 Issue 1 February 2010.

Herterich G E, (2004) One Day Workshop: Mathematics for Computing: 17th November 2004 University of Birmingham.

Jenkins T, Davy J, (2001), *Diversity and Motivation in Introductory Programming*, ITALICS 1(1).

Jonassen D H, (2000) *Computers as Mindtools for Schools: Engaging Critical Thinking* (2nd Edition), Allyn & Bacon.

Lawhead P B, Bland C G, Barnes D J, Duncan M E, Goldweber M, Hollingsworth R G, Schep M, (2003), A Road Map for Teaching Introductory Programming Using LEGO Mindstorms Robots, *SIGCSE Bulletin*, 35(2): pp 191-20.

Matrix Multimedia, (2010), Formula Flowcode, http://www.matrixmultimedia.com/Formflow-X.php, (date accessed 8.Jun.2010).

McCracken M, Almstrum V, Diaz D, Guzdial M, Hagan D, Kolikant Y, Laxer C, Thomas L, Wilusz T, (2001), A Multinational, Multi-institutional Study of Assessment of Programming Skills of First-Year CS Students. SIGCSE Bulletin 33(4).

McDermott R, Eccelston G, Brindley G, (2007), More than a Good Story – Can You Really Teach Programming Through Storytelling?, Proceedings of the 8th Higher Education Academy Information and Computer Sciences Conference (HEA-ICS), Southampton, UK.

Morrison M, Newman T S, (2001), A Study of the Impact of Student Background and Preparedness on Outcomes in CS1, Proc. 32nd SIGCSE Technical Symposium on Computer Science Education, SIGCSE '01.

Oldehoeft R R, Roman R V, (1977), Methodology for Teaching Introductory Computer Science, ACM SIGCSE Bulletin, Proceedings of the seventh SIGCSE technical symposium on Computer Science Education, Volume 9, Issue 1, ACM Press.

Turner S, Hill G, (2007), Robots in Problem-Solving and Programming 8th Annual Conference of the Subject Centre for Information and Computer Sciences, University of Southampton, 28th – 30th August 2007, pp 82-85.

Whitfield A K, Blakeway S, Herterich G E, Beaumont C. (2007), Programming, disciplines and methods adopted at Liverpool Hope University, Italics Vol 6, issue 4, 2007.