Introductory Programming Workshop for Children Using Robotics

Jacky Baltes and John Anderson
Autonomous Agents Laboratory
University of Manitoba
Winnipeg, Manitoba
Email: {jacky,andersj}@cs.umanitoba.ca

Abstract

Teaching programming concepts to children is a challenging task: to do so effectively requires both curriculum that is organized in a manner that assists children to learn the concepts involved easily, as well as making the content motivating and accessible. While many of the abstract concepts involved in programming are not easily grasped by children, physical metaphors have been used in the past to improve the learning process. Moreover, children are fascinated by the manipulation of physical objects and the creation of physical artifacts. We combine these ideas to use robotics to teach children both the elementary concepts involved in programming (e.g. control flow), as well as some concepts that are often difficult for university students to grasp (e.g. parallel processing). This paper overviews our techniques for accomplishing this through the design of a one-day workshop to introduce children to programming using robotics.

1 Introduction

Teaching programming concepts to children is a challenging endeavour. The particular difficulties of teaching programming at both the high school and undergraduate university level have been well noted (see [4] for a good overview): even the most basic concepts such as variable manipulation have been shown to be very easily misconstrued. From the point of view of teaching programming to children, the core task of designing effective exercises and methods of curriculum presentation must consider the fact that children learn differently than the adolescents and adults for whom programming courses are typically designed. Moreover, there are additional challenges when working with children: a lack of world experience and mathematical knowledge make some traditional computer science metaphors difficult to relate (e.g., recursion or control flow), for example. More importantly, a shorter attention span means that exercises must be well-timed and to the point, and contain some stimulus to continue. The curriculum must also be presented in a manner that is both motivating and entertaining.

At the same time, children are fascinated by manipulating physical objects, especially those they can put together into new creations: the simple act of creating a physical artifact brings forward great joy and enthusiasm. While this is also true for many adults, we believe this is an important focus for educating children, and one that we can leverage to improve how we teach programming.

In this paper, we describe our experiences with the use of robotics to bridge the gap between the abstract world of computers and the physical world that children relate to so well. Some of the strong points that we see in employing robotics for this purpose can also be seen in earlier tools for teaching programming to young people. The physical metaphors used in Logo [9], for example, have previously been shown to be of great use in illustrating elementary programming concepts. Turtle graphics, in particular, have been used in numerous products (dating back to Apple's *HyperCard* and earlier) intended for use by those with little programming experience. We take these ideas further, using real mobile robots that serve as a more obvious physical metaphor, and place these in more complex physical situations than can be easily reproduced using turtle graphics. This allows us to teach not only basic control structures such as selection and iteration, but more complex concepts commonly found in robotics, such as asynchronous events.

We have found the results of employing mobile robots as a medium to introduce programming to children to be very positive: children are captivated by the robots themselves, and are extremely motivated. Consequently, we find that they are more able to grasp the essence of exercises presented to them, and come away with an excitement for robotics in general as well as the core computer science concepts that underly programming. Moreover, the sense of command that young children obtain from seeing their work directly affect the operation of an autonomous entity provides an internal locus of control that is an important motivator in learning [6]. This effect may be strengthened by the tendency to anthropomorphize robots, thereby giving one an even greater sense of personal control.

The main difficulty in employing robotics for the purpose of teaching children is dealing with the sophistication necessary to complete running robotics examples. Indeed, this difficulty is also significant even for university-level students [2]. In order for a demonstration of something as simple as a basic line-tracker to operate, for example, perception and control methodologies must already be in place, and the hardware chosen must be amenable to the students' skills at loading and running code. While a PDA might be useful as a computational platform for university-level students, for example, it would be a very poor choice for children.

The process of adapting robot architectures and approaches for educational purposes, to allow for the gradual exposure of students to more sophisticated concepts, is itself a very interesting one. For university-level students, we use increasingly complex software agents as an abstraction, to deal with the process of controlling robotic devices, along with a collection of such agents to manage interfacing with hardware (e.g. for collecting a series of agent responses to be sent to robots on the field via infrared) [2]. Such a setup requires many thousands of lines of code, and while it is possible to provide much of it to students directly and ask them to write simple behavioural additions (as is done in Jujin soccer software), it is very difficult to grasp the necessary details of the system as a whole. Such a grasp is necessary to make the educational experience meaningful, and, since much of the system would not be built by the children, the same sense of control and accomplishment cannot be expected.

For children, we have found a customized version of the Lego MindStorms package to be the most useful. It is easy to construct and modify a robot, and downloading a program to the MindStorms RCX and running the program is relatively straightforward. The MindStorms provide the visible stimulus to both capture their attention and illustrate the results of changes to programs, and we have also found the familiarity surrounding

Lego MindStorms to be of advantage in the comfort level of children. The analogy to basic Lego pieces makes construction more inviting, and it is possible for children under the right conditions to build a complete, working robot without substantial assistance from adults. This also allows us to present some of the ideas involved in physically constructing robots, in addition to basic programming. Finally, an approach using an affordable, consumer-level product allows children the option of continuing their explorations at home.

This paper describes the design and delivery of a series of workshops that employ robotics, in a very structured fashion, to introduce children to programming. Our workshop design reflects modern robotics work and modern educational approaches by advocating a constructionist approach, where simple examples are built into a complex application, allowing children to gather new skills in programming along the way. We present the plan for the workshop itself in a general form that can be adapted to groups with different needs, and also include details of particular choices made for the group of children for which the first workshop was designed. We also describe the advantages and disadvantages of choices we made, and take note of anecdotal evidence of the utility of various aspects of the first workshop to its participants. This is done both to provide evidence on the workshop's effectiveness, and to allow those planning future workshops using this as a basis to better adapt elements of our ideas to children with different needs or abilities.

2 Sample Participants

The strategies described here were first employed for a group of gifted children, specifically the "Explorers' Club," in the Auckland, New Zealand region [1]. The club is the Auckland chapter of the New Zealand Association for Gifted Children (NZAGC).

The average age of the children that participated in the workshop was approximately 12, but the age ranged from 8 to 16. The large age difference made the design of the workshop particularly challenging. The fact that the children were all members of this club also had an impact on the material that was presented as well as the speed at which the material was presented. The workshop design was intended to appeal to children of approximately age 12. However, because of the use of the robotic paradigm, the breadth of this range can be stretched considerably: younger children can grasp some of the rudiments while being mainly captivated by the moving robots, and in the case of gifted children, can grasp much of what could be presented to the average 12 year old. Similarly, older children and those with prior programming experience are equally interested in the application of what they already know to a new and exciting application such as robotics. No particular consideration was given to attributes such as gender in order to target the appeal of the workshop: we considered only age and the abilities we could likely expect at that age.

The children were in general highly motivated. After setting a date for the workshop, many questions from children who wanted to get a head start were received. For example, one 15 year old child contacted the instructor in an email message: "I heard that we will use NQC. I searched for it on the Internet and downloaded NQC. What should I do next?" Most professors would be only too happy if their first year programming class would show such initiative and motivation.

This anecdote also serves to show the keenness with which we see mobile robotics to be typically received by children in general. The robotics information session had 140 registered participants and was one of the most popular of all the Explorer's club activities in that year. The workshops were scheduled on Saturdays in the University of Auckland Robotics

Lab and lasted from 10:00 am to 5:00 pm, including a lunch break. This is obviously much too short a time to teach young people all the intricacies of programming. At the same time, however, it is a very long time from the standpoint of a child's attention span. In our case, the intent was to present the participants with a challenging and motivating problem, and to show them how this problem can be solved with a computer and robots.

The problem was done in a series of steps, partly to ensure that the children had adequate breaks and that attention was properly maintained, but also to ensure that the problem was broken down into digestible pieces and that there was time to see the intricacies in its various pieces. At the end of the workshop, the children had learned enough to be able to take their programs and modify them to perform more sophisticated programming at home. The choice of a platform that was easily obtainable by the average consumer was an important part of this. The ability to perform further work immediately at home was also a major motivator for the older children. From feedback of parents, this worked well. Quite a few children requested Lego MindStorms for their birthdays.

There was a limit of 20 participants for each workshop. Since only 10 Lego MindStorms sets were available, the children were put into groups of two, and one adult was assigned to each group as a mentor. Working in a group is advisable for reasons other than scarcity of equipment, however. We have found that children get more out of such a workshop when they can reinforce one another in terms of individual strengths and weaknesses, especially in a group such as this one where the age range is disparate. Group work also helps to alleviate some of the complexity inherent in robotics as well. The selection into groups was performed by the organizers and supervisors of the club, since they were familiar with the personalities of the children as well as the requirements of special needs children (for example, there was an autistic child who could not be paired with other children, and would accept only certain adults as mentors).

Choosing a robotics research lab as a location had both advantages and disadvantages. Being a working robotics lab, many projects are of course under development at any one time. The presence of the children meant that very limited work could be done by the graduate students who would normally be working in the lab - this is not something that could be done with great regularity in such a setting without an impact on lab output. On the other hand, the fact that other projects are going on in the lab is also useful in that they can serve as a quick demonstration if anything goes wrong and requires a few minutes to fix (a situation that should always be prepared for when working with robots). This did not prove necessary during the first workshop, but graduate students that dropped in did demonstrate their projects to the children during spare moments, which proved to be motivational to all involved and also helped deal with the issue of short attention spans and a long workshop. A lot of external activity would be very distracting to the children, however (indeed, the presence of interesting pieces of equipment also caused attention to stray on its own), and so an area accessible to the laboratory but distinct from it would be the likely best choice for such a workshop from this standpoint.

More significantly from the perspective of preparing for a workshop, all of the computers in the laboratory run Linux. As would be expected at this level, none of the children or mentors had any previous experience with Linux. However, there are numerous good GUIs and simple text editors available for Linux, and an overview of these was the first order of business for the workshop. After a quick introduction, the children were able to login, open and save a text file and download a pre-written control program to the Lego RCX brick. The exposure to Linux was motivating to some of the children, and in general they were excited at being able to use the equipment that would ordinarily be used by the residents of the lab.

3 Course Outline

The stated purpose of the workshop was an introduction to programming using robotics, to be delivered in such a manner that it would be both enlightening and enjoyable.

As stated in the previous section, the platform chosen was the Lego MindStorms robotics kits, mainly due to the ease with which they can be obtained and used by children. Lego MindStorms come with their own visual programming language, which runs under Windows. However, this language is not powerful enough to present more complex programs. Since it is a visual programming language, it is also very different from more common programming languages that children may be interested in (e.g., C++, Java, PHP, etc.) or are likely to use in future. Because of this, we elected to use David Baum's *Not Quite C* (NQC) language [3] as an alternative to provide a common imperative programming language. Its syntax is similar to C, but the language also includes additional constructs. These constructs are useful in this application to simplify using the sensors, actuators, and especially for the handling of asynchronous events.

The task around which the workshop was structured was the construction and programming of a robot that is able to find a line in the environment and then track this line. Note that this is a fairly complex problem. For example, Lego includes the code for a line tracker in its MindStorms set, but the code only works if the robot starts already on the line and is only able to track right hand curves. At the end of the workshop, a small competition is planned, wherein children would program their robots to try and follow a line.

In order to deal with the complexity surrounding a working robot, and to segment the time into reasonable pieces for the retention and general attention span of the children, we partition the activities surrounding the construction and programming of this robot into five segments. Each of these is described in the subsections that follow.

3.1 Construction and Hello World

During the first 10 minutes of the workshop, the children are given a demonstration of how to use the computers in the lab. This includes the login procedure (accounts were set up and configured as necessary prior to the start of the workshop), opening and saving a textfile in kate, and running nqc from the command line. A simple program to move a robot forward and backward with a short delay between commands - essentially *hello world* for a robot - is provided for the children as a first step (Figure 1). This allows us to teach them the process of compiling and loading a program without having to learn to program as well at this stage. Source of this and the other code described here can be downloaded from *http://www.cs.umanitoba.ca/~jacky*.

Note from the code in Figure 1 that we begin at a simple but reasonably low level. As opposed to the forward and backward instructions that would be common to a Logo-based approach, we illustrate motors being turned on, left to run, and then turned off. Before actually beginning any coding of their own, the children are thus exposed to the idea that what they think of as basic commands are in fact sequences of more basic operations, a fact that is difficult to illustrate in most computer applications. The children can now begin thinking at the level of robot motion when considering what they want their programs to accomplish.

Figure 1: *Hello World* for a Robot.

```
task main() {
  OnFwd(OUT_A);
  OnFwd(OUT_C);
  Wait(400);
  Off(OUT_A);
  Off(OUT_C);
  Wait(500);
  OnRev(OUT_A);
  OnRev(OUT_C);
  Wait(200);
  Off(OUT_A);
  Off(OUT_C);
}
```

After gaining the ability to compile and run a robotic program, the children then spend about 45 minutes building robots out of the provided Lego MindStorms kits. The basic design was a simple differential drive robot with a light sensor mounted on the front , and the children were shown two sample designs in order to assist them (Figure 2).

In the explorers' workshop, many children customized their robots or made modifications to them during the remainder of the day, so the designs were by no mean complete at this stage. This is a very useful phenomena to see emerging from the point of view of robotics education, since it illustrates that the the children have learned that designs must be adapted to the world in conjunction with the activities the robot is expected to engage in.

3.2 Sequence: Turns and Straight Lines

The first real programming task for the children involves implementing simple turns and straight lines with their robots. The children are shown in detail the commands to set the direction of a motor, to turn a motor on and off, to wait for a specific amount of time in order to allow the motor to turn and affect the robot.

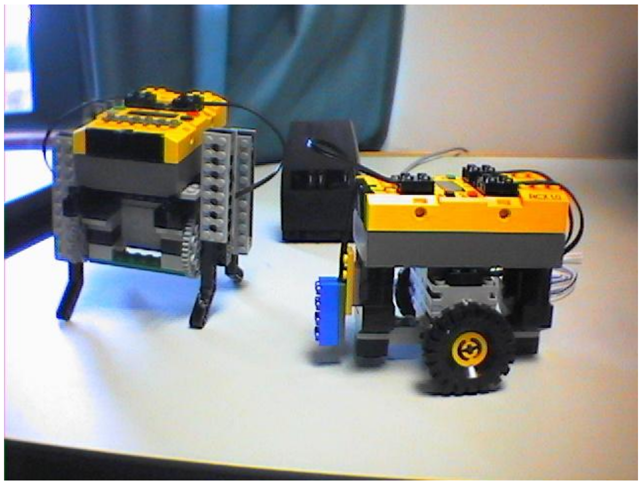
Simple code that drives the robot in a straight line is then described and made available to them. The children are then asked to change the code such that:

- the robot drives for 10cm in a straight line,
- the robot turns to the left/right by 10 degrees.

This introduces the children to the concept of a *sequence*. Children in the explorers' workshop quickly realized that the commands in the program were executed one after the other. The children also learn about the relationship between the time the motors spend on and the distance travelled by the robot, since they must adjust the program so that the robot turns an appropriate amount by leaving the motor on the (approximately) correct amount of time.

These routines are then encapsulated so that the children can call on them to build more complex routines. Thus, this also serves to introduce the children to simple subroutines. All variables were declared globally, allowing issues of parameter passing and variable scope to be avoided.

Figure 2: Sample Lego MindStorms Line-Tracking Robots.



3.3 Conditionals, Iterations, and Variables: Squares

Having some understanding of sequence and the basic ability to move the robot forward and turn it to the left and right, we then turn to using these concepts to illustrate basic control structures. The simplest of these is to create larger patterns (e.g. squares) from single line segments (Figure 3).

This is then extended from a constantly-specified iteration value to illustrate the use of a variable in such a structure, and ultimately to a loops controlled by termination conditions (Figure 4). Following this, the use of random numbers is shown to the children (by changing the square construction code to random straight ahead/right/left turn choices), along with a basic demonstration of the light sensor that comes with the Lego MindStorms kits. The light sensor returns a numeric brightness value, which can be used to discern when the robot is passing into dark or light coloured areas. In our case, we begin using it in a dark field bounded by white lines, and then ultimately use the sensor to allow the robot to keep following a white line marking a race track perimeter.

3.4 Components of a Line Tracker

Now that the children have some idea of the control structures involved in any program, and have been introduced to the light sensor, the remainder of the workshop is devoted to exercises that directly contribute to the line tracking task. The children are first given an exercise that lets them produce code to move randomly, stopping after a given distance or if the robot leaves the playing field (detectable by the light sensor when the robot passes into a white area surrounding the dark field surface).

This exercise introduces the children to more complex control structures. The first time this workshop was run, we organized this coding around the use of while loops with termination conditions. However, the combination of conditions necessary turned out to be too complex for the children. Even children that already had skills in programming had problems getting the code right the first time. To overcome this problem, the children were

Figure 3: Iteration: making increasingly larger squares.

```
#define TEN_DEGREE_WAIT 3
#define TEN_CM_WAIT     50

task main() {
  /* Make 10 Squares */
  repeat(10) {
      repeat(4) {
         /* Drive 1m then turn 90 o*/
         OnFwd(MOTOR_LEFT);
         OnFwd(MOTOR_RIGHT);
         Wait(10*TEN_CM_WAIT);

         OnRev(MOTOR_LEFT);
         OnFwd(MOTOR_RIGHT);
         Wait(9 * TEN_DEGREE_WAIT);
     }
  }
  Off(MOTOR_LEFT);
  Off(MOTOR_RIGHT);
}
```

introduced to the concept of tasks. Once organized around tasks (checking for the line and moving the robot), the code is reasonably straightforward (Figure 5).

The children took quite naturally took to the idea that things can happen at the same time in the real world, and so they were not surprised that a computer can do more than one task at the same time. This is quite contrary to teaching asynchronous/parallel processing to computer science, who often have difficulty grasping the concept of parallelism and associated pitfalls such as race conditions and dead locks. However, since children have not been taught about basic processor design, this concept was much easier to grasp than complex while loops with terminating conditions. It is also interesting in that organizing programming around tasks and behaviours is a very natural approach in modern robotics, and points to this similarity as being one well worth exploiting for computer science education in general.

3.5 A Full Line Tracker

We then perform a series of exercises that allow the children to combine their experiences thus far into a full line tracking robot. We begin by telling the children that the robot will not be started on the line, but must first find the line and then follow it. It is reasonably easy for them to abstract the task they just performed – randomly exploring a field and stopping on the white boundary that encloses it – to the idea of randomly exploring a field and stopping when a white line is found. It requires only small modifications to the code of Figure 5 to achieve this. More advanced children also make the connection to other previous exercises: for example, driving in squares of increasing size (Figure 4) as a method for more systematically searching the area in which the robot is placed.

Extending this to a complete line tracker then involves adding code to follow a line once

Figure 4: Iteration: variables and termination conditions.

```
/* Where is the right and left motor connected to? */
#define MOTOR_LEFT  OUT_A
#define MOTOR_RIGHT OUT_C

/* Constants for the robot */
#define TEN_DEGREE_WAIT 3
#define TEN_CM_WAIT     50

task main() {
  int square_length=1;
  int square_add=1;
  /* Do this until the square length is less than 0.5m */
  while (square_length < 5) {
      repeat(4) {
         /* Drive forward then turn*/
         OnFwd(MOTOR_LEFT);
         OnFwd(MOTOR_RIGHT);
         Wait(square_length*TEN_CM_WAIT);
         OnRev(MOTOR_LEFT);
         OnFwd(MOTOR_RIGHT);
         Wait(9 * TEN_DEGREE_WAIT);
      }
      /* Make the next square larger by adding square_add */
      square_length = square_length + square_add;
  }
  Off(MOTOR_LEFT);
  Off(MOTOR_RIGHT); }
```

Figure 5: Tasks: stop after travelling for 1m or when leaving the field.

```
task main() {
  start drive_random;
  start check_playing_field;
}

task drive_random() {
  int distance_travelled = 0;
  int rand;
  while (distance_travelled < 10) {
      /* Drive forward 1m then turn*/
      OnFwd(MOTOR_LEFT);
      OnFwd(MOTOR_RIGHT);
      Wait(1*TEN_CM_WAIT);

      rand = Random(3);
      if (rand == 0) {
           /* Right Turn */
           OnRev(MOTOR_LEFT);
           OnFwd(MOTOR_RIGHT);
           Wait(9 * TEN_DEGREE_WAIT);
      } else if (rand == 1) {
           /* Left Turn */
           OnFwd(MOTOR_LEFT);
           OnRev(MOTOR_RIGHT);
           Wait(9 * TEN_DEGREE_WAIT);
      } else {
           /* Straight ahead */
           OnFwd(MOTOR_LEFT);
           OnFwd(MOTOR_RIGHT);
           Wait(1 * TEN_CM_WAIT);
           /* Add distance travelled */
           distance_travelled = distance_travelled + 1;
      }
  }
  Off(MOTOR_LEFT);
  Off(MOTOR_RIGHT);
}

task check_playing_field() {
  /* Initialize the light sensor */
  SetSensor(LIGHT_SENSOR, SENSOR_LIGHT);
  SelectDisplay( DISPLAY_SENSOR_1 );

  while( true ) {
      if (LIGHT_SENSOR >= 42) {
         stop drive_random;
         PlaySound(SOUND_DOUBLE_BEEP);
         break;
      }
  }
  Off(MOTOR_LEFT);
  Off(MOTOR_RIGHT);
}
```

it is located. A simple approach was to add another task (*drive_forward*) which would be started by the *check_playing_field* task when it detected the line.

After adding this task, the children were asked to add a task that would search in increasing amounts to the right and to the left if the robot was not on the line. This allows the robot to find the line again whenever it leaves the track (due to the line bending away from the robot's current path, for example). The addition of this task (*search_line*) results in the code shown in Figure 6

Without a doubt, this is the most difficult part of the workshop. We also put particular emphasis on the fact that the children should work on this problem by themselves given the knowledge supplied thus far without any help from adults. This was apparent in the success rate: all groups were able to implement the code to find a line and then stop. About one-third of all groups were able to complete the last most challenging part of the workshop. However, since the children were informed that this was the most difficult part, they were not disappointed if they did not complete the final part of the workshop. Moreover, it gave them a challenging task they could take home with them as well.

Some of the children realized that this would be too difficult for them and chose to implement some other variation of previous tasks instead. For example, one group decided to program their robot so that it would drive an eight pattern and stop when it hit a white marker. From the point of view of constructionism, this is also a positive result.

After the exercises are complete, we run a competition to allow the line-trackers to operate together. The children placed their robots in the middle of the field, and the robots were expected to find and track a line (which was a wandering path around the field). However, the robots are not timed and their are no prizes for the teams. The major intent is to instill in the children not only a sense of accomplishment, but the idea that even after a program will run, we can judge its work as more or less effective than some other design. This is especially true in robotics, where the hardware and software design can differ significantly from robot to robot.

Figure 7 illustrates some participants in various stages of the workshop: receiving the initial explanation of what they are about to do; building their robots; and finally examining the performance of their creations.

4 Discussion

This workshop design employs a very hands-on, constructionist approach [10, 5] to introduce programming to children. In contrast to other workshops for children using Lego MindStorms, the focus here is not on the construction of the robot, but on the art of programming. The manner in which exercises are separated in our approach allows for exploratory learning, but at the same time provides limits so that the children are not overloaded, a factor that has been mentioned previously as necessary in exploratory learning [7].

We have found that this workshop provides children with the confidence to start exploring programming on their own. Although not every group completes the final line tracker, they all understand its basic components, and everyone winds up with a robot that can do some interesting activities.

Although the children in the Explorers' Club are certainly not a random sample of all schoolchildren, we believe the results experienced in these workshops can be achieved in other groups as well, given the ability to modify the timing and speed at which new material is presented. At the same time, while this workshop is intended for children, others have

Figure 6: A complete line tracker (constants are omitted and are similar to previous examples.)

```
task main() {
  start drive_random;
  start check_playing_field; }

task drive_straight() {
   ...
}

task drive_random() {
   ...
}

task search_line() {
  int angle = 1; int turn = 1;
  Off(MOTOR_LEFT); Off(MOTOR_RIGHT);
  while (angle_searched < 18) {
     if (turn == 0) { /* Right Turn - do left next*/
          OnRev(MOTOR_LEFT); OnFwd(MOTOR_RIGHT);
          Wait(angle * TEN_DEGREE_WAIT);
          turn = 1;
     } else { /* Left Turn - do right next*/
          OnFwd(MOTOR_LEFT); OnRev(MOTOR_RIGHT);
          Wait(9 * TEN_DEGREE_WAIT);
          turn = 0;}
      angle = angle + 2;
     }
  Off(MOTOR_LEFT); Off(MOTOR_RIGHT);
}

task check_playing_field() {
  /* Initialize the light sensor */
  SetSensor(LIGHT_SENSOR, SENSOR_LIGHT);
  SelectDisplay( DISPLAY_SENSOR_1 );
  while( true ) {
      if (LIGHT_SENSOR >= 42) {
          stop drive_random;
          PlaySound(SOUND_DOUBLE_BEEP);
          break;
      }
  }
  while( true ) {
      if (LIGHT_SENSOR >= 42) {
          stop  search_line;
          start drive_straight;
      } else {
          stop drive_straight;
          start search_line;
      }
  }
  Off(MOTOR_LEFT); Off(MOTOR_RIGHT);
}
```

Initial Explanation — Constructing Robots

Testing — Testing and competition

Figure 7: Images from the Introductory Programming Workshop

pointed out [8] that current approaches to teaching programming at the university level are also outdated, and while multimedia-based approaches are currently advocated by many , we believe a physical metaphor such as robotics would be as motivating for University level novice programmers as it is for children.

One of the secrets to the success of the workshop was that instead of requiring sudden flashes of inspiration, the children can make progress by simply playing with the programs that they had written. In fact, a lot of time was given to the children so that they can play with the robots. This meant that children would also modify the basic construction of their robot or go back to previous programs that they had written during much of the day. Both the time for play and exploration and the ability to turn back to previous problems is a very important part of scheduling a workshop such as this one: it helps the children reinforce what they have learned, and turning to other problems also allows the children to take a small break from their current activity. The latter assists well in dealing with the short attention spans that are the norm in this group: the children break for a few minutes, while still performing activities that contribute educationally, and return again with new energy.

We feel that the complete line tracker is an ambitious goal for a short workshop such as this. To make it possible, the intermediate steps must be very carefully chosen. The break-up of intermediate tasks (drive, drive squares, detect the line) detailed here has worked very well. Interestingly enough, the utility of this breakdown (and the fact that even professionals underestimate the difficulty of making robots do simple things) was demonstrated amply by one of the observers (who was also a computer science professor) at the Explorer's workshop. He had brought along his own Lego MindStorms kit, and instead of taking direction from the workshop instructors, chose to develop his own code. Toward the end of the day, the professor insisted that the light sensor had to be broken in his MindStorms kit, as the final line-tracker would not work. Upon examining his code, it was evident that instead of following solid engineering principles used in the breakdown defined here, and building on tested working code, the professor's code had turned into a 300 line program, which was impossible to debug on short notice. The first author then downloaded the code from one of the children onto the same robot and started the robot, which then tracked the line perfectly. The instructor smiled and simply said "Looks like it is working to me." The look on the professor's face was priceless.

The workshop design proved a great success. We received about thirty email messages from parents after the workshop, which were all very encouraging. More advanced workshops were requested as a result. In addition, the instructors and mentors also benefitted from seeing robotics from the children's perspective, as well as the children's overall enthusiasm.

Because of this positive feedback, the workshop became a semi-annual event and was always booked out several weeks before the event. Even after the first author left New Zealand and moved to Canada, the workshop continued to be taught using the same principles by graduate students.

References

[1] Explorers club homepage. http://www.explorers.org.nz/, March 2005.

[2] John Anderson and Jacky Baltes. Agent-based control in a global-vision robotic soccer team. In *Proceedings of the Agents Meet Robots Workshop, 17th Conference of the*

Canadian Society for the Computational Studies of Intelligence (AI-04), pages 60–68, London, ON, May 2004.

[3] David Baum. Not quite c homepage. http://bricxcc.sourceforge.net/nqc/, March 2005.

[4] Mordechai Ben-Ari. Constructivism in computer science education. In *Proceedings of ACM SIGSCE-98*, pages 257–261, Atlanta, GA, 1998.

[5] Amy Bruckman. Community support for constructionist learning. *Computer-Supported Cooperative Work*, 7:47–86, 1998.

[6] Diane Howard. *The Relationship of Internal Locus of Control and Female Role Models in Female College Students*. PhD thesis, University of Texas at Austin, 1996.

[7] Akihiro Kashihara, Kinshuk, Reinhard Oppermann, Rossen Rashev, and Helmut Simm. A cognitive load reduction approach to exploratory learning and its application to an interactive simulation-based learning system. *Journal of Educational Multimedia and Hypermedia*, 9(3):253–276, 2000.

[8] Jim McKeown. The use of a multimedia lesson to increase novice programmers' understanding of programming array concepts. *Journal of Computing Sciences in Colleges*, 19(4):39–50, April 2004.

[9] Seymour Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, 1980.

[10] Seymour Papert. Situating constructionism. In *Constructionism*. Ablex Publishing, Norwood, NJ, 1991.