Using Surveyor SRV-1 Robots to Motivate CS1 Students

John Cummins[3], M. Q. Azhar[1,2], Elizabeth Sklar[1,2],
[1]Dept of Computer Science, Graduate Center, City University of New York, New York, NY 10016 USA
[2]Dept of Computer and Information Science, Brooklyn College, City University of New York, Brooklyn, NY 11210 USA
[3]Dept of Mathematics, Brooklyn College, City University of New York, Brooklyn, NY 11210 USA
contact authors: jrpcummins@yahoo.com, mqazhar@sci.brooklyn.cuny.edu, sklar@sci.brooklyn.cuny.edu

Abstract

This paper focuses on the development of new software and agent-centric curriculum for teaching introductory computer science (CS1) students, using Surveyor's SRV-1 robot, with the intention of creating an engaging, interactive learning environment. The SRV-1 platform is an interesting contrast to the older and more well-known LEGO Mindstorms RCX. We present our work with the SRV-1 and contrast our experiences with previous work using the RCX.

Introduction

Over the last 5-10 years, there has been a growing trend toward the use of small, simple robots as a hands-on learning environment for teaching introductory computer science (CS). To date, the most popular platform has been the LEGO Mindstorms RCX Robotics Invention System[1], which entered the market in 1999 and has since appeared in primary, secondary and undergraduate classrooms around the world. A wide range of programming environments were developed for the RCX, branching out from the graphical, drag-and-drop puzzle-piece paradigm employed by LEGO's RCX Code to the graphical wired-blocks paradigm of RoboLab (Erwin, Cyr, & Rogers 2000) and a number of text-based extensions to popular introductory languages such as the C-based Not-Quite C (NQC) (Baum 2000) and the Java-based leJOS (Laverde, Ferrari, & Stuber 2002). In 2006, LEGO introduced their second generation Mindstorms robot, called NXT, along with another graphical programming environment called LME. The RCX community has begun to adapt, introducing NXT-based versions of NQC, leJOS and RoboLab (version 2.9).

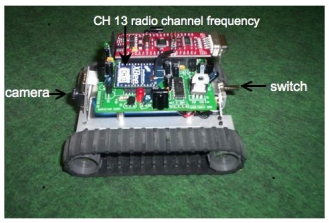
In our previous work, we have used the RCX extensively in CS classrooms, from introductory computer science for non-majors to introductory and advanced programming for CS majors, to artificial intelligence for undergraduate and graduate students (Sklar, Parsons, & Azhar 2007; Sklar, Parsons, & Stone 2004). We have also employed the RCX for outreach programs at the middle and high school levels (Sklar *et al.* 2008; Goldman, Eguchi, & Sklar 2004) and as a demonstration platform for an international educational robotics initiative called RoboCupJunior[2] (Sklar & Parsons 2002). Our results confirm the prior findings of many others: that robotics-based curricula can help motivate students (Stein 1996; Beer, Chiel, & Drushel 1999; Kumar 2004; Weinberg & Yu 2003; Fagin & Merkle 2003; Carbonaro, Rex, & Chambers 2004; Sklar, Parsons, & Stone 2004; Blank *et al.* 2005; Bhave *et al.* 2005). In addition, robotics-based learning environments assist students in developing teamwork, communication and time management skills (Sklar, Eguchi, & Johnson 2002; Sklar, Parsons, & Azhar 2007).

We have been interested in exploring platforms other than the LEGO robots, initially because we found that the LEGO robots are often perceived by undergraduates and high school students as children's toys, thus diminishing their effectiveness as a motivational aid. Additionally, we have been interested in using a platform that provides more powerful visual sensing than is available with either LEGO platform; in particular, we wish to employ a camera sensor in order to provide a larger space for program development and to give students a more realistic experience with robotics.

In early 2007, we acquired a small number of SRV-1 robots from the Surveyor company[3]. Our aim was to use these robots in our introductory programming class, which is taught in C++ and is geared toward CS majors; this is the canonical "CS1" classroom. Our goals here are to provide students with: (1) hands-on activities to create for them an interactive learning environment, (2) an initial experience programming a robot, and (3) exposure to the fields of artificial intelligence (AI) and agent-based systems. The SRV-1 is a well-built tracked[4] robot that resembles a small tank. The SRV-1 has a low-resolution color camera located on its front. The SRV-1 also has several infra-red (IR) sensors that can be used to estimate the robot's distance from obstacles, but our limited experimentation with this sensor produced results that were unsatisfactory. Here, we focus on the use of the camera sensor, which was more than adequate to achieve our aims.

With both LEGO platforms, users create programs on an off-board computer (like a Mac or PC laptop) and then

[1]`http://www.legoeducation.com`
[2]`http://www.robocupjunior.org`
[3]`http://www.surveyor.com`
[4]i.e., the platform has 4 wheels covered by 2 treads, one on each side



SRV-1 platform

XBee dongle

Figure 1: Surveyor platform

compile the program into a language like LEGO Assembler and download the executable code onto the robot. With the leJOS interface, a small Java Virtual Machine (JVM) is downloaded onto the robot, along with leJOS-based byte code, and the robot executes accordingly. With the SRV-1 robot, programs are also created on an off-board computer, but they are also executed off-board, by streaming commands to the robot using wireless communication. The robot runs a tiny server on-board that uses a simple command/response protocol to talk with the robot. This software is provided by Surveyor. The server can also be used by the robot to send sensory feedback back to the off-board computer. The SRV-1 robots that we used initially have an ARM-7 processor that uses its own radio channel and communicates with the off-board computer via a serially connected XBee radio (see Figure 1). The newer SRV-1 robots possess a Blackfin processor with wireless communication hardware and software.

Processing sensory data on the SRV-1 is a different experience from that of other inexpensive robots. The LEGO robots have simple sensors such as the touch (bump) sensor which requires almost no processing: they return one value (1) when in contact with something and another value (0) when not in contact. In contrast, the SRV-1 has a color camera which provides a lot of data that requires considerable processing to become useful. This is intentional. According to Howard Gordon of Surveyor: "That was exactly my goal in creating the robot—to create a device that had to rely on vision (albeit with a lot less neurons) the same way a human does. Humans don't have a built-in compass, GPS or much in the way of odometry, but we seem to manage with visual cues and memory" (Gordon 2008).

The remainder of this paper is organized as follows. In the next section, we describe the software that we developed to provide a simple C++ programming interface designed to be usable by CS1 students to communicate with the SRV-1 robot. Then, we outline two lab assignments that were given to our students and present results of a survey given to students after completing the labs. Finally, we conclude with a discussion and directions for future work.

Software Development

We have developed a software package in C++ that can be used to communicate with the Surveyor robot and, as above, was designed simply, to be used in a classroom by introductory programming students. Note that although the package

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <assert.h>
using namespace std;
#include "SVR.h"

Surveyor robot( ADDRESS );

int main() {
  robot.drive( 50, 50, 100 );
  return 0;
} // end of main()
```

Figure 2: Sample code

was developed and tested for the SRV-1 platform, it could be adopted to other platforms for use in the same type of classroom environment. The C++ package consists of one class, called `Surveyor`, that is intended to hide the details of protocol management and provide the user with a simple interface to the robot's capabilities.

The package is used by first including a header file (`SVR.h`) in the student's C++ source code file. Then the student must declare and instantiate a `Surveyor` object. The student can then use this object's functions to control the robot. An example is shown in Figure 2. The functions in the `Surveyor` class include:

- `drive()`—which tells the robot to move, based on three parameters: the speed of the left and right motors (i.e., the power sent to each) and the duration that the motors should run
- `takePhoto()`—which takes no parameters and tells the robot to take a picture, store it in compressed JPEG format and transmit the bytes of the image back to the off-board computer

During the course of our software development, Surveyor upgraded the SRV-1 from the ARM7 processor to the Blackfin processor and from the XBee radio communication to WiFi. This greatly speeded up everything to do with SVR-1. We had to adjust the package accordingly, so we are currently maintaining two versions of the code: one for the ARM7 processor and one for the Blackfin. The main difference between versions is in the communication methodology; the Blackfin uses sockets to communicate instead of the serial communication used with the XBee radio device. Both versions of our C++ `Surveyor` package are available on-line at our robotics.edu repository[5].

AI-centric CS1 Curricula

We have integrated the SRV-1 robot platform and C++ Surveyor class into our existing CS1 curricula by giving assignments that are relevant to and appropriate for CS1 students.

[5] `http://agents.sci.brooklyn.cuny.edu/robotics.edu`

This section describes two labs that the students have completed.

Robot Lab 1: remote controller

Our first lab explores the CS1 concept of *programmer-defined functions*, by using the robot's motion capabilities. In this lab, students had to write their own functions to create different behaviors (e.g., program the robot go in a square, or triangle or spiral). Students defined three different functions (`square()`, `triangle()` and `spiral()`) and invoked them based on user input. The `drive()` member function of the `Surveyor` class is called by these functions. The left and right track speeds (i.e., first two parameters passed to the `drive()` function) can be in the range -128 to $+127$, and the duration (i.e., third parameter) is in the range 0 to 255. A duration of 0 means "until the next drive command". Any value longer than a few hundredths of a second is probably best done by using a duration of 0 and having your PC do the timing. We gave students the sample program shown in Figure 2 that will move the robot forward one foot.

Robot Lab2: vision processor

The second lab focuses on the CS1 concept of *arrays*, while at the same time experiencing robot sensor processing. When a robot reads and processes sensory input, it can make decisions about what to do based on feedback it receives from its environment. The SRV-1 has a color camera as its main sensor. In the second lab, students started using this camera. We gave the students the sample program shown in Figure 3; this program will keep the robot turning until it sees the orange ball.

We create a new object called `ballColor` which captures the robot's idea of the color of the ball. We included `color.h` to store a constant value, T_ORANGE_BALL, which is used as a reference for recognizing the `ballColor` object. Note that this value is calibrated *a priori* by the instructor or teaching assistant, not by the CS1 students.

One of the aspects of C++ that we illustrate to our students is that objects can be used without knowing everything about them. For the moment we can think of `ballColor` as specifying "orange". Then, the function:

```
robot.setBin( 1, ballColor );
```

tells the robot we are interested in orange things. The robot has ten "color bins", numbered from 0 to 9 and we have set bin number 1 to orange. The function:

```
robot.getCountScan( 1, ball );
```

fills its argument array `ball` with values depending on where the robot sees the color orange.

The Surveyor robot divides its field of vision into an 80×64 grid, i.e., there are 80 pixels horizontally (in the x direction) and 64 pixels vertically (in the y direction). The robot sees this as 80 columns, and the function `getCountScan` will load each element of the array with the number of orange pixels in the corresponding column.

For example, if the robot sees the image shown in Figure 4, then the call:

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <assert.h>
using namespace std;
#include "SVR.h"
#include "color.h"

Surveyor robot( ADDRESS );

YUVRange ballColor( T_ORANGE_BALL );

int main() {
  char buffer[256];
  int  ball[80];
  robot.getVersion(buffer);
  cout << "SRV-1 version "
       << buffer << endl;
  robot.setBin( 1, ballColor );
  for (;;) {
    robot.getCountScan( 1, ball );
    if ( ball[40] > 0 )
      break;
    robot.drive( -40, 40, 3 );
  }
  return 0;
}
```

Figure 3: Sample camera code

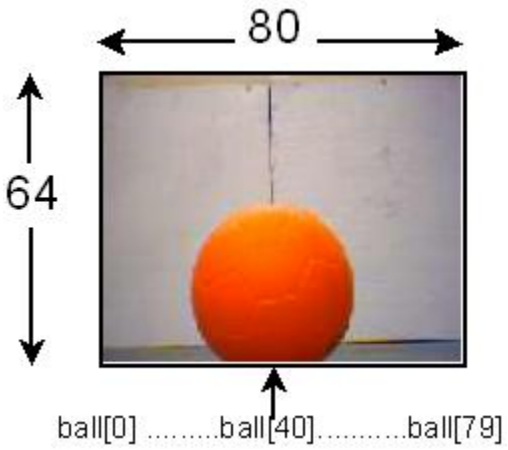


Figure 4: Sample robot view of ball

```
getCountScan( 1, ball );
```

will fill the array `ball` with the following values:

```
ball = 0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  1  4  4  5  3
       5  3  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0
```

The robot does not see any orange in colums 0 through 34, so those array elements are set to zeros. In colums 35 to 41, the robot sees the orange ball and so those array elements are loaded with values greater than zero. Finally, the robot does not see any orange in columns 42 to 79 and so those array elements are set to zeros.

Continuing with the explanation of the program shown in Figure 3, inside the "for loop" we call `getCountScan`, if array element 40 (i.e., the middle, horizontally, of the robot's field of view) is zero, then we turn the robot a bit (note use of the duration parameter to get a small precise turn). When element 40 is greater than zero we break out of the for loop, stop the robot and terminate the program. The effect of this is that the robot will turn until it sees the ball.

Students who finish the basic requirements of the lab can complete "challenges" in the remaining class time. One of the challenges from this lab is to make the robot move around (either randomly or according to a pattern), checking regularly to determine if the robot sees the ball. Here, students are exposed to the challenges of vision processing in AI.

All of our curricula materials and student demos are available at our course website[6].

Survey

We pilot tested the SRV-1 platform, our C++ `Surveyor` software package and the labs described above in one section of our CS1 class, during the Spring 2008 term. There are 15 students enrolled in the class, 2 of whom are female. All students were able to complete the first lab. Eighty percent (80%) of them completed the second lab.

Students were given an informal survey after the lab sessions were over. The survey asked questions about their experience with the robots, whether they liked working with them, whether they felt using the robots distracted them (or not) from learning the C++ language, and whether they planned to continue with robotics-based programming in CS2. Nine students completed the survey. Four of the nine (44%) said that they would continue with robotics-based programming in CS2; interestingly, the ones who said they would not continue all offered reasons like they were transferring to another school or they did not need the course for their major (not all students enrolled in CS1 are computer science majors). Several students commented that programming the robots helped them learn to pay more attention to detail. Quite a few noted that the classroom lab environment helped them get to know their classmates better, which is a positive aspect of the robotics labs. Finally, many said that they realized how complicated it is to program a robot, but were pleased when they could learn how to do so themselves. In general, the feedback was overwhelmingly positive.

[6]http://www.sci.brooklyn.cuny.edu/ mqazhar/teaching/cis1.5

Discussion

We have been very careful about introducing robotics materials to novice programmers because we do not want to overwhelm them. We only exposed our students to simplified robot vision processing and low-level motion commands, believing that a straightforward environment would motivate the students to learn more about robotics and computer science. Our students' reactions more than validated our intuitions. All students in the class completed the first robot lab. Many who completed it early created their own extensions and asked insightful questions.

We believe that our students did so well with the SRV-1 robot, software and labs for the following reasons:

- *Absolute integration:* One of the obstacles of integrating robotics into introductory programming course is a steep learning curve. As novice programmers are already overwhelmed with all new concepts, we do not want to burden them further by making them learn additional, complex robotics materials. We carefully developed the software and curricula to fit in with what students are doing and to relate new materials to what they have done before. We provided them with "just-in-time" information that was relevant for completing each lab.
- *Same programming environment:* Students were able to program the robot using the same C++ programming environment throughout the semester, using either the Mac OS-X terminal (UNIX) window or the PC command window. We introduced two-step processes in order to send instructions to the robot. First, students need to execute a `makefile` at the UNIX command prompt to compile and link the program, and then they call the executable to start sending instructions to the robot. All the students were comfortable with this process.
- *Simplicity of programming:* The `Surveyor` class provided simple, pre-formed examples of many of the concepts that neophyte programmers need to grasp. This allowed the students to become accustomed to using the concepts before having to do the detail work to implement them. For example, the `YUVRange` class encapsulates the robots idea of the color of the ball. Students are exposed to one of the best features of an object-oriented programming language, such as C++, that is, we can use objects without knowing everything about them.
- *Robust communication:* The Surveyor ARM7 robot uses the XBee radio dongle to facilitate communication between the robot and the off-board, controlling computer. One of the primary problems we have faced in the past

when using the LEGO Mindstorms RCX robot was its faulty and flaky communication using an IR communication tower to download programs from an off-board computer to the RCX robot. Here, we had 6 Surveyor SRV-1 robots in a classroom with 15 students, and we had no communication problems.

- *Robust hardware:* The Surveyor robot has a longer battery life than the LEGO Mindstorms RCX robots. It recharges fairly quickly, coming with its own dedicated power adaptor. As well, it has an excellent motor drive.

These are still the early days of this project. The initial reactions from students are encouraging. The challenge presented by the SVR-1 sensor (i.e., camera) is, we think, an excellent introduction to AI. The idea of a robot dependent on vision alone provides challenges appropriate to programmers of all capabilities, particularly when vision processing can be simplified through the use of a class like the one outlined here. Once we started to look at the world as the robot does, it became much easier to put together some useful programs, to create different robot behaviors such driving in a square or finding an orange ball and even playing as a simple game of soccer. This mental change, the ability to see a visual field without preconceptions such as perspective or object recognition was perhaps the most valuable outcome from this project.

Currently, we are developing additional labs to be used in our CS2 class (which will be piloted in Fall 2008). With this in mind, we have extended the `Surveyor` class to provide a doubly linked list of "blobs" (areas of high density of a particular color) so that students can get comfortable with using a list before having to construct one.

Acknowledgements

We thank Howard Gordon of Surveyor.com for technical support.

References

Baum, D. 2000. *Dave Baum's Definitive Guide to LEGO Mindstorms*. APress.

Beer, R. D.; Chiel, H. J.; and Drushel, R. F. 1999. Using autonomous robotics to teach science and engineering. *Communication of the ACM* 42(6).

Bhave, A.; Hamner, E.; Hsiu, T.; Perez-Bergquist, A.; Richards, S.; Nourbakhsh, I.; Crowley, K.; and Wilkinson, K. 2005. The robot autonomy mobile robotics course: Robot design, curriculum design and educational assessment. *Autonomous Robotics Journal* 18(1).

Blank, D. S.; Kumar, D.; Meeden, L.; and Yanco, H. 2005. Pyro: A python-based versatile programming environment for teaching robotics. *ACM Journal on Educational Resources in Computing*.

Carbonaro, M.; Rex, M.; and Chambers, J. 2004. Using lego robotics in a project-based environment. *The Interactive Multimedia Electronic Journal of Computer-Enhanced Learning (IMEJ)* 6(1).

Erwin, B.; Cyr, M.; and Rogers, C. B. 2000. LEGO Engineer and ROBOLAB: Teaching Engineering with LabVIEW from Kindergarten to Graduate School. *International Journal of Engineering Education* 16(3).

Fagin, B., and Merkle, L. 2003. Measuring the effectiveness of robots in teaching computer science. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer Science Education*, 307–311.

Goldman, R.; Eguchi, A.; and Sklar, E. 2004. Using Educational Robotics to Engage Inner-City Students with Technology. In *Proceedings of the Sixth International Conference of the Learning Sciences (ICLS)*, 214–221.

Gordon, H. 2008. Personal e-mail communication (15 Feb 2008).

Kumar, A. N. 2004. Three years of using robots in an artificial intelligence course: lessons learned. *Journal of Education Resources in Computing* 4(3):1–15.

Laverde, D.; Ferrari, G.; and Stuber, J., eds. 2002. *Programming Lego Mindstorms with Java*. Syngress.

Sklar, E., and Parsons, S. 2002. RoboCupJunior: a vehicle for enhancing technical literacy. In *Proceedings of the AAAI-02 Mobile Robot Workshop*.

Sklar, E.; Parsons, S.; Tejada, S.; Lowes, S.; Azhar, M. Q.; Chopra, S.; Jansen, R.; and Rudowsky, I. 2008. Using artificial intelligence to help bridge students from high school to college. In *AAAI Spring Symposium on Using AI to motivate greater participation in Computer Science*.

Sklar, E.; Eguchi, A.; and Johnson, J. 2002. RoboCupJunior: learning with educational robotics. In *Proceedings of the Sixth RoboCup International Symposium*, 238–253. Received Scientific Challenge Award.

Sklar, E.; Parsons, S.; and Azhar, M. Q. 2007. Robotics across the curriculum. In *AAAI Spring Symposium on Robots and Robot Venues: Resources for AI Education*. AAAI Press.

Sklar, E.; Parsons, S.; and Stone, P. 2004. Using RoboCup in University-Level Computer Science Education. *Journal on Educational Resources in Computing.* 4.

Stein, L. A. 1996. Rethinking cs101: Or, how robots revolutionize introductory computer programming. *Computer Science Education*.

Weinberg, J. B., and Yu, X. 2003. Robotics in education: Low cost platforms for teaching integrated systems. *IEEE Robotics and Automation Magazine* 10(2):4–6.