

WEBBOT: A PROJECT TO INSPIRE INTEREST IN ROBOTICS DISCOVERY

by

John A. Ansorge

A THESIS

Presented to the Faculty of  
The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Master of Arts

Major: Teaching, Learning & Teacher Education

Under the Supervision of Professor David Brooks

Lincoln, Nebraska

July, 2006

# WEBBOT: A PROJECT TO INSPIRE INTEREST IN ROBOTICS DISCOVERY

John A. Ansorge, M.A.

University of Nebraska, 2006

Adviser: David Brooks

Combining robotics and education makes possible new and exciting opportunities to motivate, challenge, and inspire youth to learn about science and technology. This thesis reports on one robotics-based educational project called WebBot. WebBot is a robot that can be controlled and observed via the Internet.

Existing research reveals several characteristics that make robots uniquely suited to teaching science and technology concepts. Current implementations of robotics in the classroom are investigated and described. Positive and negative results of these implementations are described and categorized.

The WebBot project's goals and technological architecture are described in detail. WebBot leverages the unique characteristics of robotics described in the literature to encourage children to learn more about robotics, science, and technology. WebBot's objectives are reviewed and connected to the literature and an explanation is provided regarding how WebBot's components work. First, the design and construction of the Lego™-based robot at the heart of the WebBot project is detailed. Next, the physical environment built for WebBot is described. Finally, the software architecture of the

robot, the Internet server, and the Internet client are described. Possible future improvements to the WebBot project are listed and described. The software code used in the project is attached in appendices.

### **Acknowledgments**

Thanks to my parents who always support me in whatever I do. I should also thank my sisters for motivating me by getting their masters degrees first.

Thanks to Dr. Brad Barker for giving me a job and providing material, financial, and psychological support that made this project possible. Thanks also go out to the rest of the people at the Nebraska 4-H State Office for making it a great place to work.

I also want to thank Dr. Sheldon Stick and Dr. Al Steckelberg for serving on my committee. Thanks to Dave Bentz for suggesting to me the idea of a Lego robot controlled by over the Internet.

Finally, I owe thanks to Dr. David Brooks, my adviser, who changed the way I look at technology and education. Without his guidance and encouragement I never would have returned to school and completed this project.

## Table of Contents

I. Introduction .....	1
II. Review of Literature.....	2
Science and Technology Education in the United States.....	2
The Case for Using Robots in Classrooms .....	3
Current Applications of Robots in the Classroom .....	6
Impact of Using Robots to Teach Science and Technology .....	9
Positive Impacts .....	9
Negative Impacts .....	11
III. Results.....	13
Objectives .....	13
Description of WebBot’s Components .....	16
WebBot’s Robot.....	16
WebBot’s Environment .....	18
WebBot Robot Software .....	20
WebBot’s Server Software .....	22
WebBot Web Client Software .....	23
IV. Future Directions .....	27
V. Summary .....	30
References.....	31
Appendix A.....	33
Appendix B .....	37
Appendix C .....	39
Appendix D.....	43
Appendix E .....	44
Appendix F.....	45
Appendix G.....	46
Appendix H.....	47
Appendix I .....	48
Appendix J .....	51

## **I. Introduction**

Robots have long been seen as the means to a future where manual labor is performed by machines and prosperity, productivity, and leisure time for people abounds. This future is still many years away but advances in robotics continue to bring this dream closer to reality. Recent advances have made robots small and cheap enough to be practical for amateurs and classrooms. Research shows that robots can help children transform abstract science and math concepts in concrete real-world understandings through hands-on experimentation. Now, even relatively young children can meaningfully engage in this kind of hands-on experimentation with robots. This new availability presents innovative educational opportunities to teach science, technology, education and mathematics (STEM).

This thesis will report on a project called WebBot that attempts to take advantage of the educational power of robotics. Designed to be part of a broader Nebraska 4-H robotics program, WebBot is unusual in that it attempts to bring robotics-based education to new, untapped audiences by leveraging the broad reach of the Internet. This thesis will first report on some of the theories, methods, successes, and failures of using robotics in educational environments. It will go on to describe the objectives, the design, and the development process of the WebBot educational project. Finally, it will suggest new ways that an Internet-based robotics program can expand to provide even more interesting educational challenges to its audience.

## **II. Review of Literature**

### **Science and Technology Education in the United States**

The United States' economy is highly dependent on advanced technology. Bonvillian (2002) pointed out that Robert Solow of MIT won the Nobel prize in 1987 for establishing that technology and related innovation are responsible for at least half of U.S. economic growth. Industries based on technology need new scientists and engineers every year to help propel their success. It is up to our schools to produce these graduates. Data suggests that the United States is producing fewer science and technology workers, not more (Porter & van Opstal, 2001). Meanwhile, other countries are increasing the number of graduates in science and technology fields.

Currently, U.S. students are less prepared than many other first-world countries in terms of science and math. At the fourth grade level, U.S. students are competitive in science but fall behind most first-world countries in math (Gonzales, Guzmán, Partelow, Pahlke, Jocelyn, Kastberg, & Williams, 2004). By age fifteen, U.S. students are still relatively poor math performers and fall behind the international average in science literacy as well (Lemke, Sen, Pahlke, Partelow, Miller, Williams, Kastberg, & Jocelyn, 2004). If innovation is going to continue to drive the United States' economy, its educational system must improve these scores and entice graduates into STEM careers (Bonvillian, 2002).

Waddell (1999) tells an apt story that illustrates the risks of missing technological opportunities and falling behind competitively. In the 1960s, Joseph Engleberger and George Devol developed state-of-the-art robotic technology in the United States. At first,

they tried to sell the technology to U.S. firms but they found little interest in their inventions. Meanwhile, in Japan the U.S. robotic technology was extremely well received and was eventually sold to a Japanese manufacturing company. Thirty years later there were five times as many manufacturing robots in Japan as there were in the United States.

### **The Case for Using Robots in Classrooms**

One new approach to improving STEM education that is gaining popularity is the use of robots to teach STEM concepts. Advances in technology have brought down the cost of robots and made it easier to bring them into classrooms with tight budgets. Researchers and implementers have suggested several reasons why robots are one of the best ways to teach these subjects.

Seymour Papert (1980) laid much of the groundwork for using robots in the classroom. As far back as the 1970s, he and his team at MIT were pioneering the technological advances that make educational robots practical as well as the educational methods and theoretical foundations that explain how robots can help teach STEM concepts. Many of the ideas that Papert wrote about in *Mindstorms: children, computers, and powerful ideas* were 20 years ahead of their time. Breaking with traditional computer aided instruction models where computers essentially programmed children, Papert wanted to create an environment where children programmed computers and robots. In doing so, the children would gain a sense of power over technology. He believed that children could identify with the robots because they were concrete, physical

manifestations of the computer and the computer's programs. The children learning with robots were able to imagine themselves in the place of the robot and understand how a computer's programming worked. Papert believed that what makes many concepts difficult for children to understand is a lack of real-world materials that demonstrate the concept. He believed that programmable robots were flexible and powerful enough to be able to demonstrate ideas that previously had no easy real-world analogy. For example, the children could see that for a robot to move in a circle, it must move forward a little and then turn a little. This realization helped them see the connections between the abstract geometric theory of angles and a real-world example of them.

Other researchers have also identified the concrete nature of robots as being one of their important advantages. Students can understand abstract concepts and gain a more functional level of understanding by testing scientific and mechanical principles with the robots (Nourbakhsh, Crowley, Bhave, Hamner, Hsiu, Perez-Bergquist, Richards, & Wilkinson, 2005). Students can also learn that in the real world there is not necessarily only one correct answer to every question (Beer, Chiel, & Drushel, 1999). Beer et al. (1999) felt that it was more important for students to come up with creative solutions to problems than it was to recite answers they learned in class by rote. The success or failure of the robots demonstrated what their students learned better than any written exam could. Similarly, Papert (1980) believed that learning with robots helps eliminate the fear of being wrong. He claimed that students using robots learned to learn from their experiences. Rather than seeing failure as something to be forgotten, children see it as more information they can put toward a correct solution.

Another argument for teaching children with robots is that they see the robots as toys (Mauch, 2001). One widely used kit of robotic equipment is made by Lego<sup>®</sup>. Children using this kit can build and program robots out of the same materials they have in their toy chests at home. This makes anything they learn with them seem to be fun as well. Furthermore, it demonstrates that children can learn by playing.

Still another argument for using robots is that they tie into a variety of disciplines. A robot is made up of component parts of motors, sensors and programs. Each of these parts depends on different fields of knowledge like engineering, electronics, and computer science. This interdisciplinary nature of robots means that, when students learn to engineer robots, they will inevitably learn about the many other disciplines that robots depend on (Papert, 1980; Rogers & Portsmore, 2004). For instance, building a sturdy robot requires learning about engineering strong structures. Meanwhile, understanding how to make a wheeled robot move faster requires an understanding of gears and torque. Understanding how gears increase or decrease torque requires an understanding of multiplication. Children learn these basic STEM concepts and, at the same time, they connect these concepts together into a greater understanding of robotics as a whole. Once children develop a sense of how the robotics work, they can apply these concepts as they make the robots interact with the real world. Papert (1980) noted that before a child could program a robot to do a task, he or she would have to learn how to do the task first. The children were motivated to understand the task by their desire to make the robot do their will.

In the same way, teaching students how to build robots teaches them how all of the parts of a complex system interact and depend on each other (Beer et al., 1999). Beer

et al. (1999) point out that this is an important lesson for computer science students who will ultimately need to create software that operates inside much larger systems. They go on to point out that this is as important a lesson to biologists as it is to engineers.

A final practical reason to teach robotics is that the field of robotics is growing rapidly (Waddell, 1999). There are more than 80,000 robots working in American industry and those robots must be installed, maintained, and programmed. As advances in technology make robots more mobile, more capable, and more numerous, the need for workers with expertise in robots will grow.

### **Current Applications of Robots in the Classroom**

Researchers and instructors have incorporated robots into their curricula in a wide variety of ways and for many different age groups. Some approaches have used robots as tools to assist in the teaching of actual programming languages (Fagin & Merkle, 2003; Barnes, 2002). For example, Fagin and Merkle (2003) and Barnes (2002) used robots to help teach the programming languages of Ada and Java™, respectively. The main emphasis for their courses was on teaching the programming languages and basic programming structures rather than the engineering and mechanical aspects of robots. The robots were used to display tangible, physical feedback about what was happening in their programming code (Barnes 2002).

Other courses that use robots are focused on the construction and programming of the robots themselves (Beer et al., 1999; Nourbakhsh et al., 2005). Nourbakhsh et al. (2005) taught a seven-week summer course for high-school seniors using custom-built robots designed specifically for the course. Their course started by teaching the basics of

motors, wiring and simple programming. The instructors used weekly challenges to focus the students' activities and incrementally increase the sophistication of the topics being taught. By the end of the seven weeks, the students had taught their robots to dance, simulate defusing bombs, and navigate mazes. In the process of making their robots do these things, the students were taught about Java™ language programming, robotic sensors, electronics, wireless computer communication and other advanced topics.

Beer et al. (1999) used a similar approach in their classroom but, instead of simply focusing on the programmed behavior of the robots, the students designed and built their own robots. The students were given kits that included more than 1300 pieces. As the class progressed, students were taught both engineering and programming skills. Ultimately, the students put what they had learned to the test and built robots to compete in the final course activity, a robotic egg hunt. For this competition, the robots needed be able to navigate an arena, identify eggs of a particular color, and bring those eggs back to a "nest" faster than the opposing team. To accomplish these goals they had to design, build, and program robots that could locate their nest, sort eggs based on their color, and grab and move them back to their nest. They also needed to avoid the other team's robot or risk being damaged or disabled in an accident. Each of these behaviors was complicated in itself. The real challenge for the students was putting all the pieces together.

Moore (1999) used robots to teach her fourth-grade students several different topics under the umbrella of examining robots. She used the topic of robots as a "hook" to capture her students' attention. Then she weaved other disciplines into this central

theme and asked her students to think critically about robots. She challenged her students to look at the history of robots, discover how robots are being used, what their advantages and disadvantages were, and speculate on how they might be used in the future. She had her students look at robots from a variety of different perspectives. Using robots, she taught subjects ranging from geometry to poetry. To assess how well her students had learned the topics, she gave them a task for a robot to perform and asked them to design, draw, and build a robot out of construction paper.

Some educators have used robotics to teach engineering subjects as early as kindergarten (Rogers & Portsmore, 2004). Using Lego<sup>®</sup> robots and software called Robolab<sup>™</sup> they designed a curriculum to teach kindergarten through fifth-grade students about engineering. Robolab<sup>™</sup> is an icon-driven programming environment that allows the children to create simple programs visually without needing to know mysterious programming code words. Students taught with the curriculum were given tasks to accomplish with their robots that put math and engineering ideas into real world practice.

Some researchers have proposed using virtual reality robots instead of real robots because of the considerable expense of robotics kits (Geissler, Knott, Vazquez, & Wright, 2004). Instead of programming a physical robot, students manipulate and program a virtual robots represented on their computer screens. Geissler et al. (2004) claim that virtual robots have many of the same advantages of real robots while costing significantly less.

## **Impact of Using Robots to Teach Science and Technology**

### *Positive Impacts*

Most of the literature describing the use of robots to teach science and technology reports positive impacts on what their students learned about STEM. The positive impacts fall into six categories.

First, learning with robots is more interesting and improves students' attitudes about STEM subjects (Robinson, 2005; Fagin & Merkle, 2003; Mauch, 2001). In interviews with teachers using robots in after-school programs, Robinson (2005) found that teachers thought their students had more positive feelings and were more aware while learning with robots. The teachers claimed that, when successful in building a robot, students felt good about it. Fagin and Merkle (2003) reported that students taught with robots felt that the robots were "interesting," "fun," and "relevant."

Second, learning with robots helps teach scientific and mathematic principles through experimentation with the robots. Rogers and Portsmore (2004) reported success in teaching decimals at the second grade level by making a robot move for a time between one and two seconds. Through experiments, the students were able to learn decimals on their own. They discovered that 1.4 is less than 1.6 because their robots went further when they told it to travel 1.6 seconds. The also learned that 1.50 is the same as 1.5 even with the extra decimal place because the robot traveled the same distance with each value. Papert (1980) used robots to teach concepts of geometry. He created robots with a pen which could be programmed to draw geometric shapes on paper. His students saw the relationships between programming, mathematics, and movement of the robot.

Third, building and programming robots requires that the students develop problem solving skills (Nourbakhsh et al., 2005, Beer et al., 1999; Mauch, 2001). Beer et al. (1999) emphasized that designing an entire system that was needed to work in the real world required problem solving skills that would serve them well in their future careers no matter what discipline they chose. In a comparison of students' expectations from before and after a robotics summer camp, Nourbakhsh et al. (2005) discovered that students were much more likely to claim problem solving as something they learned after the workshop.

Fourth, female students are more likely to appreciate learning with robots than traditional STEM teaching techniques. Rogers and Portsmore (2004) also observed that females were likely to be interested in robotic activities if they could see a larger purpose for the robots they were constructing. Nourbakhsh et al. (2005) found that, before exposure to robots, the high-school females in their study were significantly less comfortable with technology than the males. After the workshop, the females had increased their comfort level with technology more than the males.

Fifth, researchers reported that their students learned teamwork skills. Nourbakhsh et al. (2005) noted that students mentioned teamwork on their surveys far more often after their exposure to robots than before. Beer et al. (1999) identified teamwork as being one of the important outcomes of their robotics course.

Finally, Wagner (1999) found that using robots as tools to teach science subjects can have the side effect of creating greater understanding of computer programming. Her experiment used robotics as a tool to demonstrate physics concepts. She compared the results of students using robotics-based demonstrators to students who used either

battery-powered or no demonstrators. Students who used robotic demonstrators had an improved understanding of computer programming logic than students using the other methods. She did not find that students who used robots learned the physics concepts better than students using simpler battery powered demonstrators. There was no significant difference in scores between the two groups.

### *Negative Impacts*

While it is quite natural for researchers to want to find positive impacts of the methods they are using, every educational experience comes at the expense of other educational experiences. Sometimes these missed experiences are more important than the experiences offered by new methods. For these reasons, it is even more important to look at negative impacts of new educational technologies and methods. Not all of the results of using robots in the classroom have yielded positive results. In one of the very few quantitative studies that examined the use of robots' effectiveness in the classroom with test scores, Fagin and Merkle (2003) found that robots did not help introductory computer science students learn to program. In fact, the robots were a liability. Fagin and Merkle compared the grades from standard exams given to students who were taught with robots to those taught with more traditional techniques. The students taught using robots had significantly lower scores. Furthermore, course surveys given to the robotics students found that students felt that the robots were hard to work on and time consuming during class. Fagin and Merkle (2003) noted several limitations in their study that may have affected the performance of the students taught with robots. First, because the budget for their computer science courses would not allow for enough equipment for

every student in every section of the course, students could only work on the robots during class time and lab periods. Downloading to the robot and testing them used even more valuable lab time. The traditionally taught students could work on their assignments outside of class for as long as necessary, however, and did not need to wait for their programs to load on the robot. This put the robot-taught students at a significant disadvantage in terms of the time they could be on task.

A second factor was that instructors were teaching the robotics-based curriculum for the first time. Even though steps were taken to account for this, Fagin and Merkle (2003) admitted that the instruction could probably be improved with more experience. They recommended that future efforts to teach with robots should allow the students to check out equipment to take back to their rooms. Alternatively, they recommended that students should at least be given a way to simulate the robots in virtual reality without needing the robotic equipment itself so that they can work on their assignments outside of class time.

### III. Results

#### Objectives

WebBot is a new project that uses robotics to interest youth in STEM learning and teaches STEM concepts. WebBot gives children and adults access to a programmable robot via the Internet. Rather than simply tell the users about robotics, WebBot engages users in an interactive robotics lesson. Users are able to program the robot and see the results of their program in real time. Created as part of a new Nebraska State 4-H robotics program, WebBot promotes and encourages further involvement in 4-H robotics while teaching basic lessons about what robots are and how they are programmed.

At the core of the WebBot site is a Lego<sup>®</sup> Mindstorms<sup>®</sup> robot that can be controlled by users across the Internet. To give the WebBot's users an interesting challenge to perform with the robot, the robot is enclosed in a simple maze. Users are challenged to write a program to navigate from one end of the maze to the other. A simple drag and drop interface allows users to create a sequence of commands that will tell the robot how to navigate the maze. Two streaming webcams allow users to see what the robot is doing at all times. One webcam (Figure 1a) shows users what the robot is doing from the robot's perspective and the second (Figure 1b) camera shows a top-down perspective of where the robot is in its environment.



*Figure 1a: Robot's Perspective*



*Figure 1b: Top-Down Perspective*

WebBot has two primary goals: 1) teach children a few simple lessons about robotics and, 2) spread awareness and promote the 4-H robotics program.

First, the WebBot project will give its visitors some basic information about robots and how they work. WebBot will teach children (and curious adults) who visit the site some basic lessons about what makes a robot different from a computer or a machine. Most children at the age range 4-H is targeting do not understand the subtle differences between machines, computers, and robots. This project will attempt to make the differences clear. This project will also give children the opportunity to program the robot. As Papert (1980) suggests, this should give the children a sense of power over the robots and encourage them to pursue an interest in the Nebraska 4-H robotics program. While robots are gaining popularity in educational settings, many children have never had the chance to interact with one. This project reduces the barriers of cost and availability of robotics systems.

The second major goal of the WebBot project is to spread awareness and interest in Nebraska 4-H's robotics program. In the same way that WebBot teaches children

some lessons about what a robot is, this project also demonstrates to adults who visit the site about how Nebraska 4-H is using robots to encourage interest and investigation of STEM ideas. Nebraska 4-H's robotics program is composed of a newly developed National 4-H Cooperative Curriculum System curriculum and a kit of robotic components from LEGO<sup>®</sup>, called LEGO<sup>®</sup> Mindstorms<sup>®</sup>. The LEGO<sup>®</sup> Mindstorms<sup>®</sup> kit is comprised of 828 parts including axles, gears, motors and sensors. The kit includes a programmable microcomputer with three output and three input ports for controlling sensors and motors. The robots are programmed using a specialized icon-oriented visual programming language called ROBOLAB<sup>™</sup>. The 4-H robotics curriculum contains 28 lessons designed around the Mindstorms<sup>®</sup> kit intended for 4<sup>th</sup> through 6<sup>th</sup>-graders. It begins with simple building and programming challenges; it culminates in advanced robotic programming and engineering topics. Many parents, teachers, and 4-H club leaders have expressed interest in Nebraska 4-H's robotics program but wanted to know more about what it teaches and how it works. Because many of these concepts are new, they can be difficult to explain without actually demonstrating how the robots are built and programmed. The WebBot site serves as a place where interested adults can see how learning with Lego<sup>®</sup> robots works.

What sets WebBot apart from other sites with similar goals is the opportunity users have to work with an actual robot. One of the frequently-mentioned advantages to robots is that they make abstract concepts concrete. This project attempts to leverage this advantage to accomplish both of these first two goals; children can learn how robots work by interacting with a robot and parents and teachers can learn how robots are used to

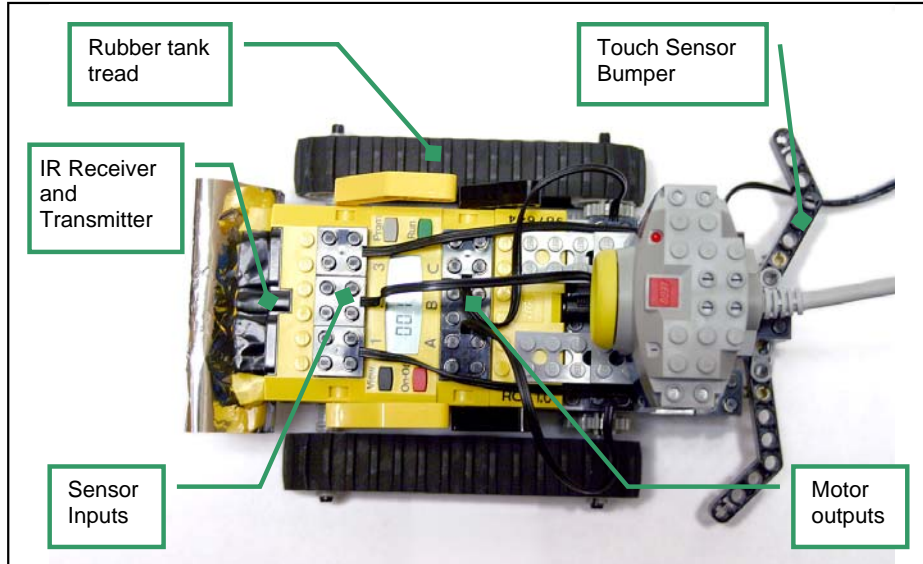
teach by seeing and completing example lessons. For each group, controlling the robot directly will provide concrete examples of the ideas the project is trying to teach.

### **Description of WebBot's Components**

#### *WebBot's Robot*

The robot at the heart of the WebBot site is built with the same Lego<sup>®</sup> components that children participating in the Nebraska 4-H robotics program use. It was important to keep this continuity in order to show both children and adults what learning with Lego<sup>®</sup> robots is like. The design of the robot is based on the Trilobot design found in *Core LEGO<sup>®</sup> MINDSTORMS<sup>®</sup> Programming* (Bagnall, 2002). Some alterations were made to help WebBot automatically navigate in its environment.

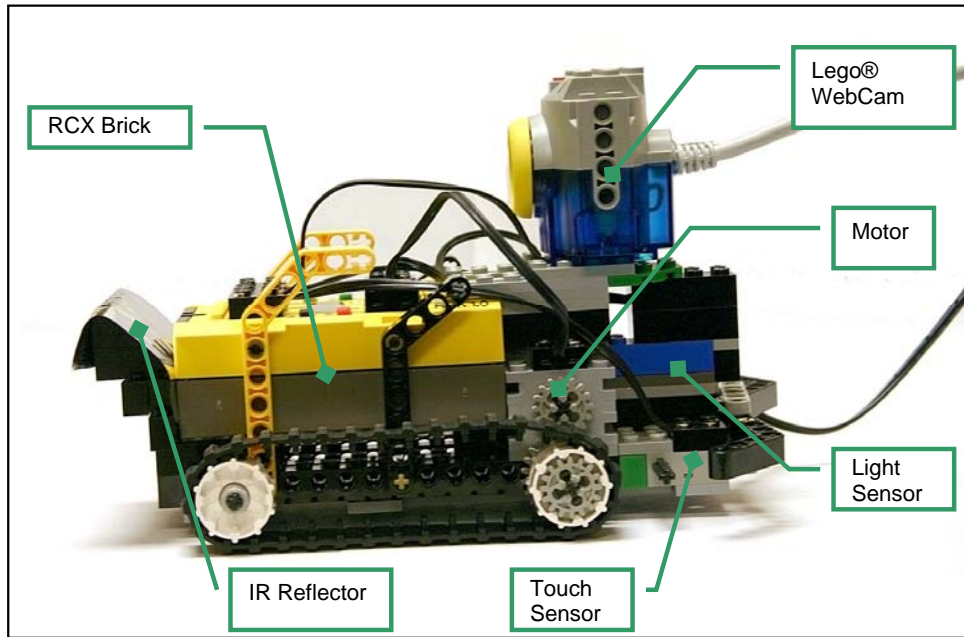
At the center of the robot is a large yellow brick called the RCX that contains many of the systems that the robot needs (Figure 2). Inside this brick is the robot's computer brain; it contains a simple microprocessor and small amount of computer memory. The processor and memory are very limited compared to a modern computer, but the amount of processing power needed by the robot is comparatively low. In addition to the microprocessor, the RCX brick also has connections for sensor inputs and motor outputs. These inputs and outputs allow the microprocessor to detect changes in the robot's environment and react by controlling its electric motors. The RCX also contains an infrared (IR) receiver and transmitter that uses IR light pulses to communicate with a special IR device attached to the Internet server computer. The Internet server downloads user programs onto the robot over this IR communication link. A reflective strip mounted in front of the RCX's IR port helps facilitate communication. Finally, the RCX brick contains the batteries needed to power the robot.



*Figure 2: WebBot Robot Top View*

Behind the RCX on the robot are two electric Lego<sup>®</sup> motors. The motors use gears to drive the rubber treads on either side of the robot. The RCX can make the robot move forward or backward or turn left or right by controlling the direction the motors turn. Behind the robot's motors is an array of sensors that the robot uses to automatically find its way back to the WebBot's "home" position in the maze.

There are three sensors in the array: touch sensors mounted on the left and right corners of the robot help the robot navigate backwards through the maze and a light sensor is used to detect when the robot has reached the home position (Figure 3). The last major component of the WebBot robot is a Lego<sup>®</sup> webcam mounted on the top of the robot. This camera shows users on the Internet what the robot sees in its environment.

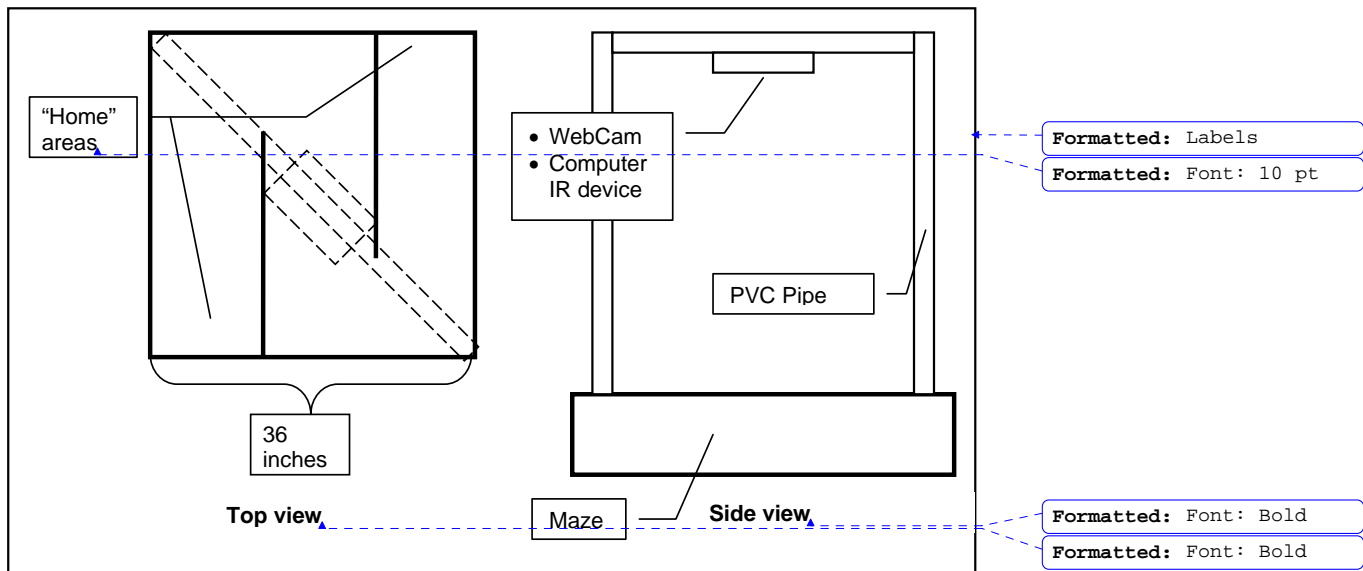


*Figure 3: WebBot Robot Side View*

#### *WebBot's Environment*

WebBot's environment was designed to fulfill several different needs. First, the environment is enclosed so users cannot unintentionally command the robot to move out of contact with the computer's IR transmitter. If this were to happen, the robot could not receive programs sent by the users and it would effectively crash the system, requiring human intervention to fix. WebBot has been designed to minimize human maintenance needs wherever possible and to insure that the site is available 24 hours a day, seven days a week. WebBot's environment also insures that the IR communication system between the robot and the computer works from anywhere in the maze. The IR communication system does not necessarily need direct line-of-sight but it does need surfaces to bounce signals off of. The robot and the computer's IR transmitter/receiver also need to be

relatively close to work. To make sure this system works as reliably as possible, the computer's IR transmitter is mounted several feet above the environment facing downwards. This prevents a situation where the robot could end up facing the opposite direction from the computer's IR transmitter and lose connection that would again require human intervention. A second webcam that provides a top-down view of the environment is mounted along with the computer's IR transmitter so that users of the web site can see where the robot is in the maze.



*Figure 4: WebBot's Environment*

The last feature that WebBot's environment provides is an interesting challenge for the robot to complete. For this phase of the project WebBot has a simple maze to navigate (Figure 4). The maze only has a few turns but it is enough to challenge users to think about how to write a sequence of commands to navigate to the end. The maze's walls are made of particle-board shelving, and all walls are white except for two walls

with black strips at the maze's "home" points. The black strips help the robot find its way home after a user attempts to navigate the maze. In order to do this, the robot feels its way backwards along the walls of the maze with its touch sensors until it backs up to the black strip and its light sensor detects the darker surface. When it detects the black strip, the robot knows it has reached the starting position, stops, and waits for further commands from the user.

### *WebBot Robot Software*

Much effort went into deciding which programming language to use to write the WebBot project's software. The 4-H robotics curriculum uses software from Lego<sup>®</sup> called Robolab<sup>™</sup> that is based on National Instruments LabView<sup>™</sup> software. While it would have been convenient to use Robolab<sup>™</sup> for this project, there were no examples available for communicating over the Internet in the way the project's goals demanded, and it was unclear if it could be made to work. Internet research showed that there are other programming environments for Lego<sup>®</sup> Mindstorms<sup>®</sup> that have been developed by Mindstorms<sup>®</sup> enthusiasts that were potential solutions to this problem. A further Internet search on the subject resulted in a programming language called LeJOS ([lejos.sourceforge.net](http://lejos.sourceforge.net)) that had examples demonstrating most of the software features needed for WebBot. A book written about LeJOS titled *Core LEGO<sup>®</sup> MINDSTORMS<sup>®</sup> Programming* (Bagnall, 2002) provided an example of software that could control a robot from a web page. Given the examples and features that LeJOS provided, the decision was made to use it for WebBot's programming language. LeJOS is essentially a version of the Java<sup>™</sup> programming language simplified enough to work on the RCX's microprocessor. Some features of Java<sup>™</sup> had to be eliminated because of the limited

resources of the RCX, but most of the syntax and fundamental language constructs remain the same.

The software that runs on WebBot's robot performs several tasks. It downloads command sequences from the user, executes those commands, and it also has the ability to navigate the robot back to the starting point of the maze. While there are not many tasks for the robot to perform, knowing which task the robot should be prioritizing at any given moment becomes complicated. LeJOS has a built-in system for prioritizing tasks that uses the concept of behaviors of living creatures. Put simply, behaviors are different computer programs that run based on two conditions. First, behaviors are triggered when certain conditions are met; these conditions could be sensor inputs or could be based on a timer or variable. Second, behaviors are ranked by priority and only one behavior can execute at a time. If two or more behavior programs' conditions are met, only the behavior with the highest ranked priority is executed. WebBot's robot uses this behavior system to arbitrate what the robot should be doing at any given time.

<b>Behavior</b>	<b>Conditions when behavior executes</b>
1. Return to home	When "return" is the current command OR after a timeout after last command is run
2. Move forward	When "forward" is the current command
3. Move backward	When "backward" is the current command
4. Turn right	When "turn right" is the current command
5. Turn left	When "turn left" is the current command
6. Get commands from user	Always, but give up control briefly if communication times out
7. Nothing	Always. (Dummy behavior to insure proper flow of behavior arbitration system.)

*Table 1: WebBot's behavior priorities*

*WebBot's Server Software*

WebBot requires a two-layer client-server system to relay commands from the user's web browser to the robot (Figure 5). This multiple-relay system is necessary because computers on the Internet cannot talk directly to the robot; software on the web server must listen for commands from the users' web browsers and then re-transmit those commands to the robot. The decision to use LeJOS on the robot dictated the decision to use Java™ to write the software that runs on the Internet server. Because LeJOS is based on Java™, it has several built-in communication features that work directly with full Java™ programs. Furthermore, Java™ has the capability to listen for incoming communication over the Internet. This server program waits for a TCP/IP socket connection from a user somewhere on the Internet. Then, once connected to a user client, it waits for a user program to be downloaded to it. Once the server software receives a user program, it opens an IR connection to the robot and downloads the program to the robot. The Internet server sends updates on the status of the downloading progress to the Internet client as it downloads the program to the robot.

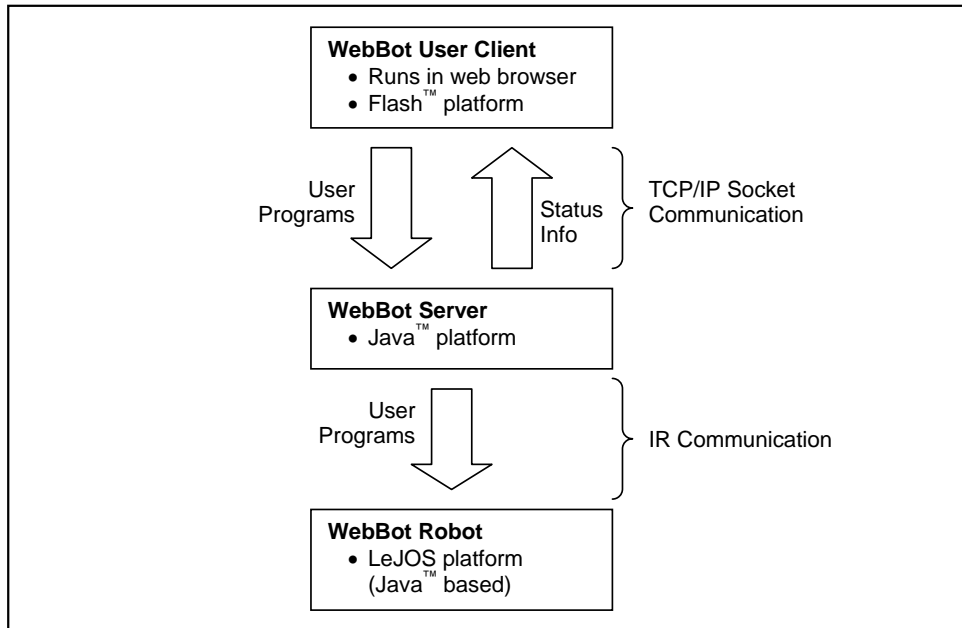
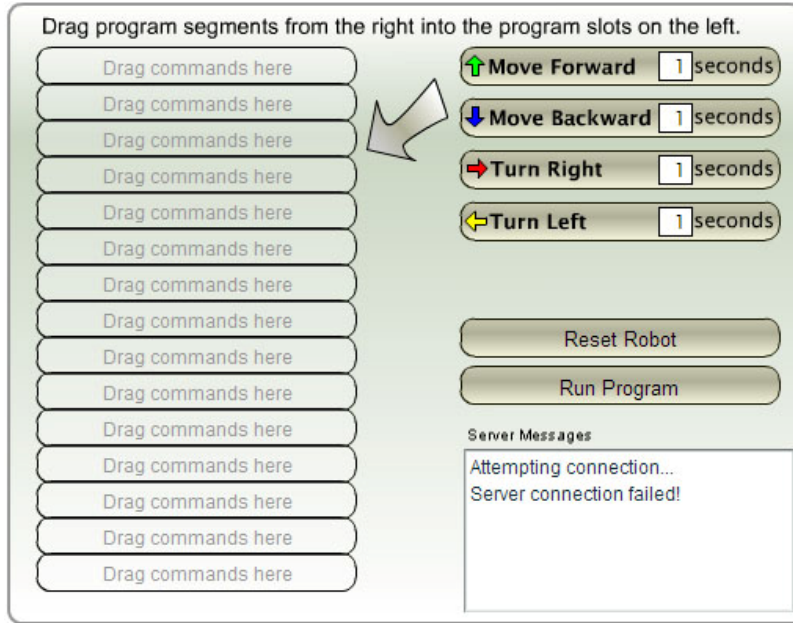


Figure 5: WebBot Software Communication Diagram

#### WebBot Web Client Software

The WebBot web client performs two tasks; it acts as a user interface for programming the robot and transmits the user programs to the server software. While the examples provided in *Core LEGO® MINDSTORMS® Programming* (Bagnall, 2002) used Java™ applets or Java™ Server Pages to communicate with the robot, it was decided to use Flash™ to create the user experience for a variety of reasons. The developer of WebBot was more familiar with Flash as a development environment on the web which allowed him to quickly prototype an interactive programming environment. A web search provided working code examples that showed how to establish TCP/IP socket communication between a Flash™ applet and a Java™ server program.

The user interface for programming the robot is fairly simple (Figure 6). A column with 15 rows on the left half of the screen starts out empty. On the right side of the screen there are four different command buttons representing the available commands for the robot. Users are instructed to use their mouse to click and drag commands from the right side of the screen to the column on the left. As they do so, a sequence of commands is created in the left column that represents the flow of the user program. Once the commands have been added to the left column, they can adjust the time parameter of the command by clicking in a text field and changing the number. By default, each movement command executes for one second but users must adjust these timings to successfully navigate the maze. The user can re-order or remove the commands in the left column by dragging and dropping them to a new position. If users want to delete a command, they can drag it outside of the program column and release the mouse button causing the command to disappear. When the user is satisfied with the program, he or she can click a button that will send their program to the server software and on to the robot. Users are shown a computer-controlled example of how to use the programming environment before they can use it themselves so that they will understand the fundamentals of the design.



*Figure 6: Programming Client Prototype*

In addition to providing the programming user interface, the web component of WebBot must also make sure only one user has access to the robot at a time. This is necessary because one of the limitations of using an actual, physical robot to teach is that it can only be controlled by one user at a time. If more than one user were allowed to send commands at the same time, each user would think the robot was acting erratically as it tried to execute several users' commands at the same time. In order to limit access to the robot a server script uses the Internet IP numbers of users' computers when they visit the WebBot site. When a user visits the site and the robot is available the system records their IP number and they are given sole access to the web page with the robot controls. A 30-minute timer begins counting down when the user accesses the page to prevent a user from monopolizing control of the robot. When it expires, they connection

to the user's computer is closed and that IP number is blocked from accessing the robot control page for one hour. While the robot is in use, other users with different IP numbers are shown a message asking them to return later.

The program code for WebBot is provided in appendices. The code files are listed and described in Table 2.

	File	Functionality
Appendix A	webRemoteServer.java	This program runs on the Internet Server. It relays user robot programs from the user client to the robot.
Appendix B	Flash™ client TCP/IP socket communication code	This code is part of the Flash™-based user client. It opens a TCP/IP socket connection with the Internet server
Appendix C	rcxWebRemote.java	This is the primary part of the robot software. It initializes the robot, starts the behavior system, and implements the communication system.
Appendix D	driveForward.java	This file is part of the robot software. It implements the forward movement behavior.
Appendix E	driveBackward.java	This file is part of the robot software. It implements the backward movement behavior.
Appendix F	turnLeft.java	This file is part of the robot software. It implements the left turn behavior.
Appendix G	turnRight.java	This file is part of the robot software. It implements the right turn behavior.
Appendix H	listenForCommands.java	This file is part of the robot software. It implements the behavior causes the robot to listen for user programs.
Appendix I	robotReset.java	This file is part of the robot software. It implements the behavior that lets the robot automatically navigate back to the home position in the maze.
Appendix J	nothing.java	This file is part of the robot software. It implements an empty behavior to insure the behavior system functions properly.

*Table 2: Program code for WebBot.*

#### IV. Future Directions

The version of WebBot described in this paper is very simple compared to the possibilities of the system. Most of the effort put towards WebBot so far has been directed at the fundamental systems that WebBot needs to function. There are many areas where WebBot can be expanded and improved if interest is high.

The next logical step for the WebBot project is formative evaluation of the lessons and the concept itself. WebBot has not yet been put into practical use and it remains to be seen if the concept has too many real-world limitations to be practical. While many technical problems have been overcome in creating WebBot, unforeseen problems may be discovered as more and more users access the site. Furthermore, it may be discovered that the lessons associated with the robot are ineffective or ignored. The programming environment and maze-navigation challenge may also be too challenging for many users. Careful formative evaluation could answer some of these questions and provide recommendations about how to fix or improve weak areas. It is critical that WebBot be entertaining and engaging or else it could actually diminish interest in the Nebraska 4-H robotics program.

In addition to formative evaluation and improvement, there are design improvements that could be made to the robot as well. For example, currently the webcam attached to the robot is tethered by a long data cable. Steps have been taken to minimize tangling but it is probably inevitable that it will eventually cause a problem. Wireless video cameras are available that would make the robot completely autonomous. The robot could also be improved by adding more sensors with different functionality. Custom third-party sensors are available over the Internet that can perform proximity

detection, compass functionality, and other parts expand the number of sensors the robot can use at one time. One practical use for the compass sensor would be to make turning much more precise. Currently the robot can only perform timed turns making turning an accurate number of degrees virtually impossible. Trial and error methods must be used to find the best time to make a proper 90 degree turn. With a compass sensor it would be easy to simply turn the robot until the sensor showed a 90 degree turn.

Another exciting area for improvement of this project would be to re-implement it using Lego<sup>®</sup>'s new NXT robotics kit. This is a brand-new system of robotics based around a significantly more powerful processor, more memory, and more accurate motors and sensors. Furthermore, NXT robots can communicate using wireless Bluetooth signals, eliminating many of the IR communication range issues with the current robot. It may take quite some time, however, before the Mindstorms<sup>®</sup> enthusiast community supports the NXT platform as well as the current RCX platform. It would not have been possible to create WebBot without these community-developed tools.

Another area for future exploration would be to create new challenges for users to attempt with the robot. WebBot's environment could be re-configured for a variety of different experiences. For example, the programming environment could be changed to allow programmable loops or give users the ability to read sensor values. The maze activity would be a completely different challenge using sensors instead of timing to navigate the maze. It would also be possible to remove the maze and design an activity that challenges users to push balls into a designated goal area. Another activity could be to make the robot measure light-sensor readings on various objects in the environment and report them back to the user in a Mars rover-type activity. It would also be possible

to create a second web-controlled robot allowing two users to compete and try to score goals on each other. The experiences could be used to demonstrate the wide variety of ways that robots are used in real life.

A final area for future exploration could be to conduct research on the effectiveness of teaching STEM with robots. It would be relatively easy to add on-line testing components to the web site that could investigate a variety of research questions. For example, a simple pre-test/post-test arrangement could show if the users learned from the lessons on the site. A testing instrument could also be devised to investigate whether the lessons they learn with robots transfer into broader STEM knowledge. As noted above, the maze activity could be replaced with another activity depending on the needs of different research questions.

## **V. Summary**

Combining robotics with education provides exciting opportunities for progress and innovation. Research shows that robots have unique characteristics that help learners understand difficult STEM concepts while engaging and motivating students at the same time. The WebBot project attempts to build on these strengths and add some unique strengths of its own. If WebBot can successfully connect the power of robotics with the reach of the Internet, many children who would have missed the opportunity to experiment with robotics could be reached. Even if only a few of these children are encouraged to choose to pursue STEM careers, WebBot could be considered a success. Still, WebBot has only just begun its robotic life and is untested. It still faces challenges and obstacles that have not been foreseen. Research should be done to determine if robotics, education, and the Internet are ready to be combined.

### References

- Bagnall, B. (2002). *Core LEGO® MINDSTORMS® Programming: Unleash the Power of the Java Platform*. New York: Prentice Hall PTR.
- Barnes, D. J. (2002, February-March). *Teaching introductory Java through Lego Mindstorms models*. Paper presented at SIGCSE 2002 Technical Symposium on Computer Science Education, Covington, Kentucky.
- Beer, R. D., Chiel, H. J., & Drushel, R. F. (1999). Using robotics to teach science and engineering. *Communications of the ACM*, 42(6), 85-92.
- Bonvillian, W. B. (2002). Science at a crossroads. *FASEB Journal*, 16, 915–921.
- Fagin, B., & Merkle, L. (2003, February). *Measuring the effectiveness of robots in teaching computer science*. Paper presented at SIGCSE 2003 Technical Symposium on Computer Science Education, Reno, Nevada.
- Geissler, J., Knott, P.J., Vazquez, M.R., & Wright, J.R. (2004). Virtual Reality Robotic Programming Software in the Technology Classroom. *The Technology Teacher*, 63(6), 6-8.
- Gonzales, P., Guzmán, J. C., Partelow, L., Pahlke, E., Jocelyn, L., Kastberg, D., & Williams, T. (2004). *Highlights from the Trends in International Mathematics and Science Study (TIMSS) 2003*. Washington, DC: U.S. Department of Education, National Center for Education Statistics.
- Lemke, M., Sen, A., Pahlke, E., Partelow, L., Miller, D., Williams, T., Kastberg, D., & Jocelyn, L. (2004). *International outcomes of learning in mathematics literacy and problem solving: PISA 2003 results from the U.S. perspective*. Washington, DC: U.S. Department of Education, National Center for Education Statistics.

- Mauch, E. (2001). Using technological innovation to improve the problem solving skills of middle school students. *The Clearing House*, 75(4), 211-213.
- Moore, V. S. (1999). Robotics: Design through geometry. *The Technology Teacher*, 59(3), 17-22.
- Nourbakhsh, I., Crowley, K., Bhave, A., Hamner, E., Hsium, T., Perez-Bergquist, A., Richards, S., & Wilkinson, K. (2005). The robotic autonomy mobile robots course: Robot design, curriculum design, and educational assessment. *Autonomous Robots*, 18(1), 103-127.
- Papert, S. (1980). *Mindstorms: children, computers, and powerful ideas*. New York: Basic Books, Inc.
- Porter, M., & van Opstal, D. (2001) *U.S. Competitiveness 2001: Strengths vulnerabilities and long-term priorities*. Washington, DC: Council on Competitiveness.
- Robinson, M. (2005). Robotics-driven activities: Can they improve middle school science learning? *Bulletin of Science, Technology & Society*, 25(1), 73-84.
- Rogers, C., & Portsmore, M. (2004). Bringing engineering to elementary school. *Journal of STEM Education*, 5(3&4), 17-28.
- Waddel, S. (1999). Why teach robotics? *Tech Directions*, 58(7), 34-35.
- Wagner, S.P. (1999). Robotics and Children: Science Achievement and Problem Solving. *Information Technology in Childhood Education Annual*, 101-145.

## Appendix A

### WebBot Server Source Code: webRemoteServer.java

Code written by John Ansoorge based on example code by Brian Bagnall (2002).

```
import java.io.*;
import java.net.Socket;
import java.net.*;

import josx.platform.rcx.Serial;
import josx.rcxcomm.*;

public class webRemoteServer {
    //Declare class variables
    ServerSocket server;
    Socket client;
    BufferedReader clientInStream;
    PrintWriter clientOutStream;
    RCXPort port;
    DataInputStream inRCX;
    DataOutputStream outRCX;
    char EOF = (char)0x00;

    public webRemoteServer(int portNumber) {
        try {
            // Internet connection:
            server = new ServerSocket(portNumber);

            // RCX Connection:
            port = new RCXPort();
            inRCX = new DataInputStream(port.getInputStream());
            outRCX = new DataOutputStream(port.getOutputStream());
        }
        catch (IOException io){
            io.printStackTrace();
            System.exit(0);
        }
    }

    public static void main(String [] args) {

        //Create an instance of this class that listens to port 5067
        webRemoteServer host = new webRemoteServer(5067);

        //Endless loop.
        while(true) {

            //Listen for connections from the Internet
            host.waitForConnection();

            //When a connection is made, start waiting for commands.
            while(host.client!=null) {
```

```

        //Diagnostics that put out to the console.
        System.out.println("Going into waiting for commands
mode.");
        host.waitForCommands();
    }
}

public void waitForConnection() {
    try {
        System.out.println("Listening for client.");

        /*Accept connections from Internet clients. The program
will
        * stop here until a connection is made.
        * */
        client = server.accept();

        //Create input/output streams for connection.
        clientInStream = new BufferedReader(new
InputStreamReader(client.getInputStream()));
        clientOutputStream = new
PrintWriter(client.getOutputStream());
        System.out.println("Client connected.");

        //Send reply to connected client to verify connection
status.
        clientOutputStream.print("connect" + EOF);
        clientOutputStream.flush();
    } catch(IOException io) {
        io.printStackTrace();
    }
}

public void waitForCommands() {
    System.out.println("Waiting for commands from client.");
    //Initialize method variables
    short programLength;
    short [] commandArray = null;
    short [] argArray= null;
    String inputString;
    String [] inputStringArray;

    try {
        System.out.println("Trying to read values from
programming client.");

        /*
        * Read a line from the Internet client. The program
will stop here until the line has been read.
        */
        inputString = clientInStream.readLine();
        System.out.println("Read Values:_" + inputString +
        "_");
    }
}

```

```

        //Reply to Internet client that data was received
successfully.
        clientOutputStream.print("received" + EOF);
        clientOutputStream.flush();
    } catch(IOException io) {

        /*If the client disconnects, reset variables, drop
out of waitForCommands(),
        * and go back to waitForConnection().*/
        System.out.println("Client disconnected.");
        client = null;
        return;
    }

    //Process string
    /*Sometimes when the client disconnects this server gets
badly formatted data.
    * We'll check for that and disconnect if so.*/
    if (inputString==null){
        System.out.println("Received NULL, disconnecting.");
        client = null;
        return;
    }
    if (inputString.trim()==""){
        System.out.println("Received empty string,
disconnecting.");
        client = null;
        return;
    }
    try {

        /*Split the data read from the client into an array
of commands.
        * (Commas are used for delimiters in the input
data.)
        * */
        inputStringArray = inputString.trim().split(",");
        System.out.println(inputStringArray.length);
        //If array is too short, disconnect.
        if (inputStringArray.length < 2) {
            client = null;
            return;
        }

        /*Move the input data from the input array into an
array just for commands
        * and an array just for arguments.*/
        commandArray = new short [inputStringArray.length/2];
        argArray = new short [inputStringArray.length/2];
        for (int i=0; i < inputStringArray.length/2; i++) {
            commandArray[i] =
Short.parseShort(inputStringArray[i*2]);
            argArray[i] =
Short.parseShort(inputStringArray[i*2+1]);

```

```

    }
    } catch (Exception e) {
        return;
    }

    /*Now that input from the client has been validated and
formatted into arrays,
    * we will send it down to the robot.
    * */
    try {
        System.out.println("Starting");

        //First, we tell the robot how many commands it should be
reading in
        outRCX.writeShort(commandArray.length);
        System.out.println("Wrote Array Length");

        //Now we send the commands and the arguments.
        for (int i=0; i < commandArray.length;i++) {
            outRCX.writeShort(commandArray[i]);
            System.out.println("Wrote cmd " + i + " " +
commandArray[i]);
            outRCX.writeShort(argArray[i]);
            System.out.println("Wrote arg " + i + " " + argArray[i]);
        }

        /*Here, we tell the Internet client that the download to
the robot has completed
        * successfully and the robot will run the program now.
        * */
        clientOutputStream.print("starting" + EOF);
        clientOutputStream.flush();
    }
    catch (Exception e) {
        System.out.println("Exception " + e.getMessage());
    }
}
}

```

## Appendix B

### Flash client TCP/IP socket communication code

Code written by John Ansoorge based on example code by Marco Lapi

([http://www.gotoandplay.it/\\_articles/2003/12/xmlSocket.php](http://www.gotoandplay.it/_articles/2003/12/xmlSocket.php)).

```
import net.guya.Timeout;
mySocket = new XMLSocket();
serverConnected = false;
instanceName.onUnload = function () {
    mySocket.close();
}
msgText.text = "Attempting connection...\n";

mySocket.onConnect = function(success)
{
    if (success) {
        Timeout.set(checkConnect, 1000);
    } else {
        msgText.text += "Server connection failed!\n"
    }
}

mySocket.onClose = function()
{
    msgText.text += "Server connection lost.\n"
}

XMLSocket.prototype.onData = function(msg)
{
    //trace("MSG: " + msg)
    //msgText.text += "_" + msg + "_\n";
    switch (msg) {
        case "connect":
            serverConnected = true;
            //msgText.text += serverConnected + "\n";
            break;
        case "received":
            msgText.text += "Program sent to the server.
Downloading to the robot.\n"
            break;
        case "starting":
            msgText.text += "Starting robot.";
            break;
    }
    msgText.vPosition = msgText.maxVPosition;
}
if (!mySocket.connect(null, 5067)) {
    msgText = "Connection failed!\n";
}
```

```
function sendMsg(message) {
  if (serverConnected) {
    if (message != "") {
      mySocket.send(message + "\n");
    }
  } else {
    msgText.text += "Cannot send, server not connected.\n";
  }
}

function checkConnect(){
  if (serverConnected) {
    msgText.text += "Server connection established!\n";
    Timeout.set(closeConnection, 3600000);
  } else {
    msgText.text += "Could not complete server
connection...closing.";
    mySocket.close();
  }
}

function closeConnection() {
  mySocket.close();
  msgText.text += "Your time with the robot has expired.
Disconnecting so someone else can use the robot.";
}
```

## Appendix C

### WebBot Robot Software: rcxWebRemote.java

Code written by John Ansoorge based on example code by Brian Bagnall (2002).

```
import java.io.*;
import josx.rcxcomm.*;
import josx.platform.rcx.*;
import josx.robotics.*;
import josx.util.*;

public class rcxWebRemote implements SensorConstants, TimerListener {
    //Variables for communications and command queue
    RCXPort port = null;
    DataInputStream in;
    short arraySize;
    short [] commands = new short [20];
    short [] arg = new short [20];
    short commandQueueCounter = 0;
    boolean listenForCommandsFlag = true;

    //Behavior class instancing. This is necessary because we need
    to pass a reference to this class and we're using a superset of the
    Behavior classes abilities.
    robotReset myRobotReset = new robotReset(this);
    driveForward myDriveForward = new driveForward(this);
    driveBackward myDriveBackward = new driveBackward(this);
    turnRight myTurnRight = new turnRight(this);
    turnLeft myTurnLeft = new turnLeft(this);
    updateCommandQueue myUpdateCommandQueue = new
updateCommandQueue(this);
    listenForCommands myListenForCommands = new
listenForCommands(this);

    //This timer will trigger the robotReset behavior after the user
    commands are run and 25 seconds have passed.
    Timer timer = new Timer(25000,this);

    //Constructor: This method initializes everything that hasn't
    already been intialized above.
    public rcxWebRemote() {
        //Set up robot stuff
        LLC.setRangeLong();

        Sensor.S1.setTypeAndMode(SENSOR_TYPE_TOUCH,SENSOR_MODE_BOOL);
        Sensor.S1.addSensorListener(myRobotReset);

        Sensor.S3.setTypeAndMode(SENSOR_TYPE_TOUCH,SENSOR_MODE_BOOL);
        Sensor.S3.addSensorListener(myRobotReset);

        Sensor.S2.setTypeAndMode(SENSOR_TYPE_LIGHT,SENSOR_MODE_PCT);
        Sensor.S2.activate();
    }
}
```

```

Motor.A.setPower(7);
Motor.C.setPower(7);

//initialize command queue arrays
for (int j=0; j < 20;j++) {
    commands[j]=0;
    arg[j]=0;
}
LCD.showNumber(commands[commandQueueCounter]);

//Set up communication stuff
try {
    port = new RCXPort();
    port.setTimeout(1000);
    in = new DataInputStream(port.getInputStream());
    //DataOutputStream out = new
DataOutputStream(port.getOutputStream());
} catch (IOException ioE) {
    LCD.showNumber(1112);
}

//Set up behavior stuff
Behavior b1 = new nothing();
Behavior b2 = myRobotReset;
Behavior b3 = myDriveForward;
Behavior b4 = myDriveBackward;
Behavior b5 = myTurnRight;
Behavior b6 = myTurnLeft;
Behavior b7 = myListenForCommands;
Behavior [] bArray = {b1,b7,b6,b5,b4,b3,b2};
Arbitrator arby = new Arbitrator(bArray);
arby.start();
}

public static void main(String[] args) {
    //Create instance of rcxWebRemote and let the fun begin!
    new rcxWebRemote();
}

//This method handles the timer when it times out.
public void timedOut() {
    timer.stop();
    myRobotReset.timerHandler();
}

//This method returns the current command in the queue in the
form of a coded number.
public short getCommandQueueCommand() {
    if(commandQueueCounter > 19) {
        return 0;
    } else {
        return commands[commandQueueCounter];
    }
}
}

```

//This method returns the argument for the current command in the command queue. Currently these arguments are used to tell the robot how long to perform the movement behaviors.

```
public short getCommandQueueArg() {
    if(commandQueueCounter > 19) {
        return 0;
    } else {
        return arg[commandQueueCounter];
    }
}
```

/\*This method determines if the listenforCommands behavior can take control.

If the listenForCommands communication code below times out, we need to make sure its behavior gives up control.

It performs this check by making sure that all commands in the commands queue are complete and that the behavior shouldn't be gracefully giving up control.

```
*/
public boolean listenForCommandsCheck() {
    if (getCommandQueueCommand()!=0 | listenForCommandsFlag ==
false) {
        listenForCommandsFlag=true;
        LCD.showNumber(1113);
        return false;
    } else {
        return true;
    }
}
```

//This is the function that updates the command queue counter

```
public void updateCommandQueue() {
    commandQueueCounter++;
    timer.start();
}
```

//This function does the actual work of reading the commands from the server computer and putting them in the command queue arrays.

```
public void listenForCommands() {
    try {
        //re-initialize command queue
        for (int j=0; j < 20;j++) {
            commands[j]=0;
            arg[j]=0;
        }
        commandQueueCounter = 0;
        //Read Array Size
        arraySize = in.readShort();
        LCD.showNumber(arraySize);
        //Read Array values arraySize number of times
        for (int i=0; i<arraySize;++i) {
            commands[i]=in.readShort();
            LCD.showNumber(commands[i]);
            arg[i]=in.readShort();
            LCD.showNumber(arg[i]);
        }
    }
}
```

```
    }  
  } catch (IOException ioE) {  
    //This exception will usually mean that a  
communication attempt has timed out  
    listenForCommandsFlag=false;  
    LCD.showNumber(1111);  
  }  
}  
}
```

## Appendix D

### WebBot Robot Software: driveForward.java

Code written by John Ansorge.

```

import josx.platform.rcx.*;
import josx.robotics.*;

public class driveForward implements Behavior {
    rcxWebRemote initiator;

    public driveForward(rcxWebRemote r) {
        initiator = r;
    }

    public void action() {
        //Diagnostic number
        LCD.showNumber(8);
        //Start moving forward
        Motor.A.forward();
        Motor.C.forward();
        //Wait for the amount of time set by the user.
        try{Thread.sleep(initiator.getCommandQueueArg());}catch(Exception
e){}
        //Stop
        Motor.A.stop();
        Motor.C.stop();
        //Set the pointer for the command queue to the next
command.
        initiator.updateCommandQueue();
    }

    public boolean takeControl() {
        //Take control of the robot if the current command in the
queue is drive forward (encoded as "8").
        if(initiator.getCommandQueueCommand()==8) {
            return true;
        } else {
            return false;
        }
    }

    public void suppress() {
        //This will probably never be called.
        Motor.A.stop();
        Motor.C.stop();
    }
}

```

## Appendix E

WebBot Robot Software: driveBackward.java

Code written by John Ansorge.

```
import josx.platform.rcx.*;
import josx.robotics.*;

public class driveBackward implements Behavior {
    rcxWebRemote initiator;

    public driveBackward(rcxWebRemote r) {
        initiator = r;
    }

    public void action() {
        LCD.showNumber(2);
        Motor.A.backward();
        Motor.C.backward();

        try{Thread.sleep(initiator.getCommandQueueArg());}catch(Exception
e){}

        Motor.A.stop();
        Motor.C.stop();
        initiator.updateCommandQueue();
    }

    public boolean takeControl() {
        if(initiator.getCommandQueueCommand()==2) {
            return true;
        } else {
            return false;
        }
    }

    public void suppress() {
        Motor.A.stop();
        Motor.C.stop();
    }
}
```

**Appendix F**

WebBot Robot Software: turnLeft.java

Code written by John Ansorge.

```
import josx.platform.rcx.*;
import josx.robotics.*;

public class turnLeft implements Behavior {
    rcxWebRemote initiator;

    public turnLeft (rcxWebRemote r) {
        initiator = r;
    }

    public void action() {
        LCD.showNumber(4);
        Motor.A.backward();
        Motor.C.forward();

        try{Thread.sleep(initiator.getCommandQueueArg());}catch(Exception
e){}

        Motor.A.stop();
        Motor.C.stop();
        initiator.updateCommandQueue();
    }

    public boolean takeControl() {
        if(initiator.getCommandQueueCommand()==4) {
            return true;
        } else {
            return false;
        }
    }

    public void suppress() {
        Motor.A.stop();
        Motor.C.stop();
    }
}
```

## Appendix G

### WebBot Robot Software: turnRight.java

Code written by John Ansorge.

```
import josx.platform.rcx.*;
import josx.robotics.*;

public class turnRight implements Behavior {
    rcxWebRemote initiator;

    public turnRight (rcxWebRemote r) {
        initiator = r;
    }

    public void action() {
        LCD.showNumber(6);
        Motor.A.forward();
        Motor.C.backward();

        try{Thread.sleep(initiator.getCommandQueueArg());}catch(Exception
e){}

        Motor.A.stop();
        Motor.C.stop();
        initiator.updateCommandQueue();
    }

    public boolean takeControl() {
        if(initiator.getCommandQueueCommand()==6) {
            return true;
        } else {
            return false;
        }
    }

    public void suppress() {
        Motor.A.stop();
        Motor.C.stop();
    }
}
```

## Appendix H

WebBot Robot Software: listenForCommands.java

Code written by John Ansorge.

```
import josx.platform.rcx.*;
import josx.robotics.*;

/*This class is mainly here to be part of the behavior arbitration
system. Mostly it calls methods in
 * rcxWebRemote to do the heavy lifting.
*/
public class listenForCommands implements Behavior {
    rcxWebRemote initiator;

    public listenForCommands(rcxWebRemote r) {
        initiator = r;
    }

    public void action() {
        LCD.showNumber(11);
        initiator.listenForCommands();
    }

    public boolean takeControl() {
        //See listenForCommandsCheck method in rcxWebRemote for an
explanation of how this check is performed
        return initiator.listenForCommandsCheck();
    }

    public void suppress() {
        //Nothing to suppress.
    }
}
```

## Appendix I

### WebBot Robot Software: robotReset.java

Code written by John Ansorge.

```

import josx.platform.rcx.*;
import josx.robotics.*;

public class robotReset implements Behavior, SensorConstants,
SensorListener {
    private boolean notHome=false;
    private boolean inactiveTimer=false;
    rcxWebRemote initiator;

    public robotReset(rcxWebRemote r) {
        initiator = r;
    }

    //This method gets called when the master timer in rcxWebRemote
times out. It sets the inactiveTimer flag so this behavior knows it
should take over.
    public void timerHandler() {
        inactiveTimer=true;
    }

    public void action() {
        //Diagnostic number
        LCD.showNumber(10);
        /*Tell the robot it isn't home and start it on its way. It
is very important to realize that the sensors are on the BACK of the
robot, therefore
        * this behavior BACKS the robot up. In other words, for
the purposes of this behavior, backwards is forwards.*/
        notHome = true;
        Motor.A.backward();
        Motor.C.backward();
        //Loop while the robot navigates back home. When this loop
completes, the robot is home.
        while (notHome) {
            //These diagnostics show the values the light sensor
is reading while the robot navigates the maze.
            LCD.showNumber(Sensor.S2.readValue());
            LCD.refresh();
        }
        Motor.A.stop();
        Motor.C.stop();
        //If this behavior was triggered by a manual user command
(and NOT by the timer) update the command queue pointer.
        if (!inactiveTimer) {
            initiator.updateCommandQueue();
        }
        //Reset the timer flag.

```

```

        inactiveTimer=false;
    }

    public boolean takeControl() {
        //Check the timer AND command queue to see if this behavior
should take over.
        if(inactiveTimer | initiator.getCommandQueueCommand()==10)
    {
            return true;
        } else {
            return false;
        }
    }

    public void suppress() {
        Motor.A.stop();
        Motor.C.stop();
    }

    /*This method listens to input from the sensors. When one of the
touch sensors is pushed, this code is called. Most of the maze
navigation logic is here. */
    public void stateChanged(Sensor src, int oldValue, int newValue)
    {
        //This method will watch the sensors whether this behavior
has control or not so we need to make sure it only controls the robot
when it should.
        if (notHome) {
            /*When the robot bumps into something it takes a
light reading. If it is dark enough, it indicates the robot
* has found the black strip on the wall that marks
its home. When it finds its home, it should stop moving and the
robotReset behavior should
* give up control of the robot.*/
            if (Sensor.S2.readValue() < 35) {
                notHome=false;
                return;
            }
            //If the robot isn't home, the rest of this code
executes. First we back up for 2/10ths of a second. (Remember,
forwards is backwards here.)
            Motor.A.forward();
            Motor.C.forward();
            try{Thread.sleep(200);}catch(Exception e){}
            //Next we turn away from the sensor that was bumped.
"src.getID" will tell us which sensor was triggered.
            if(src.getId()==0) {
                Motor.C.forward();
                Motor.A.backward();
                try{Thread.sleep(200);}catch(Exception e){}
            } else {
                Motor.A.forward();
                Motor.C.backward();
                try{Thread.sleep(200);}catch(Exception e){}
            }
        }
    }

```

```
        //Continue moving forward after the turn.  
        Motor.A.backward();  
        Motor.C.backward();  
    }  
}  
}
```

## Appendix J

### WebBot Robot Software: nothing.java

Code written by John Ansorge.

```
import josx.platform.rcx.LCD;
import josx.robotics.*;

//This class exists in the behavior arbitration system to make sure
that we don't get "stuck" executing one of the other behaviors
(primarily the listenForCommands behavior).
public class nothing implements Behavior {

    public void action() {
        LCD.showNumber(0);
    }

    public boolean takeControl() {
        return true;
    }

    public void suppress() {

    }
}
```