NOTE TO USERS

This reproduction is the best copy available.

An Integrated Architecture for a Networked Robotics Laboratory Using an Asynchronous Distance Learning Network Tool

by

William C. Harris

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

2008

This manuscript has been read and accepted for the
Graduate Faculty in Computer Science in satisfaction of the
dissertation requirement for the degree of Doctor of Philosophy.

April 18, 2008	David Arnow
_____	_____
Date	Chair of Examining Committee
April 24, 2008	Theodore Brown
_____	_____
Date	Executive Officer

Michael Anshel
R. Lynn Bondurant
Scott Dexter
Danny Kopec

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

**AN INTEGRATED ARCHITECTURE
FOR A NETWORKED ROBOTICS LABORATORY
USING AN ASYNCHRONOUS DISTANCE LEARNING NETWORK TOOL**

by

William C. Harris

Adviser: David Arnow

This research is concerned with the design, development, and implementation of a "Networked Robotics Laboratory" that will allow anyone on the Internet to learn a robot programming language, and use that language to control a robot from a distance. The project combines Inter-Networking, Multimedia, and Distance Learning Conceptualizations into a comprehensive system that broadens the availability and effectiveness of robot programming instruction.

There were two major activities associated with this project. The first activity involved technical development — building the networked robotics laboratory, and linking the asynchronous distance learning network tool (network learning tool) with the lab. The second activity was content development — creating specific robotics programming exercises. The technical development required extending the existing capabilities of the network learning tool, designing and implementing a video manager and robot manager programs, and developing the client/server communications between

these components. Java network programming was used to develop client/server applications that define the client/server communications. The content development required the creation of exercises suitable for both students and the network learning tool.

This dissertation consists of a comprehensive application that yields a problem requiring the successful resolution of a number of networking, multimedia and real-time issues in bridging the network learning tool to the Robotics Laboratory. **My thesis is that with the appropriate client/server applications, which will be developed as part of this research, network learning and multimedia tools can be coalesced with a networked robotics laboratory and lead to the development of a framework for bringing distance-learning systems to other laboratory contexts.**

Typical robotics labs are limited by the requirement of physical access, the need for the presence of instructional and supervisory personnel, and delays resulting from the learning curve of visiting students. An important outcome of this work has been the removal of the first two requirements (physical access and the presence of personnel) and a significant reduction of the learning curve. The efficacy of learning a robot programming language via the Web was also confirmed in experiments across several academic group levels. Most importantly, the networked robotics lab has proven to be a powerful way to motivate learning.

ACKNOWLEDGEMENTS

I wish to thank the faculty of The City University of New York's Doctoral Program in Computer Science, my students, friends, family members, and The Ancestors for their support and assistance with helping me complete this dissertation. Several individuals, however, deserve special mention for their contributions to this dissertation.

First, I want thank my advisor, colleague, and friend Dr. David Arnow (Brooklyn College) for his years of patience, support, and guidance. When I needed a space to set up my Networked Robotics Lab, and a quiet refuge to do the research with minimum interruptions, he offered me his faculty office. Dr. Arnow also assisted with putting together my excellent dissertation supervisory committee: Drs. Michael Anshel (The City College), Daniel Kopec (Brooklyn College), Scott Dexter (Brooklyn College), and R. Lynn Bondurant (NASA Glenn Research Center, Retired). Together, they offered insightful comments, suggestions, and recommendations that strengthened the dissertation, and helped me clarify basic ideas and articulate others.

Completing a Ph.D., teaching full time and coordinating the computer science program, and directing the NASA Aerospace Education Laboratory (AEL) has been difficult. For years I was torn between my doctoral studies, AEL responsibilities, and my computer science students at Medgar Evers College. Fortunately, I always had the support of the chairpersons in the Department of Physical, Environmental, and Computer Sciences: Drs. Seymour Ien, Hiroko Karan, John Flowers, Leon Johnson, John Gibbs, and Shermane Austin. I also had the support of my NASA AEL Leadership Team at MEC: Fulvia Jordan, Assistant AEL Director and Crystal Cumberbatch-Greene, Family Café Coordinator.

I gratefully acknowledge the institutional support that I have received while working on this project. In particular, I thank the National Aeronautics and Space Administration; Science, Engineering, Mathematics, Aerospace Academy (SEMAA); Turing's Craft, Inc.; Lego Education; Education Technology Think Tank; and Medgar Evers College for supporting me with generous grants.

My friends and colleagues Donald Davis, Michael Hatchette, Michelle Jones, Ronnie Lowenstein, Charles Perry, Winston Roche, Beverly Tarver, Gregory Vaughn, William G. Wright, and Alfredo Yates have each played a significant role in helping me to complete this dissertation. They, and so many others, would often call me Dr. Harris for years before I finished. An obvious ploy to get me to finish my degree; well, it worked! To all of the wonderful people who never stopped asking, "Did you finish yet?", thank you!

Mr. & Mrs. Timothy and Janet Richardson (my parents-in-law) deserve special thanks as they have opened their Florida home to me for many summers of quite reflection and offered me moral support. Last summer, my mother-in-law also provided a gentle nudge by showing me an article in the St. Petersburg Times, "Student makes UF History: She graduates with 2 Ph.D.s." That really got my attention! Thanks, mother-in-law.

I especially wish to thank my loving wife Rhonda, and our two beautiful daughters, Rashida and Khadija who believed in me throughout this journey. Rhonda's love and support is truly amazing. I would also like to thank my parents William and Tempia without whom, none of this would have happened. Finally, I thank The Ancestors, those significant ones and those who have been obscured in history. Your valiant struggle for life made me what I am today. Thank you for transcending time and space, and transforming me.

TABLE OF CONTENTS

TABLE OF FIGURES

1. AN INTERGATED ARCHITECTURE FOR A NETWORKED ROBOTICS LABATORY USING AN ASYNCHRONOUS DISTANCE LEARNING TOOL

1.1 Introduction

This research involves the design, development, and implementation of a "Networked Robotics Laboratory" by combining Inter-Networking, Multimedia, and Distance Learning into a comprehensive system that broadens the availability and effectiveness of robot programming instruction. The networked robotics lab allows anyone on the Internet to access the lab, learn a robot programming language, use that language to program the lab's robot, and view the robot's programmed behavior. Thus, the networked robotics lab allows anyone with Internet access to control a robot.

This research examines how computer-based and web-based asynchronous distance learning tools can be efficiently integrated into a networked robotics laboratory while using conventional-bandwidth communication links. This work also involves the design and development of network client/server applications for the seamless integration of several disparate systems with an eye towards the development of a framework for bringing distance-learning systems to other laboratory contexts.

Chapter 1 defines artificial intelligence (AI) and robotics, and their relationship to STEM. It also gives motivations and outlines the objectives of this research activity. Chapter 2 states the problem, talks about the challenge of finding a way to address the shortage of engineers and computer scientists, and discusses the need for innovative pedagogical tools for robot programming. A review of related work is presented in Chapter 3. Chapter 4 presents the robot and the other architectural components of the networked robotics laboratory in stand-alone form. The components described in

Chapter 4 are integrated into a cohesive whole, and implemented into a working system in Chapter 5, where network client/server applications and asynchronous distance learning are discussed. Chapter 6 presents the results of several robotics laboratory learning experiments. Chapter 7 describes potential contributions of this work to knowledge in the field. The Appendices define terminology for robots and robotics, illustrates the anatomy of a robot, discusses the engineering approach to robotics, describes the Aerospace Educational Laboratory, gives a listing of the Java network (socket) programs, and provides sample pre/post tests.

1.2 Artificial Intelligence, Robotics, and STEM

Artificial Intelligence (*AI*) is the area of computer science that deals with the study and construction of artifacts that perceive, reason, and act to maximize their performance. An *agent* is an entity that perceives, reasons, and acts in an environment (Russell and Norvig 1995). An agent perceives through its input sensors, reasons by sorting through its perceived input to select a possible action, and then acts upon its environment through its output effectors. Thus, AI is concerned with intelligent agents. An example of a thermostat agent is shown on Fig 1.1.

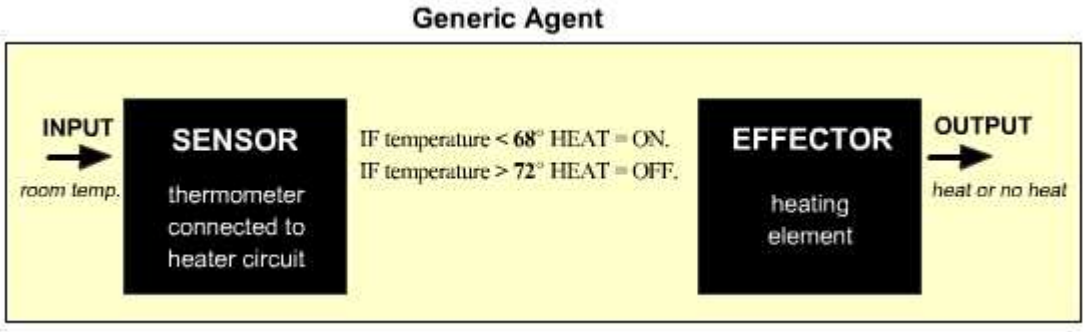

Fig 1.1 Thermostat agent

The thermostat agent receives input from a sensor, which is embedded in the environment, to detect the temperature. This agent responds to a very specific feature of the environment with only three possible actions: turn heat on or turn heat off or take no action. Notice that the environment determines the action of the agent and the agent's action, in turn, modifies the environment, in a relationship of mutual determination.

In simplest terms, a *robot* is a computer whose primary purpose is to produce motion (Fuller 1999). An *industrial robot* is officially defined by International Organization for Standardization (ISO) as an *automatically controlled, reprogrammable, multipurpose manipulator programmable in three or more axes*. While industrial robots have a manipulator that is designed to move material parts through various programmed motions, its base is stationary and the robot cannot relocate itself. *Autonomous mobile robots*, or *autonomous robotic rovers*, however, are programmable and are capable of transporting themselves to other locations within their environment; there is no remote control. An autonomous robotic rover is an example of an autonomous mobile agent. An autonomous robotic rover makes decisions on its own by using environmental feedback obtained from its sensors. Robot programs, written by humans, tell the rover how to process this feedback to make decisions about its environment and itself.

Autonomous mobile robots can be developed as ground, aerial and underwater vehicles, and be designed to operate semi-autonomously. Semi-autonomous mobile robots operate with the aid of a human operator who can see what the robot sees through its camera, and can directly control the movement and actions of the robot. This paper describes robot programming using an autonomous robotic rover that is located in our Networked Robotics Laboratory and accessible via the Web.

In AI courses, both theoretical and practical aspects of intelligence are typically explored using traditional AI programming languages such as LISP and PROLOG. Robotics, however, offers a unique opportunity to see the abstract (theoretical) programming concepts "come to life" in the form of a robot agent interacting with the physical world.

AI Robotics, the AI approach to robotics, is the application of AI techniques (such as learning, planning, reasoning, problem solving, knowledge representation, and computer vision) to robots (Murphy 2000). AI Robotics is concerned with designing robots and devices that can move and react to sensory input. These robots normally operate in dynamic and unpredictable environments and must be able to adapt to their environment through the use of their sensors. Sensors can supply the robot with information about its environment (e.g., block A is on block B), and about its own state and condition (e.g., block C is in my gripper). There are several types of sensors (e.g., touch, light, temperature, audio, proximity and rotation) with which a robot can gather information.

1.3 Motivations

A Networked robotics laboratory can support education and public awareness by making valuable robot hardware accessible to a broad audience. The interdisciplinary nature of robotics, however, demands a compendium of basic skills from such diverse fields as mechanical engineering, electrical engineering, computer science, and artificial intelligence; skills that will prepare students for careers and help to alleviate the shortage of professionals in these areas. The networked robotics laboratory will provide students with experiences that will draw on this compendium of basic skills. Teaming is a program

of workshops and publications designed to encourage educational reform by changing the fundamental learning dynamics of the classroom (TRLI 2006). Through effective Teaming participants learn how to form a team and work effectively in a team environment to develop a fundamental workplace skill. Even the U.S. Department of Labor reported in, "Skills Employers Want," that what employers want most in today's workplace include: Verbal communication, Interpersonal abilities, and Teamwork (Carnevale, Gainer et al. May 1990).

1.4 Objectives

The goal of this research is to design, develop, and build a "Networked Robotics Laboratory" that will teach visitors a robot programming language and allow them to use that language to control the robot in the Robotics Lab. The objectives in achieving this goal are to resolve the networking, multimedia and real-time issues involved in bridging a network learning tool to the Robotics Laboratory, and to test the efficacy of students learning a robot programming language via the Networked Robotics Lab. Java network programming is used to develop specific client/server applications that define the communication between the components of the Robotics Lab. Along the way, specific robotics programming exercises were created, and the existing capabilities of the network learning tool were extended.

2. STATEMENT OF THE PROBLEM

In this chapter the problem of the shortage of engineers and computer scientists is discussed. The need for innovative pedagogical tools for robot programming is also explored as a way to address this problem. After exploring these issues, initial, final, and future systems operations are also presented.

2.1 Challenge in Engineering and Computer Science

The professions of Engineering and Computer Science are faced with the challenge of finding ways to address the shortage of qualified workers. The potential for this to become a long range problem is reflected in American institutions of higher learning. In particular, CS education is suffering from three serious crises. First, a huge fraction of those students seeking to major in CS drop the subject within the first two semesters of study. Second, the proportion of female and minority students in the CS group, low from the start, also drops dramatically during the first two semesters. Thirdly, even for the best CS students transition from student to professional is difficult.

For many academic institutions, CS educational practice encourages passive listening in large lectures, rather than active engagements. Instead, students must use the information, and learn to think critically about it, if they wish to cope effectively with novel and real world situations (Chiel 1996).

According to Standards for Technological Literacy: Context for the Study of Technology (ITEA 2000, 2002), there are six core concepts that technology educators should concentrate on for their students. These core concepts include: systems, requirements, optimization, trade-offs, processes, and control. The use of low-cost educational robots provides an excellent context for teaching students about many

concepts important to technological literacy. Designing, building, and programming robots require that students integrate control, electrical, and mechanical systems into a working device, and solve problems as a team. The result is students learn from each other. This requires students to think creatively and explore multiple possible solutions, and bring to bear information from multiple disciplines. This differs from finding the answer that is in the book, or the solution that the professor wants. Moreover, robotics has been shown by a number of researchers to be motivating and beneficial in teaching science and technology (Beer, Chiel et al. 1999).

We can no longer rely on importing technological talent to address this shortage of qualified workers. Regarding this shortage, Microsoft Chairman Bill Gates told Congress that overhauls of the nation's schools and immigration laws are urgently needed to keep jobs from going overseas. "The U.S. cannot maintain its economic leadership unless our workforce consists of people who have the knowledge and skills needed to drive innovation," Gates told the Senate committee that oversees labor and education issues. "We simply cannot sustain an economy based on innovation unless our citizens are educated in math, science and engineering," Gates said (Gates 2007). Effectively, there is a skills gap facing America -- the mismatch between the skills required for the new jobs being created in our country, and the skills of the workforce.

To help alleviate this shortage of qualified workers and to close the skills gap, a growing number of new academic collaborations and community technology initiatives (CTIs) are being formed to promote robotics and computer science education for African-American and female students. The Advancing Robotics Technology for Societal Impact (ARTSI) Alliance is a collaborative education and research community of predominately

African American robotics and computer science faculty that investigate how robotics can contribute to society and impact healthcare**,** arts**,** and entrepreneurship (ARTSI 2008)**.** Spelman College, a historically black college (HBCU) for women, is leading the alliance in partnership with Carnegie Mellon University, together with several other HBCUs and Research I (R1) institutions. The ARTSI Alliance is working to increase robotics education research at Historically Black Colleges and Universities (Spice and Watzman 2008).

The Intel Computer Clubhouse is a CTI project of Boston's Museum of Science in collaboration with the MIT Media Laboratory. The Computer Clubhouse provides a creative and safe after-school learning environment where young people from under-served communities work with adult mentors to explore their own ideas, develop skills, and build confidence in themselves through the use of technology (Clubhouse 2008).

FIRST (For Inspiration and Recognition of Science and Technology) is an organization founded by inventor Dean Kamen in 1989 in order to develop ways to inspire students in engineering and technology fields. The mission of FIRST is to inspire young people to be science and technology leaders, by engaging them in exciting mentor-based programs that build science, engineering and technology skills, that inspire innovation, and that foster well-rounded life capabilities including self-confidence, communication and leadership (FIRST 2008). FIRST sponsors an annual Robotics Competition that involves teams of mentors (corporate employees, teachers, or college students) and high school students. Teams collaborate to design and build a robot in six weeks using a standard "kit of parts" and a common set of rules. This robot is designed to play a game, which is designed by FIRST and changes from year to year.

FIRST LEGO League (FLL) is a result of an exciting alliance between **FIRST** and the **LEGO** Company. FLL is a program for students age 9-14 which combines a hands-on, interactive robotics program with a sports-like atmosphere (FLL 2008). FLL provides students with an opportunity to challenge their math and science skills in an internationally recognized competitive environment. Using LEGO bricks and other elements such as sensors, motors, and gears, teams gain hands-on experience in engineering and computer-programming principles as they construct and program their unique robot inventions. Each September, FLL announces the annual Challenge, which engages teams in hands-on robotics design and authentic scientific research. After eight exciting weeks of design and testing, the FLL season culminates at high-energy, sports-like regional competitions.

NASA's Robotics Alliance Project (formerly the NASA Robotics Education Project) is an effort to bring together students, engineers, private organizations and other government resources to pursue the goal of increasing robotics expertise in the U.S. The ultimate objective of NASA's Robotics Alliance Project is to expand the national resource of experienced, talented robotics experts that are available to develop future robotics systems needed by NASA and support the national investment in the robotics market (RoboticsAllianceProject 2008). In part, this is done by supporting a series of robotics competition programs that inspire students to become involved with technical fields, facilitating robotics curriculum enhancements at all educational levels, and developing a national clearinghouse for robotics education and career resources.

Over the past decade educators have developed a myriad of tools to help novices learn programming (Powers, Gross et al. 2006). These include *visual programming tools* (e.g., Karel Universe), *flow-model tools* (e.g., Raptor), *narrative tools* (e.g., Alice), *tiered language tools* (e.g., RoboLab) where novices can use more sophisticated versions of a language as their expertise develops, and *specialized output realizations* (e.g., Lego Mindstorms - the inexpensive robotics kit from LEGO).

This researcher believes the use of low-cost educational robots provide an integrative approach to problem solving that can be used in a wide range of CS courses, and can effectively help students make the transition to professionals. This study will address the challenge of declining enrollment of computer science and STEM-majors by developing a pedagogically sound web-based robot programming tool. The result of this research will be the creation of a Networked Robotics Lab that will make a difference in both the programming experiences and the academic lives of students. Specifically, this study will provide an additional tool to help novices learn programming and help ameliorate the current big problem in computer science education: attracting and retaining students.

2.2 Need for Innovative Pedagogical Tools for Robot Programming

Networked robots support education and public awareness by making valuable robot hardware accessible to a broad audience. The interdisciplinary nature of robotics, however, demands a compendium of basic skills from such diverse fields as mechanical engineering, electrical engineering, computer science, and artificial intelligence. These are the basic skills that will prepare middle school students for STEM-based specialized

high schools, and prepare high school students for college majors in these fields. The robot-programming laboratory will provide students with experiences to apply these skills.

2.3 Problem Statement

Existing stand-alone robotics laboratories that support educational computer programming-related activities are restricted in their operation physically, technically, and pedagogically. Physically: access is limited to the lab operational hours. Technically: there is a need for the presence of instructional and supervisory personnel. Pedagogically: there are delays resulting from the learning curve of visiting students. These restrictions reduce the effectiveness of stand-alone robotics laboratories. Moreover, typical online robot laboratories only allow visitors to teleoperate the robot (i.e., use remote control to directly operate the robot).

2.4 Proposed Solution

The networked robotics laboratory removes the restrictions of stand-alone robotics laboratories by making available a Web-based system for learning both about robotics and a robot programming language. Unlike typical online robot labs, the networked robotics lab will motivate learning by teaching visitors a robot programming language, and allowing them to program the lab's robot.

3. REVIEW OF RELATED WORK

The previous chapter stated the problem of the physical, technical, and pedagogical restrictions of existing stand-alone robotics laboratories. Among these were the need for innovative pedagogical tools for teaching robot programming, and the challenge of finding ways to address the declining enrollment of computer science majors by attracting and retaining students.

This chapter looks at related work with networked robots (i.e., online), and reviews some of the existing robot types, robot domains, robot programming systems, and existing network robotic systems. The focus is on a few particularly interesting and relevant concepts.

3.1 Robot Types

There are three types of robots: visible, virtual and unconscious. The 'visible type' or 'real existence' robot is a type of robot which has a concrete body in a real world. Typical examples are a humanoid robot and a pet robot. In contrast, the 'virtual type' robot works in a cyber world. It has a graphical representation, and interacts with a human user only through a display. The 'unconscious type' of robot is a type of robot embedded in the environment, such as roads, towns, rooms, and equipment. Its examples include a 'robotic room' which monitors people in the room and provides supports with actuators integrated with the room. The different types of robots can also be connected via ubiquitous networking (e.g., the Internet, and highway networks) to allow them to collaborate with each other. Together with various sensors embedded in an environment, a network of robots can provide services which cannot be realized with a single robot (NRF 2007). It is

important here to differentiate between a “network of robots”, and an autonomous “networked robot”; this study deals with the later.

3.2 Robot Domains

A "*networked robot*" is a robotic device connected to a communications network such as the Internet or LAN. The network could be wired or wireless, and based on any of a variety of protocols such as TCP, UDP, or 802.11 (IEEE 2007). There are two subclasses of Networked Robots: Tele-operated and Autonomous. Tele-operated robots allow human supervisors to send commands and receive feedback via the network. Autonomous robots use sensors to exchange data via the network.

Networked Robotics (or, Online Robotics) is a relatively new domain of robotics. Its growth path may be described as passing through the domains of: conventional robotics, teleoperation, and the more web-centric, networked robotics. Much of the work in Networked Robotics, however, remains in the teleoperation of robots (permitting the performance of physical work at a remote site under operator control) with very few references on autonomous online robotic control. There is even less work covering robot programming languages and none, that this researcher found, actually taught users how to program a robot prior to having them attempt to control it.

Networked robots pose a number of technical challenges related to network noise, reliability, congestion, fixed and variable time delay, stability, passivity, range and power limitations, deployment, coverage, safety, localization, sensor and actuation fusion, and user interface design (IEEE 2007).

3.3 Robot Programming Systems

In their survey of Robot Programming Systems (Biggs and MacDonald 2003), Geoffrey Biggs and Bruce MacDonald divide the field of robot programming systems into three sub fields: Automatic Programming, Manual Programming, Software (Control) Architectures.

Automatic programming systems offer robot users and programmers little or no direct control over the program code the robot will run. These systems are sometimes referred to as "online" programming systems since they often must be running while automatic programming is performed. Examples of automated systems include: learning systems, programming by demonstration (PbD), and instructive systems. PbD, the most common method of automatic programming, use touch/pendants or gestures and voice to demonstrate the task to be programmed. A pendant is a hand-held robot control and programming terminal that provides a convenient means to move the robot, teach locations, and run robot programs. The position of the pendant is recorded and the results are used to generate a robot program that moves the robot through this same motion. The teach pendant is a useful tool, allowing you to move away from the host computer terminal and control the robot locally.

Manual programming systems require the user to directly enter the behavior of the robot (i.e., programming the robot) before running it on the robotic system. This is typically done using a traditional programming approach such as a text-based or graphical (flowchart or diagram) programming language. Since a robot need not be present while programming, these systems are often called "off-line" systems.

Software (control) architectures consist of a device or set of devices used to manage, command, direct or regulate the behavior of other devices or systems. "The essence of a control mechanism is comparing information about what is desired and then adjusting devices or systems to make the desired outcomes more likely" (ITEA 2000, 2002). Software (control) architectures provide a set of principles for organizing a control system. They provide the underlying support, such as communication, constraints, and access to the robot itself. They also provide structure control for using sensors to acquire information about the environment, processing acquired information as required in order to decide about how to act, and executing actions in the environment. *Robot control* refers to the way in which the sensing and action of a robot are coordinated. There are four basic practical approaches to robot control being used today: 1) deliberative control, 2) reactive control, 3) hybrid control, and 4) behavior-based control. A good way to think about these alternatives is as follows:

- Deliberative Control: Think hard, then act.
- Reactive Control: Don't think, (re)act.
- Hybrid Control: Think and act independently, in parallel.
- Behavior-Based Control: Think the way you act.

Each approach has its strengths and weaknesses, and so no single approach is "the best" for all robots. Robot control, therefore, requires some unavoidable trade-offs.

The robot programming system developed for the Networked Robotics Laboratory can be described as a: software architecture, with an underlying manual programming system, that uses a generic procedural, text-based, programming language. The system can be used to demonstrate the various approaches to robot control. Although this manual Networked Robotics Programming System can be classified as "off-line", since the robot

isn't physically present at the programmer's location when it is being programmed, the robot will be executed "on line" via the Internet.

3.4 Existing Educational Networked Robotic Labs

Although there are several instances of existing educational network robotics laboratories, most of these systems only provide teleoperations.

The ***Red Rover, Red Rover Project*** was created by The Planetary Society and the LEGO Company in 1996 to inspire future generations of Mars explorers. Using LEGO® Red Rover, Red Rover kits, students design and build their own Mars Rovers. Students can then use a computer to guide their rovers across a simulated Mars terrain that they research and construct themselves. Students can also connect (via the Internet) to a Rover built by students at a different school next door -- or across the world -- and use it to simulate the teleoperations of robotic rovers in real Mars exploration missions. Thus, students are able to drive the remote rover and explore a completely new environment, just as scientists explore the strange surface of Mars (Red_Rover 2007).

Mars Stations is an online project that is the result of the success of Red Rover, Red Rover. The Planetary Society and LEGO Company have teamed together to establish a network of Mars Stations around the world to allow anyone with access to the Internet to connect to and remotely operate a rover online. Mars Stations simulate the appearance of a different location on Mars by using a scenic representation in which sculptured figures and lifelike details are displayed in miniature so as to blend indistinguishably with a realistic painted background. Each Mars Station contains a LEGO rover equipped with a Web camera.

Visitors are able to use the Internet to teleoperate these rovers, and explore the different dioramic scenes through the eyes of the robot (Mars_Stations 2007). Fig 3.1 shows a screen shot of the Mars Station Operator's Control Panel, with images captured through the camera on the Carl Sagan Memorial Mars Station. Visitors can drive the rover at The Planetary Society's website by using Rover Control Panel and viewing the results in the Rover Camera Window. They can also send messages to the Mars Station by typing into the Message Window.

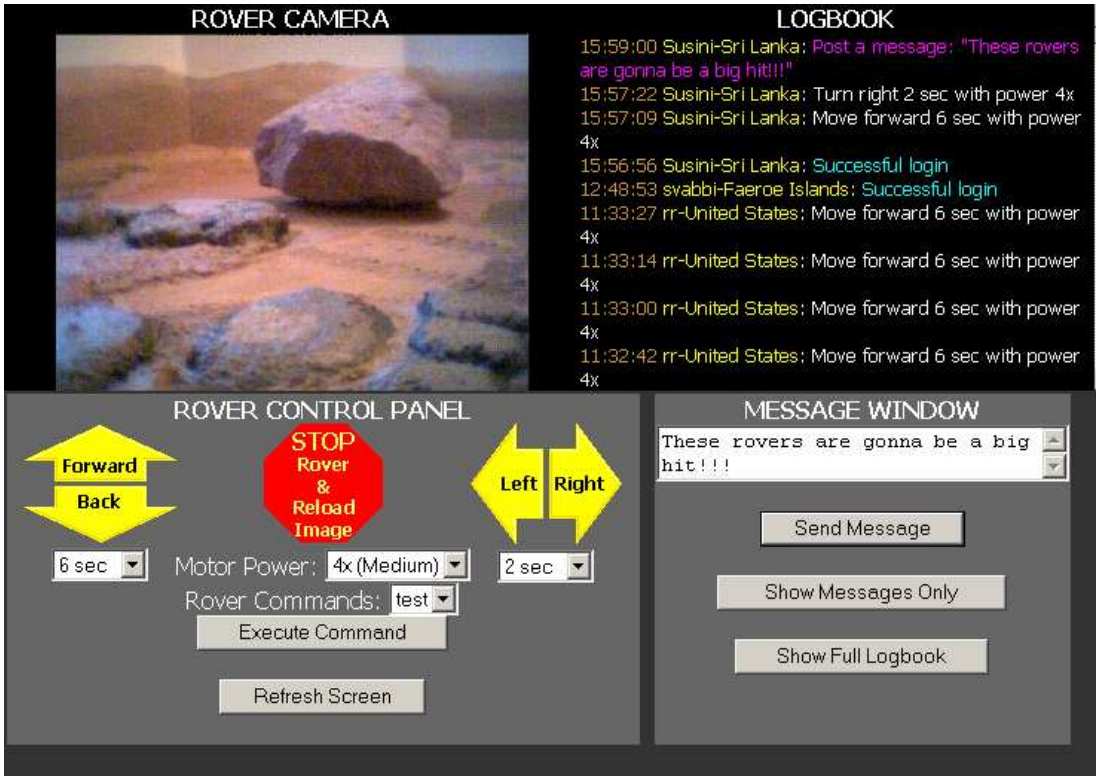

Fig. 3.1 Screen shot of the Carl Sagan Memorial Mars Station Operator's Control Panel

December 2006, Microsoft introduced ***Microsoft Robotics Studio***. Microsoft Robotics Studio is a new Windows-based development environment for creating robotic software for a wide variety of hardware platforms. Microsoft also introduced a new third-

party partner program featuring Microsoft Robotics Studio-enabled applications, services and robots from independent software vendors, service providers, hardware component vendors and robot manufacturers. The Microsoft Robotics Studio environment is an end-to-end, scalable and extensible robotics development platform that includes the following:

- A visual programming language (VPL) that enables nonprogrammers to easily program robots using a drag-and-drop environment
- A 3-D tool that simulates robotics applications in physics-based virtual environments
- A lightweight, services-oriented runtime that enables applications to communicate with a wide variety of hardware

Microsoft Robotics Studio understands a *robot* to be any system of related sensors and actuators with which electronic communication is possible. To interact with these robotic systems, Microsoft Robotics Studio facilitates the mapping between decoupled software modules and hardware components or subsystems of the robot (Chrysanthakopoulos and Nielsen 2007). Microsoft Robotics Studio interacts with a robot through the use of services. Since Microsoft Robotics Studio is implemented using .NET, robotics applications can be developed using a selection of programming languages, including those in Microsoft Visual Studio and Microsoft Visual Studio Express languages (Visual C# and Visual Basic). Microsoft Visual Programming Language (VPL) is the application development environment in Microsoft Robotics Studio that is designed on a graphical dataflow-based programming model rather than control flow typically found in conventional programming. Rather than a series of imperative commands sequentially executed, a dataflow program is more like a series of workers on an assembly line who do their assigned task as the materials arrive. As a result, VPL is well suited to programming a

variety of concurrent or distributed processing scenarios. Fig 3.2 shows a sample VPL diagram for simple bump-turn-go wander behavior.

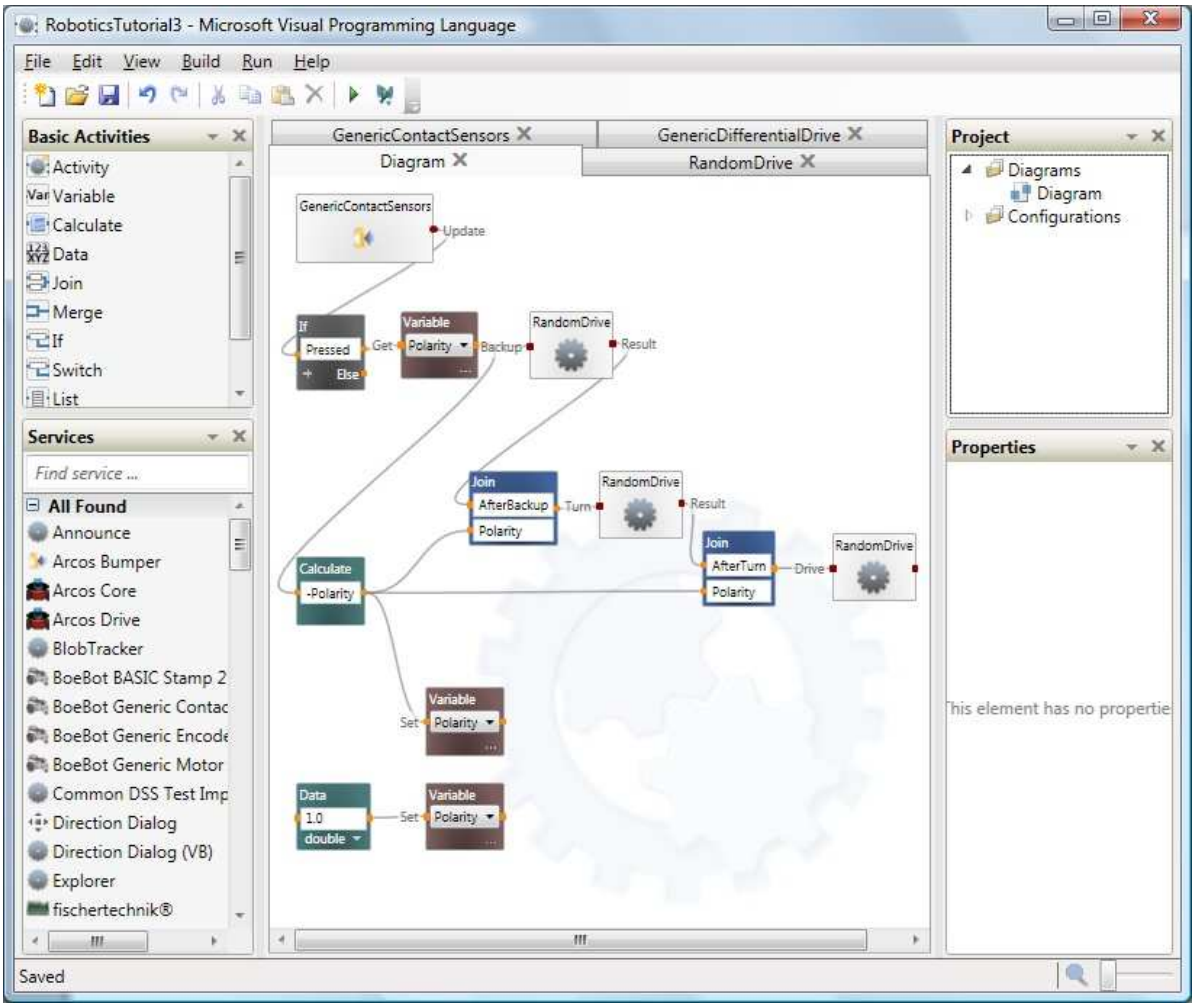

Fig 3.2 Sample VPL diagram for simple bump-turn-go wander behavior

Writing an application using the Microsoft Robotics Studio is a simple matter of orchestrating input and output between a set of services. Services represent the interface to software or hardware and allow you to communicate between processes that perform specific functions (Microsoft 2006). VPL is targeted for beginning programmers with a basic understanding of concepts like variables and logic. Online tutorials introduce uses of

Microsoft Robotics Studio which shows implementations of services on different robots, and include examples of services running in C#, Visual Basic, and IronPython – the .NET compatible version of the Python language. The limitation is that Microsoft Robotics Studio requires that all services run in an environment in which a full .NET library implementation is present. Since many onboard robot processors do not have the ability to host such an environment, robots must be tethered through wire or wireless communication, to a controlling computer acting as the robot brain (Jackson 2007).

Virtual Reality Robotic Programming provides students an understanding of programming control concepts by using a real-time computer-simulated environment. "A VR system is one that gives the user an experience of being immersed in a synthesized environment" (Earnshaw, Gigante et al. 1993). VR allows a user to interact with world, real or imagined where the actual movements of the robot can be viewed, all in a three-dimensional world (3D) on a personal computer.

Robocell, from Intelitek, is a software product for robot control (Robocell 2008). Robocell integrates SCORBASE robotics control software with 3D solid modeling software. Robocell lets students create, program, simulate and control the entire operation of robotic workcells and flexible manufacturing systems (FMS). The 3D graphics-display module provides dynamic simulation and tracking of the robot and devices in the workcell. Robocell is based on the SCORBASE-ER 4u robot with all the operating functions of the actual robot. Robocell acts as an off-line programming device for creating Robot programs and downloading to real Robots. Fig. 3.3 shows a VR world robot on a PC screen, and how it looks identical to a real robotic arm. Fig. 3.4 shows a screenshot of the Robocell/SCORBASE robotics control software.

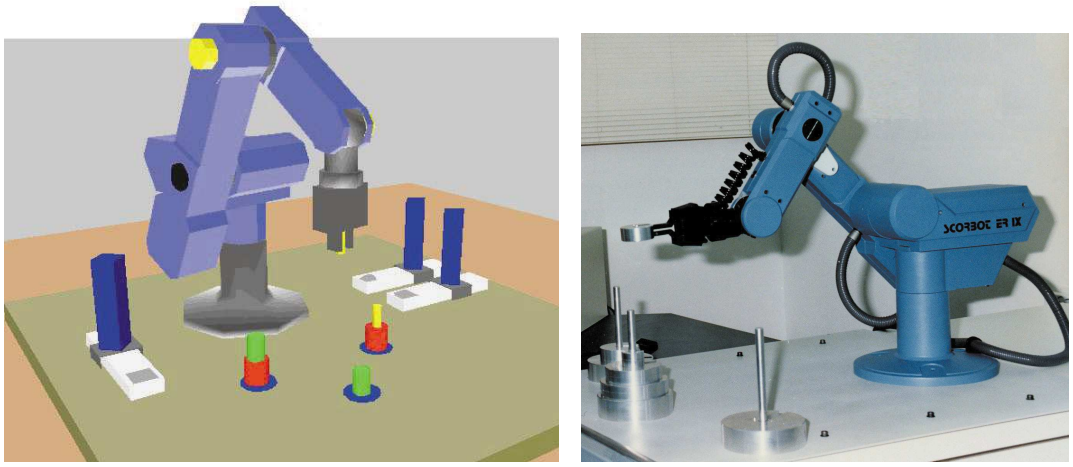

Fig. 3.3 VR world robot (shown left) and real robot arm (shown right)

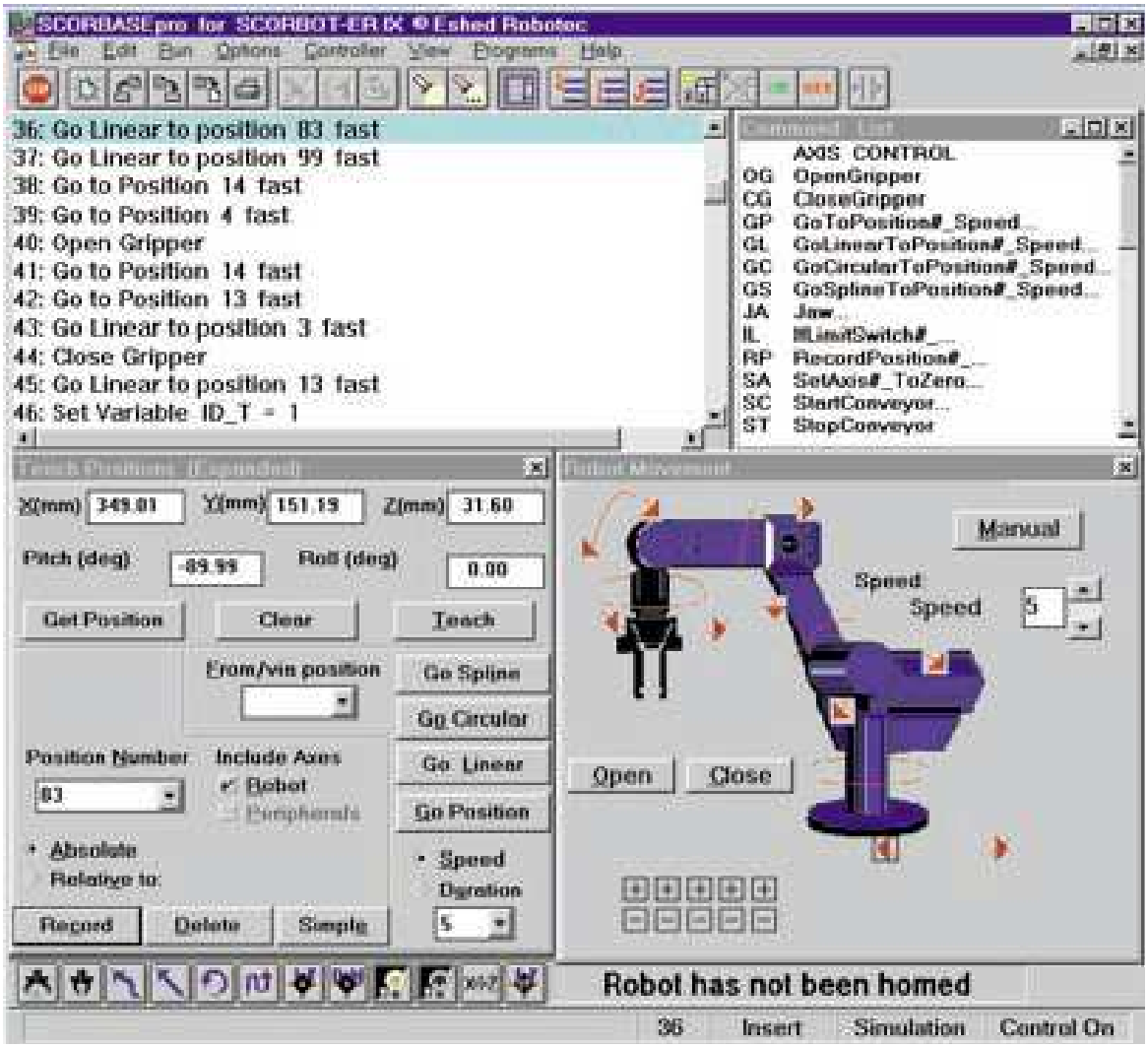

Fig. 3.4 Screenshot of Robocell/SCORBASE robotics control software

VR Robot, from Denford, is another software product for robot control (VR Robot 2008). VR Robot features a split screen that displays the VR teachbox on the left-hand side with the VR Robot on the right. The student operates the VR Robot by using the mouse to program moves via the on-screen teachbox. VR Robot is based on the Mitsubishi RV-M1 Robot with all the operating functions of the actual robot. VR robot acts as an off-line programming device for creating Robot programs and downloading to real Robots.

The advantage of virtual reality robotic software is that they offer the control features and the look and feel of programs used in the real world, and low-end, industrial grade or high end, industry-like robots may be added later for final (real world) testing of the student's program. The disadvantages are virtual reality robotic software is more expensive than low-cost educational robot systems, and provide less of a hands-on feel to the testing of the programs (Geissler, Knott et al. 2004).

3.5 Tools for Teaching Introductory Computer Programming

Computer science educators have developed several computer programming tools to help students learn programming (Powers, Gross et al. 2006). Each tool has features, or combination of features, that support the prototypical model upon which the programming tool is based. These features include: narrative tools, visual programming tools, flow-model tools, tiered language tools, and specialized output realizations.

Narrative tools, such as Alice, support programming to tell a story. Alice is an innovative 3D interactive animation environment for introducing novices to object-oriented programming. Alice makes it easy to create an animation for telling a story, playing an

interactive game, or a video to share on the web. Storytelling Alice is a programming environment designed to motivate a broad spectrum of middle school students (particularly girls) to learn to program computers through creating short 3D animated movies (Kelleher 2006). Alice uses 3D graphics and a drag-and-drop interface to facilitate a more engaging, less frustrating first programming experience. Below, Fig. 3.5, is a screenshot of the Alice 2.0 Interface.

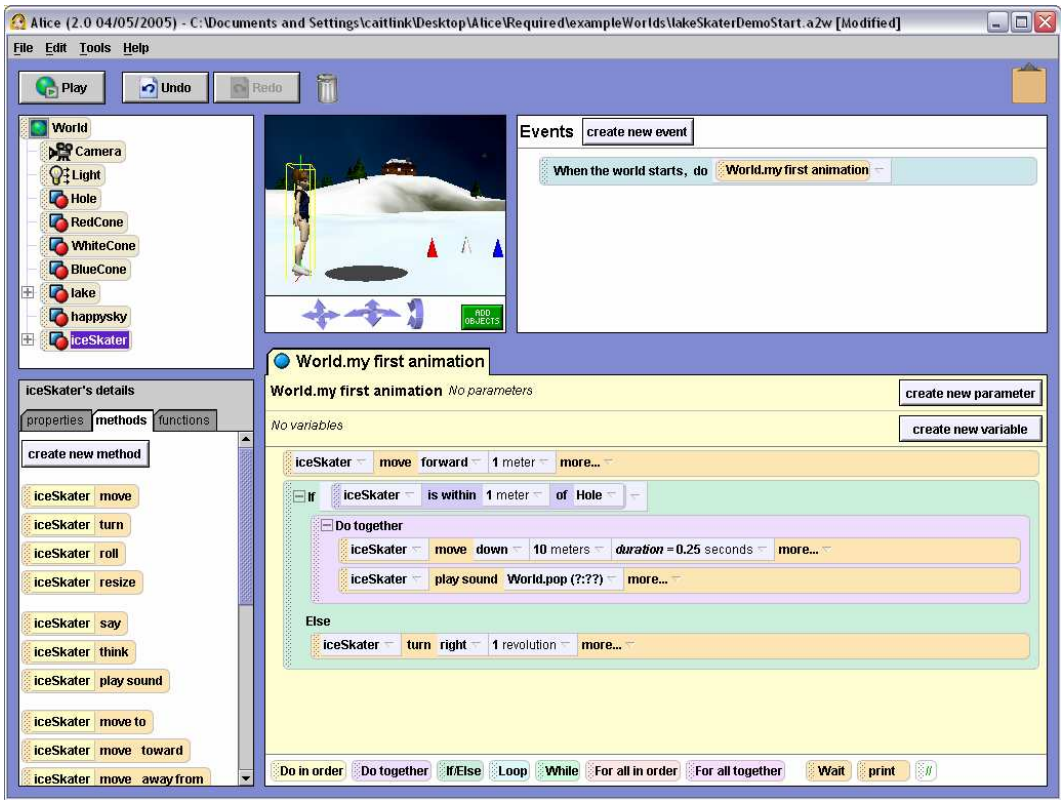

Fig. 3.5 Screenshot of the Alice 2.0 Interface. Beginning with 1) the top center window, the world window provides a view of the virtual world that a students' program will control. Moving counterclockwise, 2) the object tree contains a list of the 3D objects in the virtual world. 3) The details area shows the properties, methods, and functions for the object selected in the object tree. 4) The methods editor shows the code that defines a method a student is working on. 5) The events area allows students to call methods based on events in the world, such as mouse clicks or changes in the value of a variable.

Karel Universe is an example of a visual programming tool that supports the construction of programs through a drag and drop editor integrated with the Karel J Robot simulator system. Karel Robots (Pascal, C++, and Java versions) are robot simulators that afford a gentle introduction to computer programming. The Karel J Robot system is intended for those students who wish to learn Java with the absolute minimum of syntax (Bergin 2006). Students can create classes, objects, and programs by dragging syntactically correct program fragments from one pane and dropping them to another. The resulting programs may be then executed in the Karel J Robot simulator.

The structure of a Karel J world is a great flat plane with the standard north, south, east, and west compass points. The world is bounded on its west side by an infinitely long vertical wall extending northward. To the south, the world is bounded by an infinitely long horizontal wall extending eastward. Crisscrossing the world are horizontal streets (running east-west) and vertical avenues (running north-south) at regular, one-block intervals. Fig 3.6 is a picture of the Karel J Robot in its rectangular world (Karel J is shown at point (1,1), the beeper is located at (4,4)). Karel's task is to climb the stairs, and pick up the beeper at the top of the stairs. The methods Karel uses to perform this are shown on the right side of the diagram.

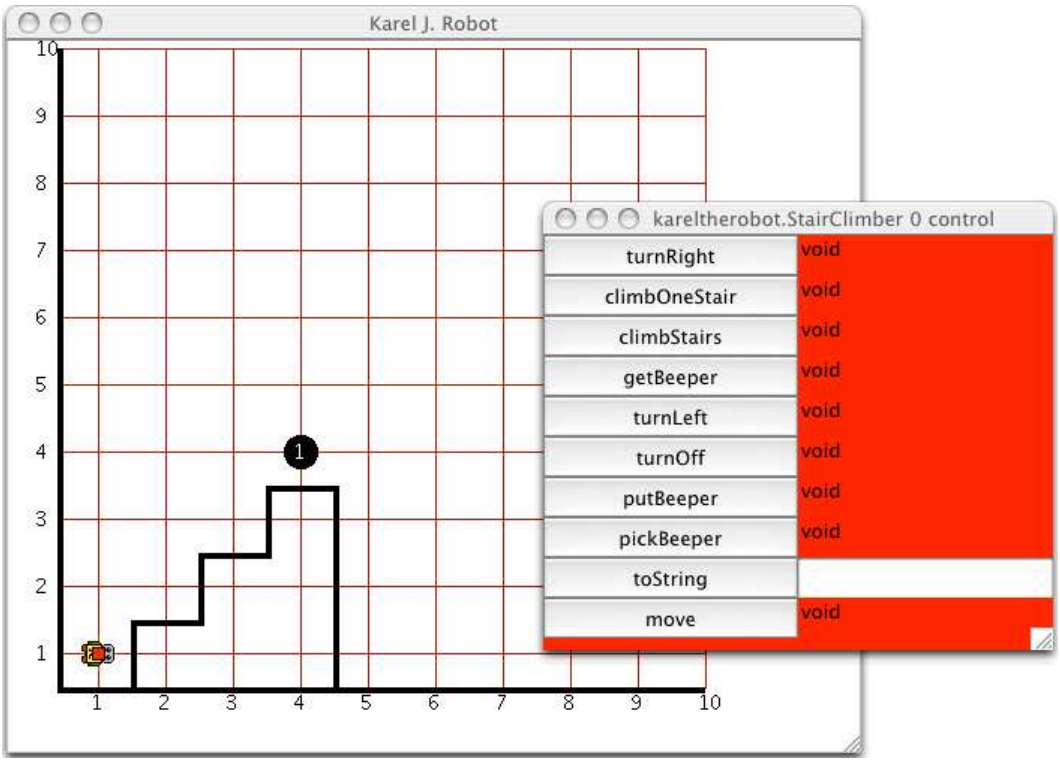

Fig. 3.6 Picture of the Karel J Robot in its rectangular world

Flow-model tools, such as RAPTOR (the Rapid Algorithmic Prototyping Tool for Ordered Reasoning), provide a flowchart-based visual programming environment, designed specifically to help students envision their algorithms and avoid syntactic baggage (Carlisle, Wilson et al. 2005). RAPTOR allows students to create algorithms by combining basic graphical symbols.

RAPTOR has 6 basic statements: Input, Output, Assignment, Call, Selection, and Loop. Each of these statements is indicated by a different symbol in RAPTOR as shown below. Fig. 3.7 is a screenshot of the RAPTOR programming environment.

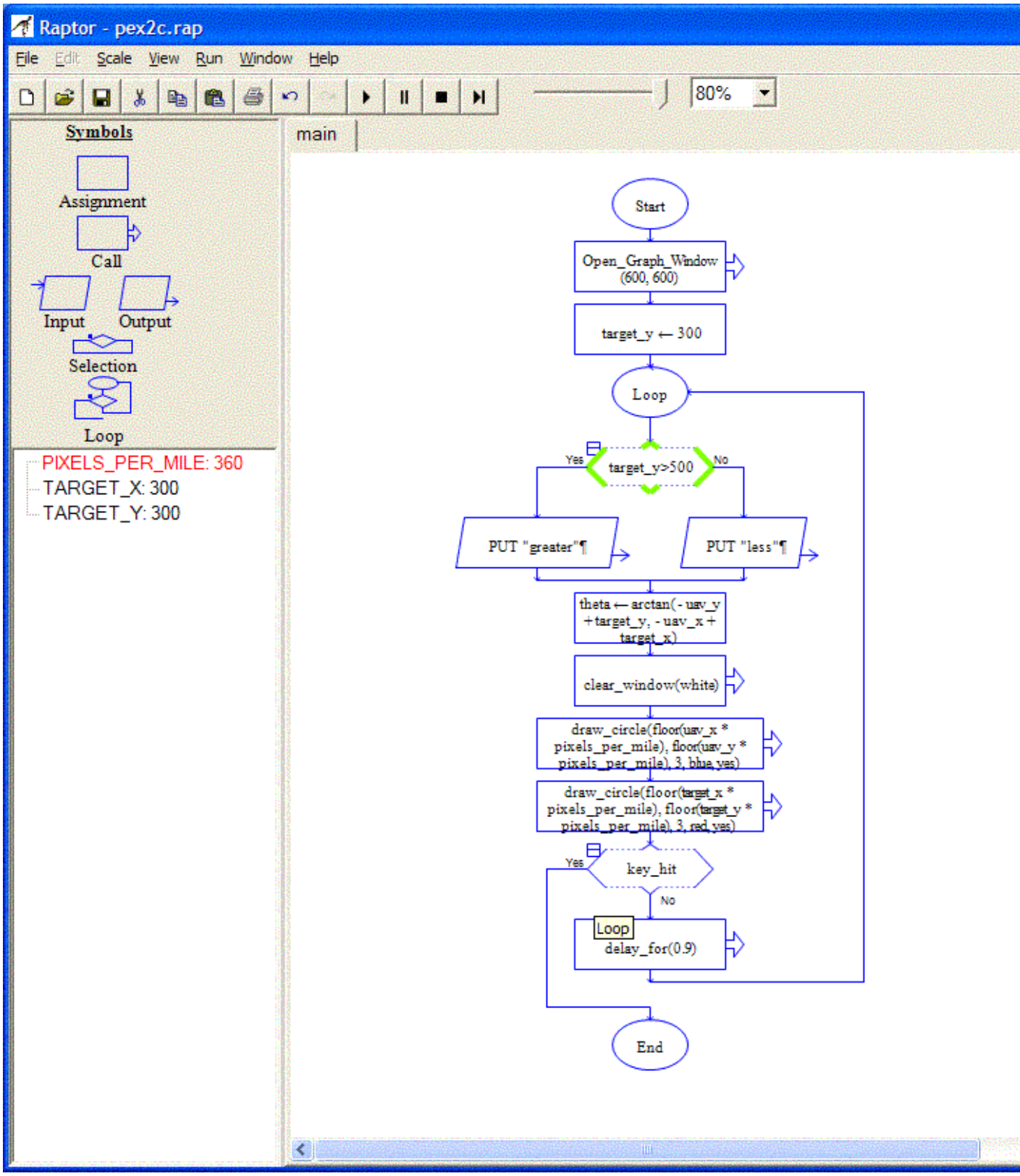

Fig. 3.7 Screenshot of the Raptor (Rapid Algorithmic Prototyping Tool for Ordered Reasoning) Programming Environment

Tiered language tools, allow novices to use more sophisticated versions of a language as their expertise develops. *ROBOLAB* is a graphical programming environment powered by LabVIEW, a powerful programming environment used by engineers. During NASA's 1997 Mars Mission, LabVIEW was used to monitor the Sojourner Rover's position and status during its exploration of the planet's surface on the Mars Pathfinder Mission (Rogers 2001).

While the power and complexity of LabVIEW is essential for NASA level projects and scientists, children and LEGO components demand less power and a less complicated interface. Robolab has three tiered levels of programming: Pilot programming, Inventor programming, and Investigator programming. Fig. 3.8 shows the screenshot for accessing the Pilot and Inventor programming tiers of Robolab. Robolab allows users to drag and drop graphical blocks of code that represent commands such as left and right turns, reverse direction, motor speed, motor power, and so on. Users can drag the icons together into a stack, in a similar way to assembling physical LEGO bricks, and arrange them in logical order to produce new behaviors for a robotic construction (Fig. 3.9).

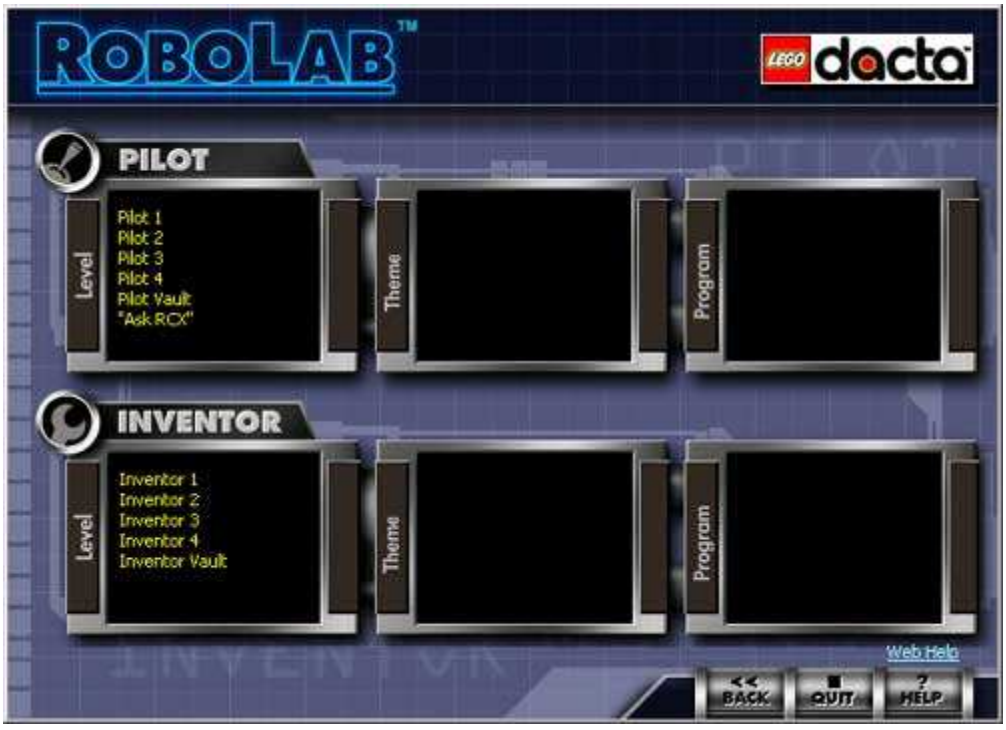

Fig. 3.8 Screenshot for accessing the Pilot and Inventor tiers of Robolab

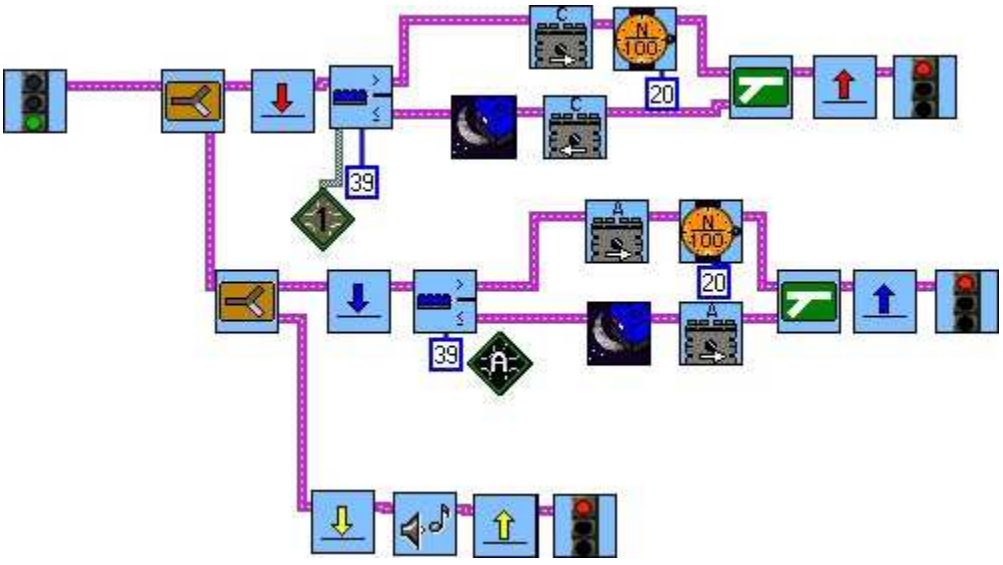

Fig. 3.9 A simple *ROBOLAB* program using Inventor Programming

Specialized output realizations tools provide unique realization of program output. Lego Mindstorms - the inexpensive robotics kit from LEGO – has been used in classrooms from kindergarten to graduate school.

Learning with robots facilitates the teaching of scientific and mathematic principles through experimentation with the robots. Rogers and Portsmore 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. The specialized output realization of the moving robot enabled them to discover that 1.4 is less than 1.6 because their robots went further when they told it to travel 1.6 seconds. They 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 (Rogers and Portsmore 2004). Seymour Papert used robots to teach concepts of a new kind of geometry -- "turtle geometry"-- by creating robots with a pen which could be programmed to draw geometric shapes on paper (Papert 1980). His students were able to see the relationships between programming, mathematics, and the movement of the robot within the specialized output realization of the "LOGO environment". Fig. 3.10 shows two robots built using the Lego Mindstorms NXT Robotics Kit.

Fig. 3.10 Two robots built using the Lego Mindstorms NXT Robotics Kit

3.6 Teaching STEM and Computer Science using Robotics

Robotics has been shown by a number of researchers to be motivating and beneficial in teaching computer science and STEM–related topics. In fact, the availability of low-cost robotic platforms, such as Lego Mindstorms, has led to the proliferation of robotics courses at a growing number of schools. Many of these schools use robots in their introductory computer science courses, while others use robots to introduce students to AI concepts.

The experiential hands-on education of robotics also provides an excellent tool for learning general STEM topics, pre-CS1 for majors and students considering a CS major, teaching computer science, teaching artificial intelligence (AI) concepts, and K-12 outreach and enrichment. In this regard, students learn best when they are able to apply the subjects they learn in school and use those subjects in a way that is meaningful. By designing, building, programming, and documenting robots, students use science, engineering, technology, math, and writing skills in a hands-on project that reinforces their learning (BotBall 2007).

Teaching general STEM topics:

Science, Technology, Engineering and Mathematics (STEM) standards have been addressed through robotics education. This has included hands-on learning experiences to generate interest in math and science, engineering, motivate students, and reinforce STEM content learning. The use of robots helps to illustrate abstract concepts by exploring topics such as mechanical design, computer programming, digital and analog sensors, and Robotic Vision that--before the integration of such devices into the classroom--had proved difficult to teach.

Teaching pre-CS1 for majors and students considering a CS major:

Robots can be incorporated into a pre-computer science course to inform and excite students about the possibilities of professional specializations for computer science majors. Robotics and artificial intelligence provide a wide variety of career options: industry, government, consulting, teaching, research, marketing, sales, and management are available to graduates of a computer science program.

Teaching the introductory computer science courses:

Introductory computer programming courses can leverage robots and *reprogram the art of computer programming*. Programming robots can now be used to teach the principles of computer programming. Robot programming languages support a variety of programming paradigms, including: procedural, object-oriented, functional, logic, and behavior-based programming.

Teaching the AI concepts:

One approach is to conceive of topics in AI as robotics tasks. -- In a robotics laboratory, students can build their own robots and program them to accomplish the tasks. By constructing a physical entity in conjunction with the code to control it, students have a unique opportunity to directly tackle many central issues of computer science including the interaction between hardware and software, space complexity in terms of the memory limitations of the robot's controller, and time complexity in terms of the speed of the robot's action decisions. More importantly, the robot theme provides a strong incentive towards learning because students want to see their inventions succeed (Kumar 1992).

Teaching K-12 outreach and enrichment courses:

Research has shown that gender and cultural issues, as they relate to students' learning styles, subject areas and occupational choices begin forming in early childhood and tend to get re-enforced along the way. Too often, negative attitudes and perceptions of robotics, and STEM-related subjects are learned and re-enforced. In July 1992 Mattel released "*Teen-Talk Barbie"*, which spoke a number of phrases including, "Will we ever have enough clothes?", "I love shopping!", and "Wanna have a pizza party?" Each doll was programmed to say four out of 270 possible phrases, so that no two dolls were likely to be the same. One of the phrases, that created a lot of controversy was, "Math class is tough!" Although only about 1.5% of all the dolls sold said the phrase, it caused a public outcry (Mattel 1992). Since early interest and success in mathematics has been found to be correlated to choosing a career in science, could the "Teen-talk Barbie Syndrome" be attributed to the under-representation of women in scientific disciplines? In October 1992

Mattel announced that *Teen-Talk Barbie* would no longer say the phrase, and offered a swap to anyone who owned the doll.

3.7 Creating A Culture of Expectancy

In addition to the "Teen-talk Barbie Syndrome", there is a common belief that girls are less mathematically capable than boys (Eccles 1987). This belief is mirrored by student's parents and teachers who often expect large discrepancies between boys' and girls' performance in math class (Lummis and Stevenson 1990). Because other's expectations can have a strong influence on one's attitudes and behavior (Tocci and Engelhard 1991), parents' and teachers' negative expectations put girls at a distinct disadvantage in the classroom. Given these factors, it is not surprising girls show less confidence in their ability to learn mathematics than boys do, even when they are successful in school (Meyer and Koehler 1990).

In spite of these social and cultural beliefs, exposure to STEM-type activities can decrease the acceptance of traditional gender roles. Short programs that last as little as one day, like The Robot Roadshow Program, can further interest about science education. The Robot Roadshow Program use robots to excite children (both boys and girls) from rural or underserved schools in Kansas, and interest them in technical disciplines (Matson and DeLoach 2004). This program used a three-step process: 1) Pre-visit workbooks, 2) the visit and presentation, 3) and a follow-up session with the faculty to evaluate student impact. The program was tailored by age groups: K-3, 4-6, 7-9, 10-12. Each student within their respective group received a workbook that developed the appropriate set of skills and knowledge necessary to get the most from the program upon the visit. One of the program's

goals was to allow students to build a relationship between the study of math and science and interesting subjects (robots).

Educational robotics has also been successfully used to engage African American Students with technology. In two 2003 summer programs (Science and Technology Entry Program – STEP, and Playing2Win – P2W), demonstrated educational robotics has the potential to attract this population of students and to provide an effective and motivating learning experience (Goldman, Eguchi et al. 2004).

While programs like those described above have been successful at getting girls and African American students to participate in STEM areas, an overall question still remains: *Does participation in such educational programs have a long-term impact on the perception of girls' and African American students' perception of their abilities and interests in STEM areas?* For girls, this question is answered affirmatively. Girls' attitudes towards a career in science increased as a result of participation in the annual seven-week KISS Institute for Practical Robotics' Botball Program. Botball engages thousands of middle and high school students in regional and national robotics competition in a team-based activity that includes a corporative component and a competitive component. The robot teams were comprised of both all-girl teams, and mixed-gender teams. Teachers mentored students in the development of a project.

The participation in Botball resulted in an increase in positive attitude toward engineering, by decreasing the acceptance of traditional gender roles (Weinberg, Pettibone et al. 2007). Mentors played a vital role. It was found that girls in mixed-gender teams with good mentors experienced an increase in self-concept and increased expectations of success

in science and math due to the program. This indicates short-term, well-structured programs can effectively modify social and cultural beliefs

The Expectancy Value Theory model developed by Jacquelynne S. Eccles and Allan L. Wigfield describe how these gender and cultural stereotypes about different subject areas and occupations, and the broader cultural milieu in which individuals grow up, influence their expectancies and values (Eccles and Wigfield 2000).

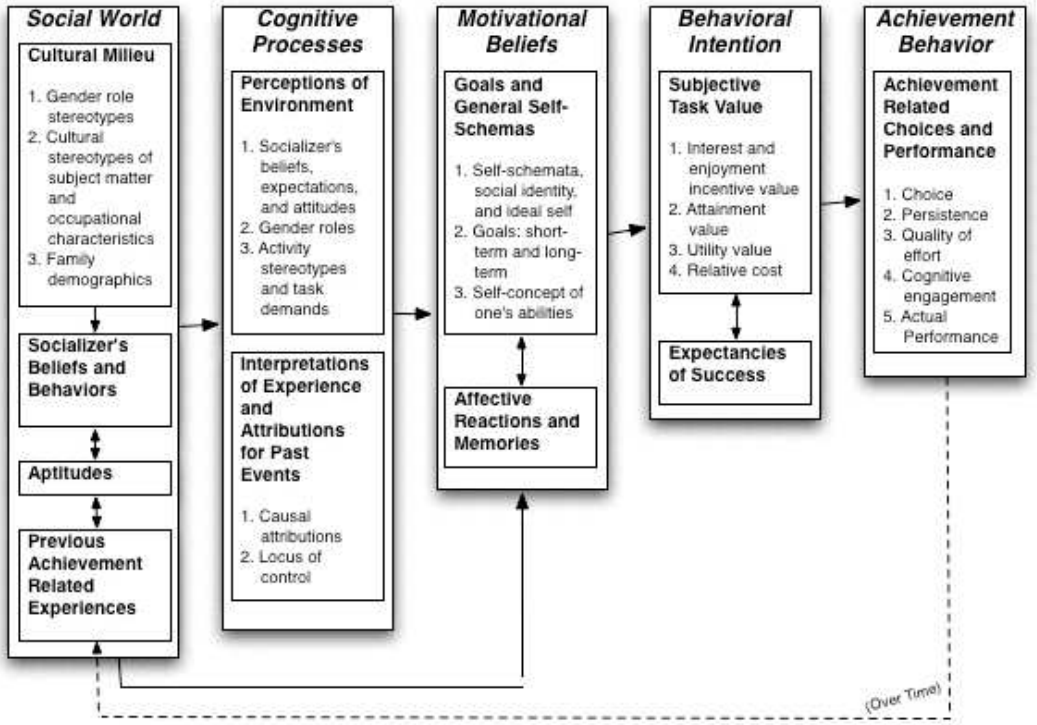

Fig. 3.11 Simplified Version of Eccle's Expectancy-Value Model of Motivation

In this model, shown in Fig. 3.11, achievement behavior is predicted by two components: expectancy and subjective task value, which, together, represent the intention to approach or avoid engaging a task, and, once engaged, the quality and quantity of effort.

Though task value and ability self-concepts (i.e. expectancy) can be independently measured, they influence each other. Ability self-beliefs and task beliefs for a subject are often related reciprocally (Eccles and Wigfield 2002).

Beliefs are "people's judgments of their capabilities to organize and execute course of action required to attain designated types of performance". *Expectancy*, according to Eccles and Wigfield, is a person's self-evaluation of his or her ability and beliefs about the probability of success on an upcoming task, whether in the immediate or longer-term future. The *achievement (subjective) task value*—beliefs about value of doing the task—is the other reason why an individual wants to engage in an undertaking. Thus, expectancy-value theory considers that individuals' choices are directly related to their "belief about how well they will do on an activity and the extent to which they value the activity".

As shown above, the early participation of girls and African American students in STEM-related activities can result in an increase in positive attitude toward engineering, by decreasing the acceptance of traditional gender roles.

The part of this study that involved K-12 students focused specifically on three of the Expectancy-Value Model's constructs that have received the most research attention: beliefs about ability, expectancies for success, and achievement task values. To insure the success of participants, an attempt was made to ***Create A Culture of Expectancy*** … an environment that provides an effective and motivating learning experience; where students' beliefs, expectations, and self-confidence for success are supported by caring mentors.

4. THE NETWORKED ROBOTICS LABORATORY: ARCHITECTURAL COMPONENTS

In the previous chapter, a review of related work was presented. This chapter introduces the components of the robot-programming laboratory, the NQC robot-programming language, and *CodeLab*: the asynchronous distance learning network tool (network learning tool).

There were two major activities associated with this project: technical development and content development. The technical development involved identifying the architectural components for the robotic programming laboratory, and integrating these components to form a complete and functioning system. Content development involved creating exercises for CodeLab that were specified clearly and unambiguously, and were amenable to some logical testing. In addition, there are exercises whose success or failure can be determined by looking at a video and comparing it with a video of a robot carrying out the correct program.

4.1 Systemic overview of the robotics lab

Computer Architecture is the art and science of determining the functional, performance and cost goals of the user of a structure (analysis), and then selecting and interconnecting hardware and software components (design and development) to meet those needs as effectively as possible within economic and technological constraints. This study uses CodeLab, a web-based programming system for learning general-purpose computer programming languages. The traditional architecture of CodeLab involves multiple separate, typically remote, *testing* servers that have the responsibility for taking

the student code submission, incorporating it in a code harness, compiling and executing the resulting program, and generating a response to the student (Arnow and Harris 2007).

The servers are responsible for different languages, and so supporting a robotics programming language in the context of CodeLab means constructing a suitable testing server: a process that can receive robot programming code fragments (for a specific robot programming language), and with suitable harnesses build an executable robot program and run it on a robot. CodeLab also supports fill-in-the-blanks type exercises that allow questions on robotics fundamentals and programming concepts.

Since the robot in the networked robotics lab is controlled by programs written in NQC ("Not Quite C"); a programming language designed to program Lego robots, an NQC CodeLab testing server was constructed. The NQC testing server is itself divided into two servers: the CodeLab front-end which interacts with CodeLab and plays the role of a testing server and the robot back-end, which directly manages the robot and the camera.

Programming robots, however, is different from programming computers. Although robots may use a computer as a building block, a robot is able to interact with its world (physical, or virtual). Because of this interaction, robots require programming languages that interpret information gathered from sensors and respond by turning on or off motors. In this regard, the value of robotics programming instruction lies in its concrete character, and its service as a bridge between operational and abstract levels of thinking. This exposure to concrete realization of concepts seems to trigger a kind of epiphany in some students, and this experience has consequences far beyond the robotics course.

The networked robotics system verifies the correctness of NQC programs, and performs remote testing of the networked robot as follows: Students submit a robot programming exercise to an NQC-based CodeLab module. An NQC cross compiler is used on the CodeLab server to check the syntax, and offer some semantic checks as well, before executing them on the robot. When the program checks out, CodeLab then acts as a client, contacts the robot server in the networked robotics lab, and submits the compiled NQC program, along with various identifying information. The robot server loads the program into a robot and lets the robot execute the program. A video camera then records the action, and the server saves the short video file. The server then contacts CodeLab, passing the video and the identification information. Finally, CodeLab makes the video available to the student. Fig. 4.1 shows the Flow Diagram for this System Architecture.

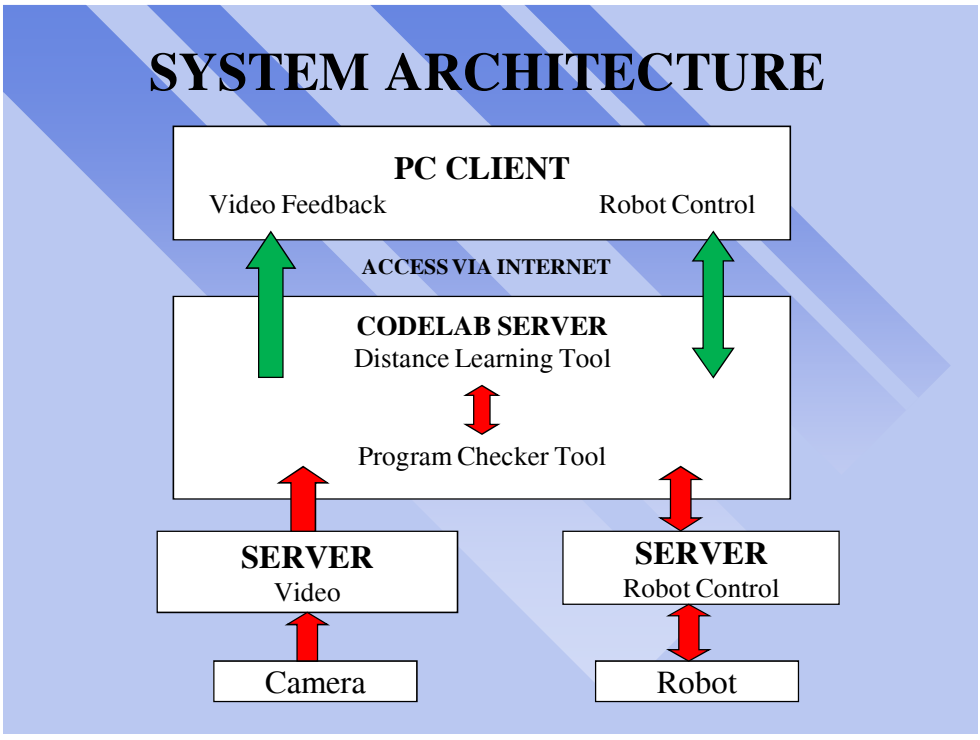

Fig. 4.1 Flow Diagram for the System Architecture

4.1.1 The Robotics Invention System

The *LEGO MINDSTORMS ROBOTICS INVENTION SYSTEM* (RIS), shown in Fig. 4.2, is a set for building robots that is sold by the LEGO Group and is used as a teaching aid for the Robot-programming laboratory.

Fig. 4.2 LEGO MINDSTORMS ROBOTICS INVENTION SYSTEM (RIS)

This Lego robotics set eliminates the difficulties of building mobile robots by including the five fundamental robot components into one kit of interlocking LEGO pieces. These five components are: controller, end-effector, arm, actuator, and sensor (Appendix: A.2 The Anatomy of a Robot). The RIS is an instance of the class of Rug Warrior robots used in the MIT Mobile Robot Lab. Rug Warrior robots consist of a microprocessor, 32K bytes of memory, a serial port, two motor actuators, a piezoelectric buzzer, and a number of sensors (Jones, Flynn et al. 1998).

4.1.2 RCX controller

The *RCX* (Robotic Command Explorer) is a small computer that is based on the Hitachi H8/3292 microcontroller. The RCX is packaged in a palm-sized LEGO brick and functions as an electronic brain that controls the action of the robot. This programmable LEGO brick is able to store and execute programs downloaded from a personal computer via an *infrared communication link* (IR). It contains 16K of internal ROM and 32K of static RAM with 6K reserved for "user memory" (i.e., writing programs for the RCX). The RCX can be in either one of two modes: Boot Mode or Full Function Mode. In Boot Mode, programs cannot be downloaded to the robot because the RCX needs firmware. Firmware is, essentially, an operating system for the RCX. It allows communication between the personal computer and the RCX to occur.

Part of the firmware is stored in ROM and is always present. The other part must be downloaded from the IR port and stored in RAM the first time the RCX is turned on. When the firmware is present, the RCX is in Full Function Mode and can receive robot programs over the IR port and run them. Programs written for the RCX do not contain native code for the CPU. Instead, they consist of special bytecodes, which are interpreted by the firmware. This allows the RCX to maintain a safe and reliable execution environment for programs.

Each RCX has three numbered *input sensors* (to sense events in its environment) and three lettered *output ports* (to react to the sensing data and make motors turn on and off). There is a small LCD view *screen* to display helpful information such as sensor readings and output port settings. Four *front panel buttons* are for turning the robot on or off (On-Off), and selecting (Prgm), starting (Run), and stopping programs. The RCX

displays values (View) read from a sensor for an input port, and displays the power value being sent to an output port. The RCX can generate *sound* and is capable of producing beeps of different frequencies. Fig. 4.3 shows the Robotics Command Explorer, or RCX.

Fig. 4.3 The Robotics Command Explorer (RCX)

4.1.3 Using sensors to control output motors

The RCX's three sensor ports (labeled 1, 2, and 3) can accommodate one of four different LEGO sensors: touch sensor, light sensor, rotation sensor, and temperature sensor. The type of sensor attached to it will determine the way in which the RCX interacts with the sensor. While the RCX will passively read a touch sensor, it must supply power to a light sensor. The RCX, however, has no way of knowing what type of sensor is connected to each sensor port. Since each type of sensor has unique requirements for reading and interpreting its values, the RCX must configure each sensor port before it can be used. This can be done using some appropriate programming language (Baum 2000).

The robot's actuators (motors or lights) can be attached to the RCX's three output ports (labeled A, B, and C). Each output has a mode, direction, and a power level associated with it. To fully determine an output's actions, the mode, direction, and power level can be set explicitly by programming the RCX.

The three output modes are *on*, *off*, or *floating*. When in the on mode, any motor attached to the output will be running. The off mode will force the motor to come to an abrupt stop; the motor will also be locked (unable to spin). In the floating mode, the motor will be allowed to coast, or float, to a stop. Although the motor will be allowed to spin freely, there is no power to the motor in floating mode.

The direction of an output can either be forward or reverse and only has effect when the output is on. The output can also be toggled to change the direction to the opposite of whatever it was previously. While the output is set in either of the off or floating mode, the direction settings are remembered for the next time the motor is turned on, and can also be modified.

The power level of an output determines the speed at which an attached motor spins. Programming the RCX allows the power level to be adjusted to one of eight settings between 0 and 7. 7 is the fastest, 0 is the slowest (but, the robot will still move). As with the direction setting, the power level only has effect when the output is turned on, but is remembered and may be altered when the output is turned off.

4.2 NQC

Not Quite C (NQC) is a programming language written by Dave Baum that was especially designed for the LEGO robots. NQC is an easy to use command line based

tool. Normally, typing an appropriate command into an MS-DOS window runs NQC. NQC source code is stored in simple text files. NQC compiles these source files to byte code and can download them to the RCX using the IR tower. NQC runs on MacOS, Linux, and Windows.

4.2.1 NQC language overview

NQC is a simple text-based language with a C-like syntax that is especially designed to program Lego's RCX robot. The preprocessor and control structures of NQC are very similar to C. NQC provides subroutines, #define macros, complex expressions, a standard set of operators and control structures. There are also commands for IR communications (to control motors and read sensors), timers, counters, data logging, and a host of miscellaneous functions. Programs are written in an editor, then compiled and downloaded to the RCX. An NQC program structure is comprised of global variables and code blocks: tasks, inline functions, and subroutines. NQC programs are built from blocks of code call tasks, and every program must contain a task named “main”. A task consists of a sequence of statements, and each statement ends with a semicolon (;). Statements, also called commands, are instructions for the program to do something. The NQC compiler is case sensitive, so “Rev” is different from “rev.”

“Hello, World!” is the classic first program a programmer writes when learning a new language. It prints out “Hello, World!” on a display device, and is used in many introductory tutorials for teaching a computer programming language; robot programming is no exception. For robot programming, however, this could be likened to turning a motor on and turning it off. I call my NQC robot programming version of this

program, “Hello, Robot World!” The program moves the robot forward for two seconds, then moves the robot back to its starting point. The “Hello, Robot World!” program is shown in Fig. 4.4 of section 4.2.2 below:

4.2.2 A Sample NQC Program: Hello, Robot World!

```
task main()
/*  This program moves the robot forward for four seconds,
  then moves the robot backwards for two seconds, and finally stops.
*/
{
  OnFwd(OUT_A);          //Starts motor connected to output port labeled A
  OnFwd(OUT_C);          //Starts motor connected to output port labeled C
  Wait(200);             //Robot moves forward for 2 seconds
  OnRev(OUT_A);          //Reverse direction of motor connected to output port labeled A
  OnRev(OUT_C);          //Reverse direction of motor connected to output port  labeled C
  Wait(200);             //Robot moves backwards for 2 seconds
  Off(OUT_A+OUT_C);  //Switch both motors off (Both motors are set at once using "+")
}
```

Fig. 4.4 Sample NQC Program

4.2.3 The RCX Command Center

The RCX Command Center (RcxCC) is an NQC programming tool that runs on Window PCs and makes writing NQC programs even easier. RcxCC provides a program editor and a graphical user interface to the NQC compiler. The good news is that all of these solutions use the same exact language to specify an NQC program. For example, a program written with RcxCC can also be used with the Linux version of the NQC compiler.

4.3 CodeLab

CodeLab is a web-based, automatic homework-checking tool for computer science classes (Arnow and Barshay 1998). It is designed to allow students the chance to practice specific programming concepts without having to write an entire program. Students are able to focus on the details of a specific language feature without having to concern themselves with multiple programming concepts and/or systems features such as the preprocessor. Most programming classes, including introductory ones, involve standard programming assignments that require the integration of many language features and programming concepts. In addition to its self-paced focused concept feature, CodeLab has an automatic program checking feature that is able to check entire programs submitted by students. This networked robotics project combines Multimedia, Distance Learning, Robots, Programming, and the Inter-Networking of these components into a comprehensive system to broaden the availability and effectiveness of robotics programming instruction.

4.3.1 CodeLab services

CodeLab offers services to both students and faculty. CodeLab offers students: web-access to exercises accessible from any browser such as Netscape Navigator or Microsoft Internet Explorer. CodeLab can be used from any computer that is on the Internet (without the need for plugins, Java or Javascript), provide immediate feedback, and the practice needed to master the basic elements of programming (Arnow and Barshay 1999). CodeLab offers faculty web-access to the system -- accessible from any

browser, anywhere -- automatic exercise checking, and a feature for sharing exercises with other faculty who may be teaching another section of the same course.

Most existing web-based learning technologies are inflexible: they are either overly specific or too general to be of practical use. Some are only useful if the learner has previous knowledge of another computer programming language

4.3.2 Distance learning

CodeLab also promotes the development and application of distance learning. Students who may be place-bound or time-bound can have equity of access to information and opportunities through mediated information and instruction. Moreover, research studies often point out that student attitudes about distance learning are generally positive.

4.3.3 An NQC robot programming module for CodeLab

The NQC module for CodeLab, developed in this study, provides additional details of the commonly used features, as well as some advanced features, of NQC. This NQC Robot Programming Module consists of a set of strategically designed, self-paced exercises. There are also mini-lessons that focus on specific programming skills. While CodeLab exercises are organized to facilitate the mastery of the cognitive essentials of programming (Arnow 2000), NQC CodeLab exercises are organized to facilitate the mastery of the cognitive essentials of robot programming. The main idea is to provide large numbers of self-paced, highly interactive exercises that focus on key ideas of programming. These exercises are intended to augment, rather than to replace, the traditional "whole program" assignments in the first year of undergraduate study.

4.4 The Technical Development

The System Development of this project required the following steps: 1) complete the technical phase (develop a set of client/server applications that will enable these discrete components to talk to each other), 2) complete the content development phase (develop specific robotics programming exercises suitable for both students and asynchronous software platforms and 3) integrate these into a comprehensive application. The application developed for the network robotics system reflects the organization of the ISO/OSI model, and is implemented using Java network (socket) programming.

When the basic development phase was completed, students were able to use CodeLab and remotely access the robotics lab using the following scenario:

- The student logs into CodeLab, accesses robotics section.
- The student chooses an exercise developed by the robotics instructional group. The exercise calls for writing NQC code that makes the robot carry out a particular task.
- The student writes the code, and submits it to the CodeLab system.
- The CodeLab system checks for syntactic, semantic and in some cases logical errors. If such errors are found, the CodeLab system displays them to the student.
- The student corrects the errors and resubmits the code to the CodeLab system.
- This process iterates until CodeLab has verified the code as syntactically correct, and logically plausible (to some degree).

- CodeLab then displays a message to the student indicating that the program has been submitted to the robotics lab for further processing.
- Upon receipt of an NQC program (message) from CodeLab, the controlling program in the robotics lab opens a video window so that the student sees a video segment of the robot's motion that results from the program.
- The controlling program then executes the NQC program submitted by CodeLab, and the student sees a video segment of the robot's motion that results from the program.
- The video stream stops, and leaves an image of the robot in its terminated position on the screen. A brief textual summary of the results of the student's program is then displayed. (At this point, CodeLab is able to submit the next student exercise to the robotics lab).
- After reading the textual summary of the results of their program execution, the student clicks a link to return to the CodeLab system for the next NQC programming exercise.

4.4.1 Camera/video system

Multimedia is called streaming because browsers receive data in a continuous stream from servers so that users don't have to wait to download large multimedia files. The robotics lab utilizes the AXIS 205 network camera (AXIS 2005) and the AXIS Camera Recorder (AXIS 2006) network camera server software to transmit "real-time" streaming video between the lab and client's browsers over the Internet. The AXIS 205 is designed for high quality video streaming. The AXIS 205 enables remote live video

and management of the camera from anywhere using any standard Web browser. With its own IP address and built-in Web server, the AXIS 205 doesn't require a direct connection to a PC. AXIS Camera Recorder enables remote viewing and recording functions based on schedule, alarm or on motion detection. Students have a real-time window into the robotics programming lab from any PC with an Internet connection.

4.4.2 Robot server communication system

The robot server communication system consists of the Lego RCX built-in infrared communication link (IR) (i.e., the robot), a Personal Computer (PC), the NQC robot programs, and the client/server applications programs that were created using Java. The robot server system can communicate on many levels: PC to IR tower, IR tower to RCX, java program to Windows 2000, and Robot Server to the Camera. Each of these levels of communication was used in the development of the networked robotics lab.

4.4.3 Network applications, services, and client/server applications

A telecommunications or computer *network* is the interconnection of computers and other communications equipment via a communications channel in such a way that data, programs, and peripheral devices like printers can be shared or communicated.

A *network application* is software (a program or group of programs) that allows users in a network to solve a specific problem or perform a specific task. An FTP client application program, for example, can be run on a local computer to contact the FTP server application on a remote computer. In this context, FTP is an application.

A *network service* is a functionality derived from a particular network application program. Network services refer to applications that provide connections to remote computers, transmit data, or provide conversion of data in a network. The client network

application can request services that the server network application provides. An FTP service provides the functionality of up/down loading files between client and server. In order for an FTP application to transmit data in a network, however, there must first be a connection. Switching contexts, FTP is also a service.

A *network protocol* is a "language" of rules and conventions for communication that controls the sending and receiving of information within a computer network, including the format of messages and how errors are handled. Some of the functions that network protocols perform include:

- Identifying the different devices in the communications pathway
- Establishing transmission speed and data formatting methods (including data compression) to support reliable and/or high-performance communications
- Alerting the receiving device to incoming data and defining the way the receiving device is to acknowledge receipt of the data
- Determining how errors are to be detected and corrected

Again, switching contexts, File Transfer Protocol (FTP) is a protocol for copying files. Both clients and servers use this protocol to insure that the new copy of the file is, bit for bit and byte for byte, identical to the original. FTP is therefore three things at once – application, service and protocol.

The International Organization for Standardization (ISO) specifies worldwide standards for different types of computing such as networking, database access, and character sets. The ISO's Open System Interconnect (ISO/OSI) is the model for interoperable networking, and defines network architecture and a full set of protocols.

The ISO/OSI defines a model that clearly delineates network layers and services through which applications, especially client/server applications, can communicate. This model consists of seven layers called a stack, and the protocols that sit in these layers are called a protocol stack. This is a modular approach to networking in which each layer provides services to the layer above it. The seven layers are: application, presentation, session, transport, network, data link, and physical layers. Each layer represents a different level of abstraction between the physical hardware and the information being transmitted.

4.5 The Content Development

The second phase of this project involved content development. Specific robotics programming exercises, suitable for both students and the CodeLab platform, were developed. These exercises form a sequence of increasing complexity and sophistication. They, like all CodeLab exercises, are specified clearly and unambiguously, and are amenable to some logical testing. In addition, there are exercises whose success or failure can be determined by looking at a video and comparing it with a video of a robot carrying out the correct program.

Initially, the content development effort provided a test-bed for the technical development and a context for determining the technical components required to implement the project. Most of the content development activity in the early phases of the project was of this character. As development proceeded, the system converged to stability and to its intended functionality; clusters of robotics programming exercises suitable for beginners in high schools and colleges were then developed. The key strategy in this design was to consciously develop three kinds of exercises: those that introduce and provide experience with one and only one programming idea or language

construct; those that integrate 2 or 3 such ideas and constructs; and those that constitute complete mini-projects that give the student the satisfaction of system mastery.

Although some of the asynchronous software components that are part of the networked robotics project exist (e.g., CodeLab), this asynchronous distance learning network tool was extended and new tools were design and developed (e.g., video manager and robot manager programs, and several Java network programs that govern how information moves around on the networked hardware).

4.5.1 Self-paced exercises

The interactive and dynamic self-paced exercises of the NQC robotics CodeLab enables students to reach their learning goals faster and provides students the convenience and control of deciding where, when and at what pace they learn.

The self-paced NQC robotic exercises provide the power of technology to deliver "just in time training" anywhere and anytime. The self-paced learning technology of the robotics CodeLab module, however, should not be used as an isolated tool, but as a key part of a comprehensive robotics training solution. Online robotics education, therefore, should not be considered as the only means to educate students about robotics, but also as an add-on to the overall robotics education process

4.5.2 Programming-specific exercises

These pages contain sample exercises that facilitate the mastery of the cognitive essentials of programming, mini-programming exercises that focus on specific skills, and other thought-provoking questions. These are the actual workstation programs that the students use to learn about robots, robotics and robot programming.

Sample_0 is an NQC robotics exercise that introduces the basic structure and syntax of an NQC program. So, first we examine the syntax of NQC programs. This exercise consists of a brief paragraph, followed by a CodeLab question where students are required to fill-in-the-blanks.

Sample Exercise_0: NQC SYNTAX

NQC is a text-based programming language especially designed for the RCX robot. NQC programs are built from blocks of code call **tasks**, and every program must contain a task named "main". A task consists of a sequence of **statements**, and each statement ends with a **semicolon (;).** Statements, also called commands, are instructions for the program to do something. Programs are written in an editor, then **compiled** and **downloaded** to the RCX. The NQC compiler is case sensitive, so the word "Rev" is **different** from "rev."

NQC programs are built from code blocks called _____, which consist of a sequence of statements each of which must end with a _____. A text editor is used to write NQC programs. Once written, the programs are then _____ and _____ to the RCX. NQC is case sensitive, so the name "OUT_A" is _____ from the name "out_A".

In the next three NQC programming exercises (Sample Exercises_1 thru Sample Excercise_3) we are told the RCX robot we will be working with has wheels powered by motors that are connected to output ports A and C. Sample_1 is an NQC robotic exercise that introduces and provides experience with one and only one programming idea or language construct.

- **Sample Exercise_1: SETTING THE DIRECTION (MODE) OF THE MOTORS**

We can set the direction of a motor attached to the RCX so that it will be remembered when the motor is turned on. The direction can be set to forward, reverse, or toggle by using the respective NQC statements: **Fwd(***outputs***), Rev(***outputs***),** and **Toggle(***outputs***).** The name in parenthesis, ***outputs,*** is called an argument and provides more specific information to the **Fwd**, **Rev**, and **Toggle** statements. The ***outputs*** argument uses the names **OUT_A, OUT_B,** and **OUT_C** to refer to the specific robot output ports A, B, C.

Reminder: The NQC compiler is case sensitive, so "**Fwd**" is different from "**fwd**."

Using the **Fwd(***outputs***)** statement, set motor **A** to the forward direction.

Sample Exercise_2 is an NQC robotic exercise that integrates 2 or 3 programming ideas or language constructs.

- **Sample Exercise_2: MOVING THE ROBOT**

Using the '+' operator to combine and set both motors A and C**, write <u>two</u> NQC statements as follows:** 1) **Fwd(***outputs***)** to set the wheels connected to motors A and C to the forward direction, then 2) **OnFor(***outputs, duration***)** to turn on the wheels connected to motors A and C for two seconds.

Sample Exercise_3 is an NQC robotic exercise that constitutes a complete a mini-project that gives the student the satisfaction of system mastery.

- **Sample Exercise_3: PUTTING IT ALL TOGETHER, AND VIEWING THE ROBOT**

Using the '**+'** operator to combine and set both motors A and C, **write <u>four</u> NQC statements**. The <u>first two</u> statements (**Fwd** and **OnFor**) should move the robot forward for two seconds. The <u>next two</u> statements (**Rev** and **OnFor**) should move the robot backwards for two seconds.

4.5.3 Programming and cross-compiling

Programming the RCX involves using a cross compiler: a compiler capable of creating executable code for a platform other than the one on which the compiler is run. Effectively, the NQC cross compiler separates the build environment from the target environment. An NQC cross compiler is used to check the syntax, and offer some semantic checks as well, before executing them on the robot. When the program checks out, CodeLab loads the program into the robot and lets the robot execute the program.

4.5.4 Remote testing via the Internet

CodeLab testing servers have the responsibility for taking the student code submission, incorporating it in a code harness, compiling and executing the resulting

program, and generating a response to the student. The NQC testing server that accomplishes this is itself divided into two servers: the CodeLab front-end which interacts with CodeLab and plays the role of a testing server and the robot back-end, which submits the compiled NQC program, along with various identifying information and directly manages the robot and the camera.

5. THE NETWORKED ROBOTICS LABORATORY: INTEGRATION AND IMPLEMENTATION

The previous chapter introduced the components of the robot-programming laboratory, the robot-programming language, and CodeLab: the asynchronous distance learning tool. Methods that address the synthesis of the discrete components into a comprehensive system are presented in this chapter. This includes: linking the asynchronous learning tool with the networked robotics lab, creating specific robotics programming exercises that are suitable for both students, and CodeLab. Methods for teleprogramming, cross-compiling, and remote testing are also presented along with training materials used in the programming experiments. The java client/server network (socket) programs that enable this communication are also described.

5.1 Iterative Development

Early results were obtained by testing the respective components in a "stand-alone" operation. In this instance, the individual components were utilized and tested in a piece-wise manner. For example, we were able to use the NQC cross-compiler to develop and compiled an NQC program on a Sun Solaris computer. A server was then set up a on this SUN computer, and a client program was written to contact the server. The client was able to request and display the contents of the NQC source it received from the server. This source, received from the server, was then executed on the client computer, which was an IBM-PC.

A video file was then successfully transferred, over the Internet, of a robot performing its task as defined by the program. A small set of questions were developed to

test the NQC interface. Several webcams were tested to determine which image processing features would be most useful.

5.2 Systems Coalescence

While the individual components produced encouraging results, these discrete components did not communicate with each other. What was needed was a set of procedures that would enable these discrete hardware/software components to talk to each other and be woven into a comprehensive system. Network application programs that communicated between CodeLab, the robot, and the camera, were developed. The synthesis of these components and their client/server interactions is our network robotics laboratory.

5.2.1 Asynchronous distance learning network tool

The asynchronous distance learning network tool (or, network learning tool) provides users with Internet access to the robotics lab. The network learning tool is mostly concerned with providing services to the outside world. This includes authenticating users and allowing access to the NQC CodeLab, generating NQC robot programming exercises, checking student submissions for syntax and semantic errors, and communicating these results to both the user and the robot.

5.2.2 Robotics laboratory pathway

While the network learning tool is concerned with interfacing with the outside world (i.e., clients), the robotics laboratory pathway describes the communication path between the client PCs (users of the robotics lab), and the client/server and server/client

components within the robotics laboratory. This pathway provides for the communication between the components of the robotics lab, and allows remote testing of NQC student submissions to the networked robot. Fig. 5.1 shows a graphic of the robotics communication pathway; the circled numbers 1-7 show the details of each step of the communication path from the PC client accessing the CodeLab Server to the PC receiving the robot video. The technical details of the respective steps outlined in this client/server communication are listed below.

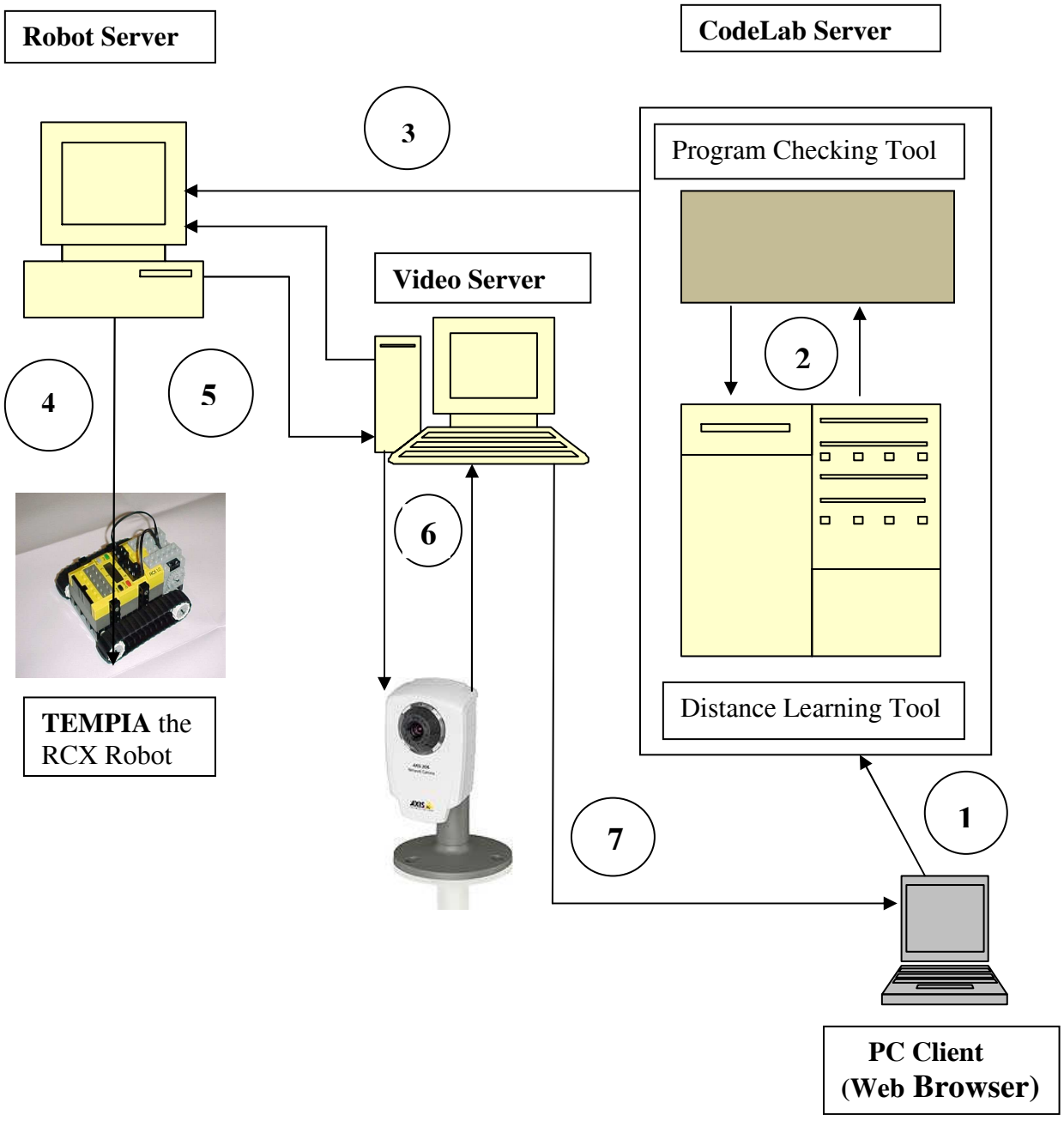

Fig. 5.1 Graphic illustrating the robotics communication pathway

1) Path from PC Client to CodeLab Server (Distance Learning Tool). The robotics lab is accessible from any browser, anywhere. It doesn't require plugins, Java or

Javascript. Initially, CodeLab acts as a server and provides Internet access to the CodeLab Robotics Module (www.turingscraft.com). After verifying user-id and password, CodeLab grants access to the system.

2) Path through CodeLab: from user Distance Learning Tool to Program Checking Tool. The NQC-based CodeLab module consists of two main components: a distance learning tool (front-end testing server), and a distance learning tool (back-end server) that loads the program into a robot and lets the robot execute the program. The distance learning tool generates an NQC robot programming exercise for the student to solve. The student submits a solution which is encapsulated into an NQC harness and passed to the NQC program checking tool. This part of the front-end testing server uses an NQC cross compiler to check the syntax, and offers some semantic checks as well, before executing them on the robot. If errors are found, messages are displayed, hints are given to the user, and the user resubmits the exercise. This process repeats until the errors have been corrected. (program: mySend.java)

3) Path from Program Checking Tool (CodeLab) to the Robot Server. When the program submission checks out, the distance learning tool, taking on the role of a client, contacts the distance learning tool, and submits the compiled NQC program, along with various identifying information. (program: myRecieve.java)

4) Path between the Robot Server and the Robot. The robot (back-end) server loads the program into the robot, and lets the robot execute the program. (program: NQC Compiler)

5) Path from Robot Server to the Camera. The robot server opens a video window and activates a camera so that the user sees a video segment of the robot's motion that results from the program. (program: myRecieve.java)

6) Path from Camera to the Video Server. The Video Server (also a back-end server) records the action of the robot as captured by the network camera, saves the short video file, and returns the status and video access information to the front-end server.

7) Path from Video Server to PC Client. This video status and access information is then passed back to the main (front-end) CodeLab server which makes the video available to the student. (program: mySend.java)

5.2.3 Java network programming for systems coalescence

At the heart of the systems coalescence is the communications between the CodeLab (back-end) server, and the robot server. The CodeLab server is very portable and runs on various versions of Unix. The robot server resides on a Dell PowerEdge 2600 Server running Windows 2000 Server. The CodeLab server takes on the role of a client when it communicates with the robot server. These client/server access objects at the end-point of a TCP/IP session are called sockets. Communication between these sockets is established by a pair of Java network (socket) programs (mySend.java and myReceive.java). The NQC client (running on CodeLab), mySend.java, initiates a connection to the server (robot), myReceive.java, and then passes or request information from the robot server as follows:

1) **myReceive.java** listens and tries to receive a connection from an NQC client

2) **mySend.java** tries to open a connection to the robot server
3) once a connection is open, **mySend.java** sends an NQC robot program
4) **myReceive.java** attempts to run to NQC robot program, and check the status
5) **myReceive.java** sends the robot video, or the status code and error messages, back to NQC client
6) **mySend.java** inputs the result from the robot server, then displays the robot or the status and error messages

The complete programs (mySend.java and myReceive.java) appear in Appendix D: Java Network (Socket) Programs.

6. EXPERIMENTS AND RESULTS

This chapter presents the results of several robotics lab learning experiments of the system developed in Chapter 5.

6.1 Contextualization

The Robotics Laboratory, housed in the Major R. Owens NASA Aerospace Educational Laboratory (AEL) at Medgar Evers College of The City University of New York, is the primary site used to teach the robotics classes. Several groups of students (from elementary school through college) have experienced the Robotics Lab, and have completed sample lessons that introduce robot and robotics fundamentals. The Robotics Lab has been used to teach several robot programming languages (including: RCX Code, RoboLab, NQC, and XSLisp). A variety of robot languages have been presented, over recent years, to a diverse set of populations at the lab.

The major focus of this research was to design, development, and implement of a Networked Robotics Laboratory that will allow visitors to learn a robot programming language, and use that language to control a robot via the Internet. The research also included a study of how the combination of length of instruction, method of instruction, and learning expectancy impacted students' comprehension of robotic design, programming, and STEM-related career interests. Student success was measured through three different forms of assessment: a pretest and posttest questionnaire, an oral presentation to parents (for elementary students), and a demonstration of their robot programming skills.

The pretest questionnaire (for the elementary and middle school groups) was distributed to the students during the second week of the program. The pre- and post tests for the elementary school group were designed as a type of guided writing and drawing activity to assess the students' career interest. The objective of the middle-school questionnaire was to collect measures such as background information, programming and robotic experience, and the subject's current experience with LEGOS. A posttest was distributed to the students at the end of the program. The measures collected were the same as the pretest measurements to evaluate the differences before and after the study.

For each experimental group (elementary-, middle-, high school, and college), the robotics training sessions began by attempting to create "A Culture of Expectancy" in which the students held high expectations for the robotics program (faculty, facilities, and equipment), for themselves, and for their success in the program. Access alone to technology (computers, Internet, and robots) as a means to educational, social, and other opportunities is not enough. Technology-based programs must also include structural components to support students' self-evaluation of their ability and beliefs about their probability of success in the robotics program and STEM-related careers. Without these structural components of success, it may be difficult to put together the whole. Students learn best when their attitudes, and perceptions about the subject are positive, and they expect to learn … *when they are motivated!*

The robotics program assessment involved a measure of students' knowledge, skills, and achievement (subjective) task value — beliefs about value of attending the robotics program. For each experimental group (elementary, middle-, high school and

college students), the goal was for students to successfully write several programs for their robots.

6.2 Experiment 1: Elementary School Students

The elementary school students (6-graders) were part of "RoboCamp 2005**":** an Intensive week-long (Monday thru Friday, 9 AM – 1 PM) LEGO Robotics Summer Camp for rising six-graders. The Camp was a collaborative partnership between LEGO Education, the Education Technology Think Tank*,* Medgar Evers College, and the Crown School for Law and Journalism (PS 161). On the first day of this week-long course, students watched videos of simple robots and built a simple rover. Throughout the remainder of the course, they were introduced to the RoboLab graphical programming language, gained familiarity with the RoboLab programming environment, uploaded several programs to the robot, and gained a basic understanding of sequential program execution. RoboLab is a 3-tiered robot programming language tool where novices can use more sophisticated versions of the language as their expertise develops. The elementary school students were introduced to Pilot Style Programming. Pilot Programming is the lowest tier in RoboLab.

6.3 Experiment 2: Middle School Students

The middle school students were selected from the College's NASA SEMAA Program (Science, Engineering, Mathematics, and Aerospace Academy (SEMAA), and took part in an 8-session program. This group of middle school students met 3 hours each week for a total of eight weeks. These students were introduced to the basic control structures of sequence, selection, and iteration to better control their robots. Middle school students studied Inventor Style Programming. Inventor Style Programming is the

second tier in RoboLab, and is more challenging than Pilot Programming. Inventor provides a more open-ended environment for constructing a program and is much more elaborate than Pilot. Inventor is very similar to Pilot, but has many more options and toolbars than Pilot.

6.4 Experiment 3: High School Students

High school students participated in a 10-session program, and were part of the New York State Science & Technology Entry Program (STEP) at Medgar Evers College. This group of high school students met 3 hours each week for a total of ten weeks. In addition to developing technical problem solving and critical thinking skills, STEP students learn and practice enrichment activities that include: student development, leadership opportunities, career exploration, college preparation, internships, and parent workshops. These skills and activities experienced by STEP Students are important for personal and social growth, and supported our efforts to create "A Culture of Expectancy" for this group of students.

The high school students had the most time on task and were also able to construct several robots, and experiment with several robot programming languages. This allowed them to get a firm understanding of the working dynamics of different types of robots and sensor types. Several of these high school students were already familiar with text-based programming since they were preparing for the Advance Placement (AP) test in Java programming. The AP provides high students the opportunity to earn credit or advanced standing at most of the nation's colleges and universities.

The High School students studied RCX Code, RoboLab (including the highest tier: Investigator Programming), and NQC. Investigator is a program which uses Pilot

and Inventor programming to incorporate data collection into projects. Data such as time and light can be measured and recorded. RCX Code is a graphical environment for programming robots and is included in the RIS. Using RCX Code, students can create robot programs by snapping together functional blocks, just like snapping together Lego bricks. RCX Code is great for people who have never programmed, but it's limiting for experienced programmers.

6.5 Experiment 4: College Students

The college students were MEC Computer Science Majors. One group of students used the NQC robot programming language in a Digital Systems course and another group used NQC in an Artificial Intelligence course. Although each course met 4 hours per week for a total of 15 weeks, only 1 week of NQC Programming was introduced in each course. Both classes were required to give a course-specific presentation. In the Artificial Intelligence course, where students studied Common Lisp, two students volunteered to search the Internet to find a Lisp Programming environment for the Lego Mindstorms programming environment as part of their class presentation. The two students were able to find XSLisp, a Lisp system developed by Taiichi Yuasa which is designed to control RCX blocks of the Lego MindStorms Robotics Invention System. Students in the Digital Systems course only programmed the RCX using NQC.

Grade Level	**Programming Language (s)**	**Length of Program**	**Methods of Instruction**
Elementary School	RoboLab (Pilot Style)	1 week (5 days, 9 AM - 1 PM)	Constructed a Rover, Slide Shows, Videos, Pre/Post Test
Middle School	RoboLab (Pilot and, (Inventor Style)	8-sessions (3 hours each weekly session)	Constructed several Rovers with touch and light sensors, Slide Shows, Videos
High School	RoboLab (Pilot, Inventor, and Investigator Style)	10-sessions (3 hours each weekly session)	Constructed several Rovers with touch, light, and rotation, sensors, Slide Shows, Videos
College	NQC and XSLisp	1 week of NQC	PowerPoint Lecture Internet Search

Fig. 6.1 Experimental Groups

6.6 Results

The elementary school students were introduced to the Pilot Style Programming of RoboLab. During the week, pairs of students were able to build a MiniRover Robot, and complete all of the RoboLab Pilot Training Missions, Through the use of slide shows, videos, and multimedia, students were able to successfully write several programs for their MiniRovers. The LEGO Education's *Camp-on-a-Disk: PROBE* software was used during "RoboCamp 2005" (Lego_Education 2005). This CD-ROM contained many of the

multimedia-based animated slide shows, videos, and robot building instructions used by the elementary school students.

For the elementary school students, pre- and post tests were given to measure possible changes in their attitudes and career interests as a result of attending the robotics workshop. The pre- and post tests for this group were designed as a type of guided writing and drawing activity (see Appendix E: Pre/Post Test: Elementary School Group). The pre-test asked the students to think about the kind of career they would like to have when they grew up and finished school, then draw a picture depicting themselves working in that career, and write a paragraph about their career choice. The group showed a significant shift of attitude and career choice as indicated with the post-test. While the pre-test showed most of the students held career interest that reflected their celebrity heroes: athletes, entertainers, or the better known professions (teachers and nurses), the post-test reveled a shift in attitude.

The post-test asked the question, "How has your experience with the RoboCamp impacted you and your career interest?" After being exposed to robots and robot programming, some wanted to be computer programmers, and video game designers. There were two students who initially aspired to become singers. One of the aspiring singers changed her mind and now wants to work on computers. Although the other still wants a singing career, she said, the RoboCamp had taught her, "When you pay attention you learn more, and if you make a mistake don't get mad look over what you did and fix it." This comment related to her experience of building a robot rover, then iteratively coding and debugging the robot programs.

Middle-school students were required to construct several robots. This allowed them to get an understanding of the working dynamics of their robots. This group was given an NQC/Robotics Pretest/Post test (see Appendix E: Pre/Post Test: Middle School Group). The pretest and posttest on fundamental concepts showed an increase in knowledge of robot programming. In addition, evaluation of answers to essay questions demonstrated an increased interest in STEM-related careers.

While this group was able to make the transition from the iconic-based RoboLab to the text-based NQC and write several programs, they all preferred programming using RoboLab. The students said the robot programs were easier to understand when they were written using graphical icons as commands, than they were when the equivalent programs were written using text-based commands.

The high school students group was the most creative, and they were better able to make the transition from RCX Code and RoboLab, to the text-based NQC and write several programs in each of the robot programming languages.

The college students were MEC computer science majors enrolled in an Artificial Intelligence course. These students did not build a robot, but they did study the NQC robot programming language for a week and wrote NQC programs to demonstrate the use of touch and light sensors. The college students were required to choose an AI-related programming project for their end-of-semester assignment. They were allowed to select the XSLisp robot language to program the RCX for their end-of-semester projects, provided they learned XSLisp on their own. Since Common Lisp was briefly covered in the AI course, the students simply had to look at the features of the XSLisp API that related to the robot motors and sensors. Two female students took up the challenge and

selected the XSLisp project. The students gave a PowerPoint presentation of this Lisp/Scheme-based language that demonstrated the extended functionality for interfacing with motors, sensors and other RIS devices. The students were then able to provide a live demonstration of several XSLisp programs that controlled the robot, and they were able to answer questions posed by their classmates. The students gave an excellent presentation, and they learned a lot. The AI students were the most excited about the idea of being able to see the physical manifestations of their abstract Common Lisp programming syntax and semantics coming to life in the robot through the use of XSLisp.

Grade Level	**Robot Design (construction)**	**Programming Skills Demonstrated**	**Increase in STEM-related Career Interest Shift?**
Elementary School	Simple Rover	Pilot Style Sequence & Selection	Yes
Middle School	Simple Rover TankBot	Pilot, Inventor, and NQC. Sequence, Selection and Iteration	Yes
High School	Simple Rover TankBot Gear Ratios	Pilot, Inventor, and Investigator , and NQC Sequence & Selection Iteration	Yes
College	TankBot	NQC and XSLisp	N/A (Computer Science Majors)

Fig. 6.2 Experimental Group Results

6.7 Conclusions and Recommendations for Further Study

The network robotics laboratory allows extended access to the Robotics Workstation in the NASA AEL at Medgar Evers College. Since the current AEL curriculum only uses the built-in RCX Code for robot programming, the Internet access to the network robotics lab provides for an expansion of the both programming language used (NQC) and the type of programming exercises assigned. The inclusion of the online NQC CodeLab network into the stand-alone robotics lab at Medgar Evers College has been found to be extremely valuable for improving the students' understanding of the subject material and its practical applicability, and for increasing their motivation. Students were engaged by constructing the physical robot, and the novelty of learning how to program both the robot and a similar online robot located in the networked robotics lab by accessing NQC CodeLab. It has also been observed that the success of these robotics programs could in part be a result of the high level of motivation and enthusiasm of the students.

Plans are to continue repeating the experiment of including NQC CodeLab into robotics courses for grades K thru college, and do a more systematic analysis of the results once additional data has been collected. In addition to the pre and post test of elementary school students, the evaluation uses a statistical analysis of student solutions to the NQC CodeLab problems, and Likert techniques to measure pre and post attitudes of middle-, high school, and college students.

Future work will involve an upgrade of all the components in the robotics lab: network cameras and software, PCs, new robots (including the new Lego NXT robot), and the development of other CodeLab modules for different robot programming languages, such as the Java-based Lejos.

7. POTENTIAL CONTRIBUTIONS TO KNOWLEDGE IN THE FIELD

This last chapter suggests potential contributions of this work to knowledge in the field. These suggestions bring together novel ideas and techniques that can be applied to other robotic programming languages. The chapter concludes with a set of methods and metrics that will be used to evaluate the performance of these ideas and techniques.

7.1 Novel Ideas

This dissertation has demonstrated how Inter-Networking Multimedia, Distance Learning, Robots, and Computer Programming can be combined into a comprehensive system that broadens the availability and effectiveness, of robot programming instruction. The novel aspect of this work is that students are able to learn a robot programming language, write and test their programs via the Internet on a remote robot located in the networked robotics laboratory.

The major contribution of this study will be the development, construction, and evolution of tools and techniques that can be used to automate the practice aspects of robot programming without abandoning the basic and traditional design processes. The tools and techniques developed from this work will address the learning continuum of robot programming from the self-paced focused programming exercises to complete programs that involve writing several source files. Given the appeal of robotics to adults and children of both sexes, as evidenced in these experiments, robot programming is both motivational and beneficial to those learning STEM-related concepts. CodeLab's capabilities will allow the NQC robotics module to be scaled up to deliver thousands of exercises to hundreds of classes. Libraries of exercises will also be developed so that

instructors around the world can immediately make this system available to students in their classes (Arnow 1999).

My belief is the Networked Robotics Laboratory can become a local and national resource for CUNY by providing instruction and practical experience in robotics, programming, and networking students virtually anywhere, virtually anytime. In fact, NASA has developed a new Microgravity curriculum, as part of its Aerospace Education Laboratories (AELs) that includes a Robotics Workstation. (Appendix C: THE AEROSPACE EDUCATIONAL LABORATORY). The technical issues resolved by studying the problem of accessing an online robot lab via the Internet, can be transferred to other laboratory contexts like the AELs. By extending access to these Robotics Workstations, this thesis project will be making a decisive contribution to the advancement of robotics education and NASA's effort to provide all students an opportunity to share in its resources.

7.2 Performance and Evaluation

This study consists of several experiments to test the effect of "educational interventions". In particular, this study tested the effectiveness of "educational methods" using the NQC Robot Programming Module for CodeLab.

Students who participated in the study benefited by learning a programming language and were able to evaluate an important teaching tool. Furthermore, the networked robotics laboratory will be making a contribution to the advancement of Computer Science education and its effort to provide all students an opportunity to share in its resources. Students interested in participating in future robot programming research

studies will be required to complete the Robotics-CodeLab Interest Form that appears in Fig. 7.1 below.

Robotics-CodeLab Interest Form

NAME: _____

SITE: _____

DEPT: _____

EMAIL: _____

Approx # Robotics Sections: _____

Approx # Students Per Section: _____

NOTE: Filling out this form is not a commitment; it is an expression of potential interest in the study.

Fig 7.1 Robotics-CodeLab Interest Form

INFORMED CONSENT FORM

This study is about the robotics workstation in NASA's Major R. Owens Aerospace Education Laboratory at Medgar Evers College/CUNY. Your child has been asked to participate as a student in our evaluation of the robot programming module of the CodeLab software. This evaluation is being conducted by William C. Harris (wharris@acm.org) as part of his dissertation research under Dr. David Arnow (arnow@acm.org). Any questions about the evaluation may be addressed to Mr. Harris or Dr. Arnow. This evaluation consists of several sessions – a tutorial session and testing sessions – which may be conducted on separate days. These experiments will be completed over the next three to ten weeks.

Students will be asked to perform various tasks with this system. These tasks involve the exploration of the robot programming module of the CodeLab software environment. We are evaluating the system to make it as usable as possible but we are in no way evaluating the students. The purpose of this study is to test the robot programming module of the CodeLab software. There is no way for students to fail. We will unobtrusively gather statistics related to task completion times to evaluate the merits of the robotics programming exercises. Sessions will last approximately 1 hour and involve less than minimum risk.

Students will be asked to complete questionnaires at the start and end of each session. Short interviews will also be conducted after students fill out the questionnaires to better understand their answers. During the session students will be asked to "speak their thoughts aloud", so their impressions of the system can be better understood.

Students are volunteers. They will not be paid for these sessions. If you allow your child to take part in this study, your child will benefit by learning a robot programming language, and will be able to evaluate an important teaching tool. We will be using a subject pool of twenty-five to fifty people.

As a participant you have certain rights, which are listed below:

1. Your child may withdraw from the session at any time for any reason. Taking part in this study is completely voluntary. If your child does not take part, he/she will have no penalty.
2. At the conclusion of the session you may see your child's data. If you decide to withdraw your child's data, please inform the evaluator immediately.
3. You and your child are requested not to discuss this session with other people who might be in the group from which other participants are drawn during the next few months.

If you have any **questions regarding this research**, you can call or write:

Mr. William C. Harris
Medgar Evers College/CUNY
PECS Department
1150 Carroll Street – Room C-413
Brooklyn, New York 11225
(718) 270-6453

Dr. David Arnow
Brooklyn College/CUNY
CIS Department - 2109 Ingersoll Hall
2900 Bedford Avenue
Brooklyn, New York 11210
(718) 951-5657

If you have any questions about **your rights as a research volunteer**, call Professor Balk, IRB Chair at Brooklyn College/CUNY at (718) 951-5000 ext. 1232.

Thank you. Your child's time and effort while participating in this study are greatly appreciated. Your signature below indicates that you have read this form in its entirety and that you consent to the participation of your child. Additionally, we will ask for your child's verbal agreement (assent) to take part in this study.

Fig 7.2 Informed Consent Form

Parental Informed Consent Form

I, (_____name of parent/guardian_____), have read and understood the attached letter describing a study to be conducted under the auspices of Medgar Evers College/CUNY, NASA Aerospace Education Laboratory, to be conducted by (Mr. William C. Harris, and Dr. David Arnow).

I hereby consent to the participation of my child, (_____name of child_____), in this study. Additionally, we will ask for your child's verbal agreement (assent) to take part in this study.

Informed Assent Statement Read to Child

> I have already asked your parents if it is ok for me to ask you to take part in this study. Even though your parents said I could ask you, you still get to decide if you want to be in this research study. No one will be upset if you do not want to participate, or if you change your mind later and want to stop. You can ask questions now or whenever you wish. If you want to, you may email me at: wharris@acm.org.

I understand that I am free to discontinue such minor's participation at any time without suffering any disadvantage.

I also understand that the findings of the study will be interpreted and recorded only on a group basis with no identification of specific individuals. All information about individuals will be held in confidence, to the extent permitted by law.

If I have any questions or wish further information about the study, I understand that I may call Mr. William C. Harris at (718) 270-6173. I further understand that if I have additional questions that may not be answered by the researcher, I may call his/her advisor, Dr. David Arnow at 718- 951-5657, or IRB Chair, Professor Balk, at Brooklyn College/CUNY at (718) 951-5000 ext. 1232.

In addition to consent for my child's participation, my signature confirms that I have received a copy of this consent form together with any attachments which describe the research to be conducted.

Parent's/Guardian's Signature: _____Date: _____

Parent's/Guardian's Name: _____

Investigator's Signature: _____ Date: _____

Fig 7.3 Parental Informed Consent Form

All research that involves human participants (including interviews, oral history, and database research) must be reviewed and approved by The Graduate Center's Committee on the Protection of Human Subjects or another CUNY campus Institutional Review Board (IRB). These experiments were approved for this project by the Brooklyn College IRB (research protocol: 08-016), contingent on participants reading and signing the informed consent forms shown in Fig. 7.2 and Fig. 7.3.

Summary of efforts and goals for the Robotics-CodeLab Programming Project

Goal 1: To **design, development, and implement of a "Networked Robotics Laboratory"** that will allow visitors to learn a robot programming language, and use that language to control a robot via the Internet.

Goal 2: **To support the existing stand-alone robotics laboratory computer/robot programming-related activities.** One of the big problems with stand-alone robotics laboratories is the limited access to a practice environment; physical access is limited to the lab operational hours. When students begin learning to program robots, they require lots of debugging time. There are several types of errors that can occur when programming robots. In addition to the syntax and semantic errors of robot programming (as with computer programming), errors can be caused by the physical robot itself. The networked robotics lab can provide increased practice time for debugging.

Goal 3: **To strengthen students' basic programming skills using a variety of robot programming languages including NQC:**

Students will learn about the fundamentals of robots, robotics, and the NQC robot programming language. NQC CodeLab has several short exercises, each focused on a particular programming idea or language construct. By completing the NQC CodeLab exercises, students will better internalize and reinforce their knowledge of the programming concepts.

Goal 4: **To give students the confidence and desire to pursue SMET careers.**

Career choice instruments will be sent to students participating in the Robotics Program at the beginning and end points for the program. A comparison of pre and post career aspirations of participating students, along with a baseline comparison of students who are not participating in the program will be made. Unlike grades and attendance, schools do not routinely measure students' career aspirations. A more national database may have to used -- the National Educational Longitudinal Study (NELS: 88), compiled by the National Center for Educational Statistics (NCES) -- for the baseline comparisons.

It is important that students have positive attitudes and perceptions of robotics, and STEM-related subjects, to increase their chances of success in these programs.

APPENDIX A Essentials of Robotics

A.1 Robots and robotics

The idea of robots goes back to ancient times with legends of mechanical elephants, witches and magicians causing inanimate objects to come alive. Over the last few centuries, only a small number of successful mechanical robots have been made. In the eighteenth century, several elaborate automatons were made, including an automaton model of a duck. It was composed of over 4,000 parts, with a working digestive system and walked, quacked, ate, drank, eliminated and responded to touch by turning its head towards you and protest (Fuller 1999). It wasn't until the age of electronics and the digital computer, however, that major advances in robotics were made.

Today, many people think of robots in terms of the anthropomorphic (human-like) images of the R2-D2 or C-3PO-like devices from the *Star Wars* movies, or the advanced prototype robot child, David, from Steven Spielberg's movie, *AI,* or the robots seen in the movie, *I, Robot*, featuring actor Will Smith as Detective Del Spooner. However, most of today's robots don't look anything like these walking, talking humanoid devices at all. Consider a mail carrier robot that is used to deliver and pickup mail; it looks like a cart, not a postal worker. Moreover, robot surgeons that are used to assist doctors in the operating room look like machine tools, not doctors. Few people think of vehicles or manufacturing devices as robots, and yet robots are predominately used in these areas.

Karel Capek, Czechoslovakian dramatist, first used the word robot; in his January 21, 1921 play RUR (Rossum's Universal Robots.) The word is derived from the Czech word "robota" which is loosely translated as menial laborer. In the play, Capek's robots were designed to be perfect and tireless workers who performed manual labor for human

beings. Today, robots are designed to assist human workers, not replace them by doing jobs that are too dirty, dull, or dangerous – *the 3 D's*.

The Robot Institute of America defines a *robot* as a re-programmable, multifunctional manipulator designed to move material, parts, tools, or specialized devices through various programmed motions for the performance of a variety of tasks. In simplest terms, a robot is a computer whose primary purpose is to produce motion (Fuller 1999).

In his 1942 science fiction short story, "Runaround", Isaac Asimov coined the term *robotics* as the study or science of robots. Robotics, therefore, refers to the collective technology that deals with designing, programming, testing, building and controlling robot movements and functions. Robotics includes diverse areas of technology as engineering (electrical and mechanical), computer science (hardware and software), and electronics.

A.2 The Anatomy of a Robot

This section describes an industrial manipulation robotic system and compares it to that of a human; this analogy is easily extended to mobile robots. An industrial (manipulator type) robot consists of the following components: controller, end-effector, arm, actuator, and sensor. To better understand the parts of a robot, it is instructive to associate the physical parts of a robot with the physical parts of a human being. Every robot is connected to a computer identified as the *controller*. The controller functions as the "brain" of the robot and is used to enable the robot to carry out built-in instructions or teach the robot how to perform new tasks.

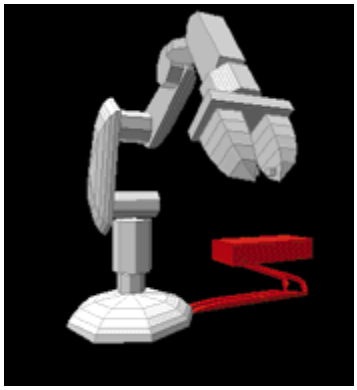

Fig A.1 Controller

The *end-effector* (gripper, or fingers) is the "hand" connected to the robot's arm. It is often different from a human hand - it could be a tool such as a gripper, a vacuum pump, tweezers, scalpel, blowtorch - just about anything that helps it do its job. Some robots can change end-effectors, and be reprogrammed for a different set of tasks. If the robot has more than one arm, there can be more than one end-effector on the same robot, each suited for a specific task.

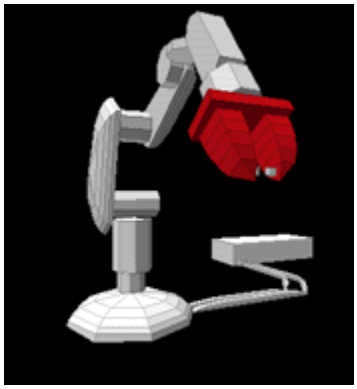

Fig A.2 End-effector

The *arm* is the part of the robot that is used to position the end-effector within reach of a part or workplace. Many robot arms resemble human arms, and have shoulders, elbows, wrists, and hands (end-effectors) where it can pivot. Joints, or axes of motion, in the arm allow the robot to position itself in its environment by bending, sliding or rotating.

The type of joints used and motion of the arm determine the work area, or cell, of the robot. Each joint is said to give the robot 1 degree of freedom of motion. The human arm has six degrees of freedom.

Fig A.3 Arm

The *actuator* (also called a drive, or power supply) is the "engine" that drives the links (the sections between the joints) into their desired position. The actuator supplies the power source needed to run the controller, actuators, and sensors thus, enabling the robot to move. The power supply for the human is the digestive system. Without an actuator, a robot would just sit there, which is not often helpful. Most actuators are electric, hydraulic, pneumatic, or vacuum.

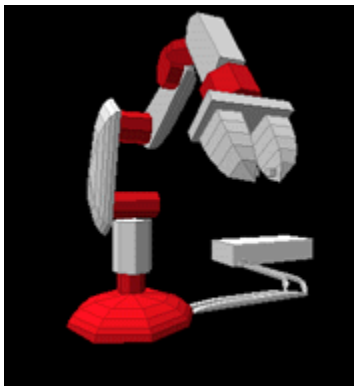

Fig A.4 Actuator

Just as humans use their sense of sight, sound, touch, taste, and smell to gather information about and adapt to our world, robots use *sensors* to obtain limited feedback and adapt to their environments. A sensor provides the robot controller with information about its surroundings and lets it know the exact position of its manipulator and the state of the world. Robots sensors use *transducers* that convert one form of energy to another form of energy. For example, a robot that uses a camera as part of its vision sensor, would sense information in the form of light patterns and convert these patterns into electronic signals. These electrical signals would then be transmitted to the robot controller. Robot sensors can be designed and programmed to extend human ability. The above robot vision sensor, for example, might be able to "see" in the dark, detect tiny amounts of invisible radiation or measure movement that is too small or fast for the human eye.

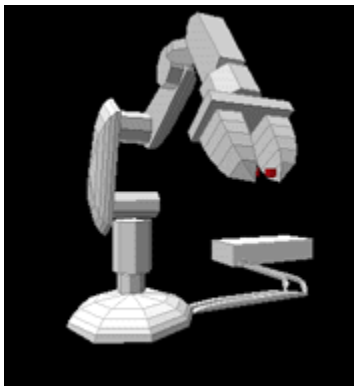

Fig A.5 Sensor

In the previous paragraphs, the parts of a robot were associated with the physical parts of the human body, and were said to consist of the following components: controller, arm, actuator, end-effector, and sensor. Some robot applications, however, may not require robots to have arms or fingers at all. While in other applications, robot components may be housed separately as discrete units, or they can be housed together as a single unit. The jointed-arm (arm, shoulder, elbow, wrist, hand with fingers), for example, can be thought of as a *manipulator*: the part of the robot that physically performs

the task. A robot's *body* can be thought of as simply the physical chassis that holds the other pieces of the robot together. The industrial robot may be stationary or sit on a vehicle. To carry the analogy further, the *vehicle*, for an industrial robot, may consist of wheels that moves it along a fixed rail, free wheels, treads, or some type of legs and feet; the vehicle for the human is the legs. It is the vehicle feature that extends stationary industrial robots, into the realm of mobile robots that move their bodies around from place to place.

APPENDIX B The Engineering Approach To Robotics

The engineering approach to robotics is primarily concerned with the design of robots for automating industrial and manufacturing processes. Industrial engineers are interested in designing industrial robots to automate the workplace. Robot arms (industrial manipulators) and mobile carts (automated guided vehicles (AGVs)) are examples of industrial robots. To get an industrial robot to perform its specific and repetitive tasks, it must be programmed.

Industrial robots receive little or no sensory input and can only execute some standard sequence of actions in accordance with their program. These robots are unable to adjust to changing conditions of their environments. Without sensory input, industrial robots are unable to detect changes in their environment. Consider a robot arm. If a part is not where it belongs when the robot tries to pick it up, the robot arm simply moves air. An AGV, on the other hand, will collide with an obstruction that might randomly appear in its path. If the AGV has limited sensory capabilities, it would simply stop and wait indefinitely until its path is cleared. Clearly, these industrial robots do not appear to be very intelligent.

The following definition of the present-day industrial robot describes its intelligence level: a robot is a one-armed, blind idiot with limited memory, which cannot speak, see, or hear (James 1985). Since industrial robots operate in an environment that is fairly fixed and predictable, these relatively dumb robots are generally sufficient for industrial applications that require repeatability, precision and speed.

APPENDIX C The Aerospace Educational Laboratory

The Aerospace Education Laboratory (AEL) is a state-of-the-art classroom that provides new technologies to partnership cities to excite students about science and math. In ten unique workstations students explore technology through “hands on/minds on” activities that model real-world challenges in aviation. The Aeronautics Scenario is based upon activities students need for the successful completion of their mission: a cross-country flight from Washington, DC to San Francisco, California. Along the way they are to visit the four NASA Research Centers and learn about these centers' contributions to aeronautics.

The new curriculum for NASA Glenn Research Center's Aerospace Education Laboratories (AELs) includes a Microgravity Scenario. This scenario is based upon activities students need to complete to successfully accomplish their mission: a "flight" via space shuttle from Kennedy Space Center to the International Space Station (ISS). Along the way they will learn about the ISS, microgravity, how human bodies react to the space environment, and different types of microgravity research activities. Some of the most sophisticated and elaborate robots are being built to assist in the construction of the ISS. At the robotics workstation, a component of the new curricular for the microgravity scenario, students will learn about the fundamentals of robots and robotics, and how robots are used in space. Students will also learn about robot programming languages through the robot-programming module developed as part of this research.

APPENDIX D Java Network (Socket) Programs

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

public class MySend3 {
    public static void main(String[] args) throws IOException {

    //  Create Input/Output Streams: br, out, and LCV c;
        File inputFile = new File("Fwd3Bak3.nqc");
        BufferedReader br = new BufferedReader(new
InputStreamReader(new FileInputStream(inputFile)));
        PrintStream out = null;
        int c;

    //  Initialize socket and try to open connection to robot
server
        Socket robotSocket = null;

          try {
          int port = 9999;
          if (args.length>0)
            port = Integer.parseInt(args[0]);
            robotSocket = new Socket("146.245.249.197", port);
            out = new PrintStream(robotSocket.getOutputStream(),
true);


          } catch (UnknownHostException e) {
            System.err.println("Don't know about host:
146.245.249.197.");
            System.exit(1);
          } catch (IOException e) {
            System.err.println("Couldn't get O/P for "
                               + "the connection to:
146.245.249.197.");
            System.exit(1);
          }

          BufferedReader brRobotStatus = new BufferedReader(
                                    new InputStreamReader(

      robotSocket.getInputStream()));
        String inline = br.readLine();
        while (inline!=null) {
                    out.println(inline);
                  inline = br.readLine();
        }

        out.println("ENOUGH!");


          //Get I/P from robot server, then display robot status
code and error messages
```

```
            String robotReply = brRobotStatus.readLine();
            System.out.println("Reply from robot server: " +
robotReply);
            robotReply = brRobotStatus.readLine();
            System.out.println("Reply from robot server: " +
robotReply);
     out.close();
     br.close();


        robotSocket.close();
    }
} // mySend3
```

// mySend.java program (above)

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

// MyRecv5 runs the robot, checks the status, and returns the
robot video file to MySend

public class MyRecv5 {
    public static void main(String[] args) throws IOException {
    //  Initialize serverSocket and try to receive connection
from NQC client
        ServerSocket serverSock = null;
        int port = 9999;
        if (args.length>0)
            port = Integer.parseInt(args[0]);

        try {
            serverSock = new ServerSocket(port);
          } catch (BindException be) {
               System.err.println("Bind Failure " +
be.toString());
               System.exit(1);
          } catch (IOException ioe) {
               System.err.println("Server Socket Creation Failure
" + ioe.toString());
               System.exit(1);
          }

        while (true) {
           System.out.println("Robot Server waiting for NQC
client.");
           Socket robotSocket = serverSock.accept();
           System.out.println("\n*******\nConnection
ACCEPTED\n");
           receiveFromClient(robotSocket);
        } //while
    } // main
```

```
//
*****************************************************************
****
        private static void receiveFromClient(Socket robotSocket)
{
              String line;
              BufferedReader br = null;
              PrintStream outRobotStatus = null;
              PrintStream ps = null;
            try {
             br = new BufferedReader(
                              new InputStreamReader(

      robotSocket.getInputStream()));
             outRobotStatus = new
PrintStream(robotSocket.getOutputStream() );

             File outputFile = new File("outMyRecv.nqc");
             ps = new PrintStream(new
FileOutputStream(outputFile));

             line = br.readLine();
             while (line!=null && !line.equals("ENOUGH!")) {
                   System.err.println("RCV: " + line);
                   ps.println(line);
                   line = br.readLine();
             }
             System.out.println("Input Now Complete");
System.out.flush();

          ps.close(); // must first close this O/P file to run
the NQC program


               // Run the robot, and check status of the return
code
             runRobotCheckStatus(outRobotStatus);

          } catch (IOException ioe) {
              System.err.println("I/O
Failure");System.err.println(ioe.toString());
          }
          if (ps!=null) try { ps.close(); } catch (Exception e)
{}
          if (outRobotStatus!=null) try { outRobotStatus.close();
} catch (Exception e) {}
          if (br!=null) try { br.close(); } catch (Exception e)
{}
          if (robotSocket!=null) try { robotSocket.close(); }
catch (Exception e) {}
       } // receivedFromClient

     private static void runRobotCheckStatus(PrintStream
outRobotStatus ) {

       try {
       String line;
```

```
      /* Execute the command using the Runtime object and get the
       Process which controls this command */

       Process p =
Runtime.getRuntime().exec("myNqcBatchFile.bat"); // Executes the
NQC O/P file

       p.waitFor();

       BufferedReader input =
         new BufferedReader
          (new InputStreamReader(p.getInputStream()));

       BufferedReader stdError =
         new BufferedReader
          (new InputStreamReader(p.getErrorStream()));

       //Read the output from the command
       while ((line = input.readLine()) != null) {
         System.out.println(line);
       }

       //Read any errors from the attempted command
       while ((line = stdError.readLine()) != null) {
         System.out.println(line);
       }

       input.close();

       // Send results back to client (both status code and error
messages)
        if (p.exitValue() == 0) {
         outRobotStatus.println(p.exitValue());
           outRobotStatus.flush();

         sendEncodedVideo(filename); // Invokes method to send
Video file to MySend.java

      }
      else {
         outRobotStatus.println(p.exitValue());
           outRobotStatus.flush();
         outRobotStatus.println("NQC return code = "+
p.exitValue());
           outRobotStatus.flush();
      }

       switch (p.exitValue()) {
       case  -3: outRobotStatus.println("RCX Errors Detected -
Robot is not turned on.");
               break;
       case -13: outRobotStatus.println("NQC Errors Detected -
Programming syntax error");
               break;
       default:  outRobotStatus.println("Hurray, it works!");
               break;
```

```
        }
        outRobotStatus.flush();

        // Display results on this Robot Server's screen
        System.out.println("Return code from NQC command (process)
= " + p.exitValue());
        System.out.println("Done executing");
        System.out.println(" "); System.out.println(" ");
    }

        /* Handle exceptions for waitFor() */

        catch (InterruptedException intexc) {

        System.out.println("Interrupted Exception on waitFor: " +
intexc.getMessage());

        }
        catch (Exception err) {
          err.printStackTrace();
        }

  } // runRobotCheckStatus


      // This method encodes robot file (avi type) to text, and
sends the file to MySend.java
      public static void sendEncodedVideo(String filename) throws
IOException {
                int chunkSize = 10;
                String outFileName = filename+".enc";
                PrintStream out = new PrintStream(new
FileOutputStream(outFileName));
                File inputFile = new File(filename);  // like
RoboIn.avi
                FileInputStream in = new
FileInputStream(inputFile);
                int c;
                int bcount = 0;
                byte[] b = new byte[chunkSize];
                while ( (c=in.read()) != -1) {
                        if (bcount>=chunkSize) {
                                // array is full, so encode it
and write it out
                                String e = encode(b);
                                bcount=0;
                                out.println(e);
                        }
                        b[bcount++] = (byte) (0xff & c);
                }
                if (bcount>0) {
                        String e = encode(b,bcount);
                        out.println(e); // send to the socket
                }
        }
```

```
        File outputFile = new File("RoboOut.avi");
        FileOutputStream out = new FileOutputStream(outputFile);

        } // sendEncodedVideo


} // myRecv5
```

// myReceive.java program (above).

APPENDIX E Pre/Post Test (Elementary School Group)

LEGO RoboCamp **(Pre Test)**

AT MEDGAR EVERS COLLEGE

NAME: _____ GRADE: _____

SCHOOL: _____ DATE: _____

WHAT WOULD YOU LIKE TO BE WHEN YOU GROW UP?

DRAW A PICTURE OF YOURSELF AND WRITE A SHORT PARAGRAPH ABOUT YOU AND YOUR FUTURE CAREER.

LEGO RoboCamp **(Post Test)**

AT MEDGAR EVERS COLLEGE

NAME: _____ GRADE: _____

SCHOOL: _____ DATE: _____

HOW HAS YOUR EXPERIENCES WITH ROBOCAMP IMPACTED YOU AND YOUR CAREER INTEREST?

DRAW A PICTURE AND WRITE A SHORT PARAGRAPH ABOUT YOUR CAREER INTEREST.

NQC/Robotics Pre/Post Test (Middle School Group)

Name: _____

Grade/Name of School: _____

Intended Major in College: _____

1. Please complete the following chart, indicating which programming languages you know, your level of proficiency and how you acquired knowledge of the language.

	PROFICIENCY LEVEL (check *one*)			HOW YOU LEARNED THE LANGUAGE (check *all* that apply)		
PROGRAMMING LANGUAGE	Slight familiarity	Basic proficiency	Advanced proficiency	Took a class	Self taught	Other (describe)

2. How do you learn best about programming? (Check one)
❒ Organized instruction
❒ Teaching self
❒ Other (describe): _____

3. a) Do you usually use a ❒ PC ❒ Mac
b) Are you comfortable using both platforms? ❒ Yes ❒ No

4. Have you participated in program/design competitions before? ❒ Yes ❒ No ❒ Unsure
If yes, where? (school, after-school program, etc.) _____
Did you work: ❒Alone ❒With others

5. Describe (briefly) what you think the design process is.

6. How would you divide tasks to complete a robotic design project?

7. Have you used NQC before this class? ❒Yes ❒ No

8. Briefly describe what the following NQC statements do or how they can be used.

```
int move_time;


move_time = move_time + 5;


if (Random(1) == 0)


Off(OUT_A+OUT_C);


#define MOVE_TIME 100
```

9. a) Re-write the following NQC statements to program a rover to move go forward a certain amount of time and then turn for a few seconds. Re-write the 5 statements below and to the right, to achieve this program behavior.

```
Wait(85);

OnRev(OUT_C);

OnFwd(OUT_A+OUT_C);

Off(OUT_A+OUT_C);

Wait(100);
```

b) Roughly draw the Lego design for a simple rover with a light sensor and a touch sensor. Label the important parts of the rover and their functions.

10. How confident are you of your NQC programming abilities?
- ❒ Very confident
- ❒ Moderately confident
- ❒ Slightly confidence
- ❒ Not very confident

11. How confident are you of your building abilities?
- ❒ Very confident
- ❒ Moderately confident
- ❒ Slightly confidence
- ❒ Not very confident

12. In your opinion, how applicable will what you learn using NQC be to other computer programming?
❒Very confident
❒ Moderately confident
❒ Slightly confidence
❒ Not very confident

13. Describe one type of data you could measure with NQC?

14. How many times did you test your robot before it worked properly?
❒ 0 ❒1-3 ❒4-7 ❒8-10 ❒more than 10

15. a) Did you change your design? ❒Yes ❒ No
b) If yes, how many times? ❒1-3 ❒4-7 ❒8-10 ❒ more than 10
c) Did you change your program? ❒Yes ❒No
d) If yes, how many times? ❒1-3 ❒ 4-7 ❒8-10 ❒ more than 10

16. a) Did you plan your robot design before starting to build? ❒Yes ❒No
b) If you planned before building, how did you plan? (check all that apply)
❒ In your head
❒ On paper
❒ By talking with your partner
❒ Other _____

17. Did your robot programs always behave like you planned?
❒Yes ❒ Mostly ❒ No

18. Describe any benefits that you experienced related to having a partner:

19. Describe any drawbacks or challenges you experienced related to having a partner:

20. How did you divide tasks with your partner? In other words, who did what? There's no need to mention names.

SELECTED BIBLIOGRAPHY

Arnow, D. (1999). An Asychronous Learning Network Tool for Improving CS Education and Retention Rates, National Science Foundation (NSF).

Arnow, D. (2000). ALN Tools For Improving CS Education, Grant Number: USE-9150719, 7/1/91-12/31/93.

Arnow, D. and O. Barshay (1998). WebToTeach: The Student Manual Preliminary Version, National Science Foundation (NSF CISE - 9522537).

Arnow, D. and O. Barshay (1999). WebToTeach: An Interactive Focused Programming Exercise System, National Science Foundation (NSF CISE - 9522537).

Arnow, D. and W. C. Harris (2007). "Remote Shared Access to A Classroom Robotics Lab." Spring 2007 Symposium. Robots and Robot Venues: Resources for AI Education. Technical Report: SS-07-09.

ARTSI. (2008). "ARTSI: Advancing Robotics Technology for Societal Impact " Retrieved January 18, 2008, 2008, from http://www.artsialliance.org/.

AXIS, C. (2005). "AXIS 205 Network Camera." 2005, from http://www.axis.com/products/cam_205/.

AXIS, C. (2006). "AXIS Camera Recorder." 2006, from http://www.axis.com/products/cam_rec_software/.

Baum, D. (2000). Dave Baum's Definitive Guide to Lego Mindstorms. Emeryville, CA, APress.

Beer, R. D., H. Chiel, J., et al. (1999). "Using Atonomous Robotics to Teach Science and Engineering." Association for Computing Machinery. Communications of the ACM **42**(6).

Bergin, J. (2006). Karel Universe Drag & Drop Editor. Annual Joint Conference Integrating Technology into Computer Science Education
Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education Bologna, Italy ACM: Association for Computing Machinery.

Biggs, G. and B. MacDonald (2003). A Survey of Robot Programming Systems. Australasian Conference on Robotics and Automation,, Brisbane, Australia.

BotBall (2007). Welcome to BotBall.

Carlisle, M. C., T. A. Wilson, et al. (2005). RAPTOR: A Visual Programming Environment for Teaching Algorithmic Problem Solving. SIGCSE 2005, St. Louis, Missouri, ACM: Association for Computing Machinery

Carnevale, A. P., L. J. Gainer, et al. (May 1990). Workplace Basics: The Essential Skills Employers Want, Training Manual. San Francisco, CA, Jossey-Bass Inc., Publishers.

Chiel, H. J. (1996). "Critical Thinking in a Neurobiology Course." Bioscene **22**(1).

Chrysanthakopoulos, G. and H. F. Nielsen (2007). Microsoft Robotic Studio Runtime Architectural Overview.

Clubhouse, I. C. (2008). "The Intel Computer Clubhouse Network " Retrieved January 18, 2008, 2008, from http://www.computerclubhouse.org/index.htm.

Earnshaw, R. A., M. A. Gigante, et al. (1993). Virtual reality systems. London, Academic Press.

Eccles, J. S. (1987). "Gender Roles and Women's Achievemen-Related Decisions." Psychology of Women Quarterly **11**: 135-172.

Eccles, J. S. and A. Wigfield (2002). "Motivational Beliefs, Values, and Goals." Annual Review of Psychology **53**(1): 24.

Eccles, J. S. and A. L. Wigfield (2000). "Expectancy-value theory of achievement motivation." Contemporary Educational Psychology **25**: 68-81.

FIRST. (2008, February 20, 2008). "For Inspiration and Recognition of Science and Technology." from http://www.usfirst.org/who/default.aspx?id=34.

FLL. (2008). "First Lego League." from http://www.legoeducation.com/about/relationships/item.aspx?art=3&bhcp=1.

Fuller, J. L. (1999). Robotics: Introduction, Programming, and Projects. Englewood Cliffs, NJ, Prentice-Hall.

Gates, B. (2007). Microsoft Chairman. USA Today. Microsoft, The Associated Press.

Geissler, J., P. J. Knott, et al. (2004). "Virtual Reality Robotics Programming Software in the Technology Classroom." The Technology Teacher: 6-8.

Goldman, R., A. Eguchi, et al. (2004). "Using Eduactional Robotics to Engage Inner-City Students with Technology." Proceedings of the Sixth International Conference of the Learning Sciences: 214-221.

IEEE. (2007). "IEEE Society of Robotics and Automation's Technical Committee on: Networked Robots." from http://www.informatik.uni-freiburg.de/~burgard/tc/.

ITEA (2000, 2002). International Technology Education Association: Standards for Technological Literacy. Reston, VA.**:** 33-40.

ITEA (2000, 2002). International Technology Education Association: Standards for Technological Literacy. Reston, VA.

Jackson, J. (2007). "Microsoft Robotics Studio: A Technical Introduction - Standarding Robotics Coordination and Control." IEEE Robotics & Automation Magazine **14**(4): 82-87.

James, R. (1985). Introduction to Robotics: A Systems Approach. Upper Saddle River, NJ, Prentice-Hall.

Jones, J., A. M. Flynn, et al. (1998). Mobile Robots: Inspiration to Implementation. Welleasley, MA, A. K. Peters Ltd.

Kelleher, C. (2006). Motivating Programming: using storytelling to make computer programming attractive to middle school girls. School of Computer Science. Pittsburgh, PA 15213, Carnegie Mellon University**:** 391.

Kumar, D., Meeden, L. (1992). "A robot laboratory for teaching artificial intelligence." Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (1998): 341-344.

Lego_Education. (2005). "Camp-on-a-Disk: P.R.O.B.E.", from https://www.legoeducation.com/store/detail.aspx?CategoryID=159&pt=8&ID=1072.

Lummis, M. and H. W. Stevenson (1990). "Gender Differences in Beliefs and Achievements: A Cross-Cultural Study." Development Psychology **26**: 254-263.

Mars_Stations. (2007). "Mars Stations." from http://www.planetary.org/programs/projects/drive_a_mars_rover/.

Matson, E. and S. DeLoach (2004). "Using Robots to Increase Interest of Technical Disciplines in Rural and Underserved Schools." Proceedings of the American Society for Engineering Education Annual Conference and Exposition.

Mattel (1992). Mattel Says It Erred; Teen Talk Barbie Turns Silent on Math New York.

Meyer, M. R. and M. S. Koehler (1990). Internal Influences on Gender Differences in Mathematics. Mathematics and Gender. E. Fennema and G. C. Leder. New York, Teachers College Press**:** 60-95.

Microsoft. (2006). "Microsoft Robotics Studio." from http://msdn.microsoft.com/robotics/.

Murphy, R. R. (2000). Introduction to AI Robotics. Cambridge, MA, The Massachusetts Institute of Technology Press.

NRF. (2007). "Network Robot Forum." from http://www.scat.or.jp/nrf/English/.

Papert, S. (1980). Mindstorms: Children, Computers, and Powerful Ideas. New York, NY, Basic Books:.

Powers, K., P. Gross, et al. (2006). Tools for Teaching Introductory Programming: What Works? Technical Symposium on Computer Science Education Houston, Texas, ACM: Association for Computing Machinery

Red_Rover. (2007). "Red Rover, Red Rover Project ", from http://www.planetary.org/programs/projects/red_rover_red_rover/.

Robocell, I. S. P. (2008).

RoboticsAllianceProject. (2008). from http://robotics.nasa.gov/about.php.

Rogers, C. (2001). LEGOS, ROBOLAB, and LabVIEW: Designing, Programming, and Collecting Data. Medford, Massachusetts, University of Wisconsin-Madison**:** 6.

Rogers, C. and M. Portsmore (2004). "Bringing Engineering to Elementary School." Journal of STEM Education **5**(3-4): 12.

Russell, S. and P. Norvig (1995). Artificial Inteligence: A Modern Approach. Englewood Cliffs, NJ, Prentice-Hall.

Spice, B. and A. Watzman. (2008). "University Alliance Works To Increase Robotics Education, Research at Historically Black Colleges and Universities." Retrieved January 18, 2008, 2008, from http://www.cmu.edu/news/archive/2008/January/jan8_artsi.shtml.

Tocci, C. M. and G. Engelhard (1991). "Achievement, Parental Support and Gender Differences in Attitudes Towards Mathematics." Journal of Educational Research **84**: 280-286.

TRLI, I. (2006). "Total Learning Research Institute, Inc. (TLRI)." from http://www.tlri.org/.

VR Robot, D. S. P. (2008).

Weinberg, J. B., J. C. Pettibone, et al. (2007). "The Impact of Robot Projects on Girls' Attitudes Towards Science and Engineering."