Designing a Computational Construction Kit for the Blind and Visually Impaired

by

Rahul Bhargava

B.S. Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA
2000

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences
at the
Massachusetts Institute of Technology

June, 2002

©Massachusetts Institute of Technology, 2002. All Rights Reserved

Author - **Rahul Bhargava**
Program in Media Arts and Sciences
May 10, 2002

Certified By - **Mitchel Resnick**
Associate Professor, Lifelong Kindergarten Group
Thesis Advisor

Accepted By - **Andrew B. Lippman**
Chairperson
Departmental Committee on Graduate Studies

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUN 27 2002
LIBRARIES

ROTCH

Designing a Computational Construction Kit for the Blind and Visually Impaired

by

Rahul Bhargava

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on May 10, 2002
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences
at the
Massachusetts Institute of Technology

Abstract

This thesis documents the adaptation and extension of an existing computational construction kit, and its use by a community of learners previously unaddressed – blind and visually impaired children. This community has an intimate relationship with the digital and assistive technologies that they rely on for carrying out their everyday tasks, but have no tools for designing and creating their own devices. Using a computational construction kit, created around the latest Programmable Brick (the Cricket), children can write programs to interact with the world around them using sensors, speech synthesis, and numerous other actuators. The Cricket system was extended with a number of specific modules, and redesigned to better suit touch and sound-based interaction patterns. This thesis documents an initial technology implementation and presents case studies of activities carried out with a small group of visually impaired teenagers. These case studies serve to highlight specific domains of knowledge that were discovered to be especially relevant for this community. Much of this work impacts approaches, technologies, and activities for sighted users of the Programmable Brick.

Thesis Supervisor: Mitchel Resnick
Title: Associate Professor

This work was supported in part by:

- The LEGO Company
- Media Lab Asia
- National Sciene Foundation (grant #ESI-0087813)
- Learning Lab Denmark
- Things that Think Consortium, MIT Media Lab
- Digital Life Consortium, MIT Media Lab

Designing a Computational Construction Kit for the Blind and Visually Impaired

by

Rahul Bhargava

Thesis Committee

Thesis Advisor - **Mitchel Resnick**
Associate Professor, Lifelong Kindergarten Group
MIT Media Laboratory

Thesis Reader - **Bakhtiar Mikhak**
Research Scientist, Learning Webs Group
MIT Media Laboratory

Thesis Reader - **Sile O'Modhrain**
Principal Research Scientist, Palpable Machines Group
Media Lab Europe

Designing a Computational Construction Kit for the Blind and Visually Impaired

by

Rahul Bhargava

Thesis Committee

Thesis Advisor **- Mitchel Resnick**
Associate Professor, Lifelong Kindergarten Group
MIT Media Laboratory

Thesis Reader **- Bakhtiar Mikhak**
Research Scientist, Learning Webs Group
MIT Media Laboratory

Thesis Reader **- Sile O'Modhrain**
Principal Research Scientist, Palpable Machines Group
Media Lab Europe

Acknowledgements

First and foremost, I'd like to thank my family for helping me get here.

Special thanks to Mitchel and Bakhtiar, for guiding me through this sea of big words and powerful ideas. And to Sile, who provided the inspiration for this work and kept me grounded in my explorations.

A huge collective thanks to Daniel, Casey, Tim, and Nell for peppering my sanity with appropriate amounts of insanity just when I needed it. Good times, my friends, good times.

An additonal thanks to all the members of the Lifelong Kindergarten and Learning Webs groups, for doing all this fantastic stuff that I could build on.

Table Of Contents

Chapter 1 - Introduction

1.1 Computers and Learning

In recent years, computers have become established elements of many types of learning settings, used in different ways for various learning activities. Unfortunately, the power of the computer as a flexible medium for designing and creating has often been ignored. Tools that are designed to let users create their own constructions empower learners to engage computation as a creative material. The process of creating these artifacts opens the door to a myriad of learning opportunities, many in fields that are otherwise difficult to approach. More specifically, in the process of creating computational objects, many of the underlying ideas of computation itself can be explored.

The blind and visually impaired often interact with more computational devices in learning settings, but have even fewer opportunities to create with computation itself. The visually impaired° use tools such as adjustable magnifiers, text scanners, and speech-synthesis devices to access curricular materials that are otherwise inaccessible. Visually impaired people use computers to do word processing, send email, and surf the web, among other things. However, none of these tools take advantage of the opportunity to engage learners in explorations of how these computational tools can be used to create their own artifacts. The growing reliance on "black-box" devices in learning settings can be

° Throughout the rest of this thesis, I will adopt the common shorthand of using the term "visually impaired" to refer to those who are blind and those who are visually impaired to some degree. I will clarify when needed.

inherently disempowering to the learner – denying them not only of an understanding of their functionality, but also the chance to explore the rationale behind their design.

1.2 New Tools for New Explorations

Playing with computational construction kits can allow people to create in new ways and engage new fields of knowledge. A computational construction kit can be described as a set of tools or objects that allow one to create a computational artifact. They build on the tradition of existing children's construction kits, such as LEGO bricks and Erector sets, by giving children a set of digital building blocks. The Programmable Brick is a computational construction kit that allows users to create behaviors for their constructions in the physical world (Resnick, 1993). It can be programmed to interact with the world around it using a wide range of sensors and actuators. These extensions, in addition to the "brain" that is the Brick itself, are the components of the kit. The Programmable Brick has been developed and iterated upon for the past 15 years in the MIT Media Lab's Epistemology and Learning Group (http://el.www.media.mit.edu/).

Creating a computational construction kit is "a type of meta-design: it involves the design of new tools and activities to support students in their own design activities" (Resnick, et al. 1996). Giving learners the tools to create and build their own computational artifacts opens the door to exploring new ideas.

In order to address some of the issues presented, I have created a Programmable Brick for the visually impaired, called the Bricket. This thesis documents the hardware and software interfaces redesigned to meet the learning needs of the visually impaired. I developed a series of activities that use the Bricket to explore a set of approaches to learning and specific skills to learn.. These activities were conducted with a group of three visually impaired children, all in their early teens.

A user interacts with a Programmable Brick by creating a software program on a host unit, usually a desktop computer, and then downloading it to the Brick's memory. The Brick can then run the program to inter-

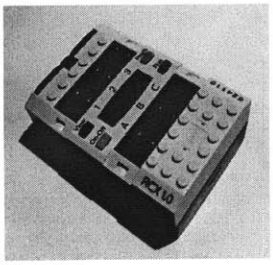


Figure 1.1 : The LEGO RCX brick.

act with the world around it. An older Programmable Brick evolved into the commercially available LEGO Mindstorms product *(figure 1.1)*, based around the RCX (also referred to as the "yellow brick"). The latest Programmable Brick is known as the Cricket *(figure 1.2)*. The Bricket *(figure 1.3)*, created for this thesis, is an adapted version of the Cricket. Users create programs for the Bricket on a Windows PC running an application called BricketLogo. This application implements a small scripting version of the LOGO programming language.

1.3 A Programmable Brick for the Visually Impaired

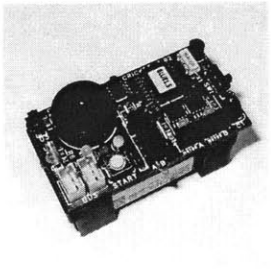


Figure 1.2 - The MIT Cricket.

The Bricket was created to bring these technologies and activities to the visually impaired community. The inspiration for this project came from Sile O'Modhrain, while she was at the MIT Media Lab as a visiting researcher°. Sile is visually impaired, and conversations with her led to the idea that this could be an interesting community to work with.

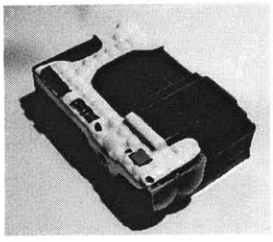


Figure 1.3 - The Bricket.

There were a multitude of reasons that the visually impaired community was an appropriate one to work with. Technologically, the basic sensors of the Programmable Brick, used with various actuators, are well suited to representing specific components of visual feedback. Simple resistive light sensors and optical distance sensors can replace things such as light and distance, which are usually sensed through vision. These basic sensors are also well suited to the limited processing power of the Programmable Bricks. From a theoretical point of view, working with this community presents an opportunity to take the research of the Epistemology and Learning group in a new direction. Addressing the learning needs and learning styles of the visually impaired led to reflections about the foundational frameworks we use in our research.

Many methodologies have been created to guide designing for disabled communities, but most share a core high-level of interaction with the community. Believing that to be a key ingredient to successful designs, I engaged two members of the visually impaired community to aid and guide me. The first was Sile O'Modhrain. The second was Rich Calaggero, a visually impaired programmer who works in the MIT assistive technologies office (ATIC). I held weekly consultations with both to get

° Sile O'Modhrain is currently a Principal Research Scientist and head of the Palpable Machines Group at Media Lab Europe.

immediate feedback on design decisions.

1.4 Reading This Thesis

This document is organized into seven chapter..

Chapter 1 - Introduction

This chapter presents a summary of this research project and a rationale for this investigation.

Chapter 2 – Working with the Bricket

This chapter tells the story of one device a participant in the study created, in order to provide a grounding for the discussion that follows.

Chapter 3 – Theoretical Foundations

This chapter presents my own interpretations of the theoretical foundations for this study, as well as related research.

Chapter 4 – Technology Design

This chapter discusses how the theories manifested themselves in the objects created for this thesis. It presents extended descriptions of the technology created and a rationale for the design decisions that were made.

Chapter 5 – Activity Design

This chapter explains the activities I planned to do with the participants of my study. It provides an in-depth description of the study carried out.

Chapter 6 – Case Studies

This chapter presents three case studies of participants and the objects they created. The projects are compared to the metrics established earlier through quotes and descriptions of objects created.

Chapter 7 – Reflections

This chapter wraps up with further discussion of lessons learned from the study and interesting questions that arose.

I have laid out this document to facilitate an agreeable reading experi-

ence. While main text flows down the inside of either page, the margins provide space for related notes. A variety of symbols (°, *, or ^) denote that the reader should look to the margins for further information on some topic. Some tables and figures appear in the margins, while others appear in-line with the body text. Figure labels appear in the margins. This document is meant to be printed two-sided, and to be read as a book would.

Chapter 2 - Working With the Bricket

In order to give the reader a better idea of what one can create with the Bricket, it is necessary to provide an example. What follows is a brief summary of the process of creating a simple computational artifact with the Bricket. This short story describes what I did with one of the participants of my small study in our first session together.

Users create a program for the Bricket with an application called BricketLogo, which runs on a Windows desktop PC *(figure 2.1)*. Accessibility is built in for the visually impaired via screen reading software, which reads out text as users type, in addition to speaking out interface items. To transfer the program to the Bricket, the code is downloaded to an Interface that is attached to a serial port of the desktop PC. The Interface transfers the program to the Bricket through infrared communication (much like a TV remote control). The Bricket can then run the user's program.

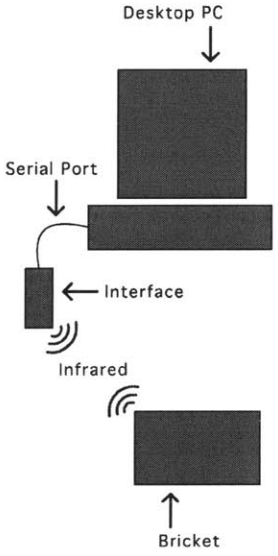


Figure 2.1 - Creating and downloading a program to the Bricket.

2.1 Saying "Arf"

In order to introduce Jim to the Bricket, I began by showing him how to use its speech synthesis capabilities. Speech is the Bricket's primary mode of output and I assumed it would become critical for the rest of our constructions.

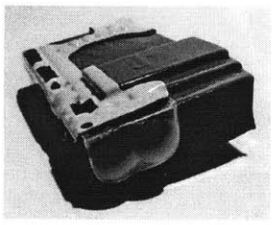


Figure 2.2 - The Bricket's infrared components lie beneath the rounded translucent shielding.

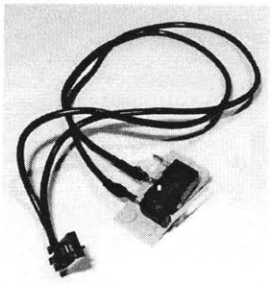


Figure 2.3 - A touch sensor.

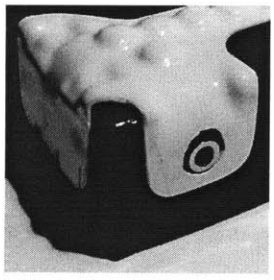


Figure 2.4 - The Bricket's headphone plug.

Using the BricketLogo application to write a program for the Bricket, Jim and I began to make simple programs that would have the Bricket respond to a touch switch. Since Jim liked dogs, I suggested having the Bricket say "arf" when a touch switch was depressed (hoping it would sound like a dog barking). This involved a simple program, created in a variant of the LOGO programming language, such as:

```
when [ switcha ] [ say "arf ]
```

This established an interrupt when executed. This means that whenever the touch sensor plugged into sensor port A is depressed, the Bricket says the string "arf". Jim was able to create this program by using a regular keyboard and a piece of screen reading software.

To transfer this program to the Bricket, we first lined up the Bricket's infrared window with the interface, which was attached to the serial port of the computer. Jim did this by finding the two rounded protrusions on the Bricket's case *(figure 2.2)*, and similar ones on the interface – these indicated where the infrared signals were being emitted (and thus had to be pointed at each other). By selecting "Download" from the "Bricket" menu, we compiled and downloaded our simple program to the Bricket over infrared. The Bricket saves it to memory, so that it remains even if the battery is removed. This downloading process took only a few seconds. The BricketLogo application spoke "download complete" to indicate a successful download.

While the program was ready to use, we had yet to plug in the components it expected. Jim located a touch switch *(figure 2.3)*, and by tracing its wires with his fingers, he was able to plug it into one of the sensor ports. It locked with a simple snap mechanism, so Jim knew it would not be pulled out by accident. By tactily following the bumps on the Bricket case around to the headphone port *(figure 2.4)*, Jim was able to attach his headphone plug.

With the program loaded and the various components connected, we were ready to test our simple program. Jim pressed the touch sensor and it said "arf". This creation took only a few minutes to make.

Chapter 3 - Theoretical Foundations

3.1 Theories of Learning

In this thesis I draw on a wide variety of approaches to learning. These theories have guided me in my study plan, technology development, and evaluation. Thus it is necessary to give a snapshot of my own interpretation of the core ideas proposed by the epistemologists, philosophers, and educators I build upon.

3.1.1 Constructivism and Constructionism

Crafted by Jean Piaget, constructivism is a theory of a child's psychological development proposing primarily that children are active constructors of their own knowledge and understanding (Piaget, 1963). Piaget revised previous opinions of children's understandings, which labeled them as "incorrect", to instead demonstrate that children develop their own theories and rationales, and that these are internally consistent. In fact, he saw that process of creating and revising theories of understanding as the learning process. His epistemology, his theory of knowledge, therefore brought a legitimacy to the world of children that forms a basis for much of what gets talked about in the field of education today.

Seymour Papert, who worked with Piaget, built on Piaget's approach to propose a philosophy towards learning called "constructionism."

Constructionism values the act of creating and reflecting on artifacts as a learning process (Papert, 1994). The artifacts people create are seen as the embodiment of their understanding of concepts and ideas. Putting these understandings in the physical world allows for an opportunity to reflect and discuss them with peers. Papert also argues against Paiget's emphasis on abstract scientific thought as the goal of learning. This, for him, ignores the value of contextualized, concrete knowledge.

As with this thesis, other studies with the Programmable Bricks were driven by the desire to enable learners to create their own computational devices. The decision to empower learners with a tool to build with, rather than simply providing them with some kind of finished product, is at its base a constructionist approach.

3.1.2 Authentic Experience

The idea of "authentic experience" provides a rationale for the contexts in which I worked with learners. The term has been used by many to describe their work, but at its core lie three central ideas. One of these is John Dewey's notion of "continuity of experience" – the idea that what is learned should be valuable to the learner's current line of interest or inquiry (Dewey, 1938). Another base is Donald Schön's criticism of the laboratory environment as a dishonest setting for learners (Schön, 1987). Schön proposes that laboratories attempting to mimic real-world situations introduce false notions of "right answers", rather than focusing on methods of reasoning. Common threads can also be found in descriptions of "situated learning" (Lave and Wenger, 1991). Situated learning suggests that learners build understanding through concrete experiences in situations. While specific metrics and classifications of "authenticity" may vary, these basic guiding principles are common.

I have taken these foundations as the basis for many of my decisions about the technologies and activities developed for this thesis. My own interpretation of authenticity implies a number of things. For one, I respect the world of the learner. I take "real-world" to imply the world of the learner, and thus attempt to situate all their experiences. I also believe that the concrete experiences I design lead to growing understand-

ings of their underlying ideas. These situated experiences make the ideas authentic, rather than abstract notions introduced in some foreign setting. I have designed my activities to be driven by the children's interests, which I believe makes them inherently authentic. In addition, the themes of all my sessions were built around creating objects that they could use to perform some fun or necessary task. Rather than thinking about their visual impairment as a hindrance to doing these activities, I saw it as an opportunity to make the activities more relevant and motivating. I have also attempted to maintain an honest attitude when working with the learners – being clear about what metrics I designed for and what my goals were.

3.1.3 Learning Relationships

Many of our culture's most popular technologies have been so widely adopted that they have become practically invisible. Automobiles, telephones, and radios are just a few. They are the "black boxes" so often mentioned – technologies that work, but provide no means to understand or relate to how they work (Resnick, et.al. 2000). When used in learning contexts, these black boxes can be inherently disempowering to the learner, leading them to often assume that they could not understand them even if they tried to.

Technologies that address this problem, ones which can be tinkered, tweaked, and played with to discover how they work, allow for what I call "learning relationships" to develop. Understanding how a tool works, and being given the power to alter how its mechanisms perform some task, can fundamentally change how one relates to it. Learning environments can, in part, be defined by the quality of learning relationships cultivated by the tools that populate them. Just as there are different levels of knowing and understanding, there are different levels of learning relationships that can develop. Computational tools, in particular, can allow for powerful learning relationships to be developed if the tools take advantage of computation's high level of "tinkerability".

In the visually impaired community, most assistive technologies do not cultivate learning relationships with their users. This is not to invalidate

the critical role assistive technologies serve, but rather to provoke an exploration of how reliance on these tools is addressed in learning environments. A computer, when being used to access information that is otherwise inaccessible, should act like an assistive technology. Using a speech synthesizer to have a book read aloud to a visually impaired user should not be a cumbersome task. However, in a learning environment, there should be some way to explore their functionality further.

3.2 The Content and Process of Learning

My main goal is to push forward the development of learning approaches, activities, and technologies by bringing them to a new community. Examining how this community engages the tools I introduce brings useful reflections to the larger goal of the work around Programmable Bricks. This work also allows for an opportunity to engage in a discussion of the growing role technologies play in learning settings.

I am guided by the desire to give a community another tool to create and express with. Tools like the hammer can be thought of in two ways. The first is to consider the utilitarian functionality of being able to nail two pieces of wood together to create a longer piece – a functional need. The second might be nailing pieces of wood together simply to see what shape appears – a creative exercise. I find this approach apt for the tool I am introducing to this community. The Bricket can be used to build something like a distance-sensing cane – a functional need. However, the Bricket can also be used to make a musical clock – a creative exercise. Obviously these two areas overlap, in that a musical clock also fulfills the functional need of telling the time, but the role of the tool is multi-faceted.

More specifically, I have a set of ideas and skills I want to introduce to the learners I work with. These can be roughly divided into two categories – approaches to learning, and things to learn. However, this distinction is not to say that I ascribe to the traditional "know how" vs. "know what" paradigm of schooling. Building on the pedagogy best expressed by those who write about "situated learning", I don't view learning what something is and learning how something can be used as distinct enti-

ties (Brown, 1989). In fact, they coexist and should thus be explored together. One often best learns the "how" by doing the "what".

3.2.1 Ways to Learn

Beyond guiding my own approach to this work, the theories of learning I introduced earlier are also concepts I would like the learners to explore. These meta-level approaches to learning are important reflections about the process I went through with the participants of my study. By creating their own computational devices, I wanted the study participants to become reflective about the designs of the devices around them. By introducing the Bricket in the home environment, I sought to provide an example of how to develop a learning relationship with a technology. By empowering visually impaired children to create their own assistive devices, I hoped to give them a chance to open the black boxes that are the technologies they might use in a school-based learning setting everyday.

Reflection and Critique

Drawing on the traditions of design education, I wanted to encourage participants to become reflective about the designs of the objects around them. Because they are visually impaired, the learners I worked with were more attuned to the tactile and auditory aspects of these tools. This offered an opportunity to harness a reality of their daily life, around which I designed contextualized learning experiences. Here I draw from Schön's "reflection-in-action" approach to design education (Schön, 1987).

While constructing their own assistive devices, participants were not only exploring technical skills, but were also building critical analysis abilities. When recreating an object from basic building blocks, questions about their construction are raised. Why was one method chosen over another? Why wasn't some other technology incorporated? Why was one interaction pattern deemed more efficient or intuitive? These questions provide opportunities to engage in reflective discussions that build critiquing skills.

Iterative Design

An iterative design process is inherent to building with the Programmable Brick (Martin, et.al 2000). One mode of programming the Programmable Brick is specifically designed to be iterative in nature°. This "try it out" attitude exemplifies one of the critical components of a constructionist approach – the idea that one makes something, reflects on it, and then remakes it, over and over.

The medium of computation provides magnificent opportunities for iterative design because of its highly flexible nature. Novice users can make quick iterations because software makes it easy to change the control of a computational device. Specifically with Programmable Bricks, the entire functionality of the object can be changed quite simply. These new technologies are allowing learners to explore iterative design in a way that was never before possible.

3.2.2 Things to Learn

In addition to approaches to learning, I emphasized a number of specific skills to learn. The skills I focus on here are the fundamentals of building these types of devices. The crucial part about the Programmable Brick is that it can be used to create objects that were never before possible. Thus they can open up the doors to multitudes of potential new learning experiences. I choose to structure around three main topics, and design activities that let the learners explore them. As my study progressed, these ideas focused onto programming, sensing and control, and representation.

Programming

In mechanics, the physical shapes of the objects one puts together let one create behaviors. With computation, programming is the method of accomplishing the same task. By introducing the children to programming I hope to introduce them to the world of creating with computational artifacts. Programming, by its very nature, leads to process-oriented ways of thinking. Manipulating variables explores ideas of how numbers can be played with, in a mathematical sense. For

° Please refer to the discussion of the BricketLogo application in section 4.1

this population, that is particularly powerful because of the permutations that can be made on digital information to represent it in an accessible and meaningful manner. In addition to introducing ways of thinking, programming is also a skill that can become valuable later on. If learners wanted to move forward in pursuing programming as an interest, communities exist to support them°.

Sensing and Control

Working with sensors gives learners an opportunity to engage in discussions of their own senses. Visually impaired people come to rely on their tactile and auditory senses more, leading me to presume that activities around sensing would generate interest in visually impaired learners. Letting learners create objects with electronic sensors leads to thinking about how their own bodies interact with the world around them.

In addition to providing a means to discuss our own senses, working with sensors requires knowledge of traditional mathematical operations. Techniques for averaging and thresholding are necessary for dealing with almost all sensors. Following the pedagogy I laid out earlier, these skills were encountered while "doing" - in real situations, rather than constructed or artificial settings. These methods drew on the relevant areas of robotics, control theory, and digital signal processing*. This was all contextualized within the goal of making devices that were personally meaningful to the creator.

Representation

One of the primary issues of handling digital information, handling bits, is how to represent them. For the visually impaired community, this becomes an even more important issue because while there is a large of amount of visual data to be gathered, there is one less channel available to represent that back to the user. Mapping senses between output modalities brings up issues of what the actual information being represented is. These issues can lead to interesting discussions about the nature of the information gathered, or how the sensor works, or why one representation might be more intuitive than another.

° Please refer to the later discussion of adult progammers who are visually impaired for more detail, in section 3.3.2.

* Please refer to the case studies in Chapter 6 for concrete examples.

Take, for instance, the idea of a device for the visually impaired that is able to detect when an obstacle is in front of its user. One mapping might output a beep that corresponded in pitch to the distance detected. However, questions of whether the pitch moves up or down when an object is closer seem to have no intuitive answer. A more suitable mapping might be to output a verbal "watch out" message. This builds on the experience of having a friend guide a visually impaired person around by the arm, verbally warning them when something is in their path. Alternate representations such as these can be explored and perfected through an iterative design process.

Many studies have explored the idea of representation of scientific data. In particular, some creations that were part of the Programmable Brick based Beyond Black Boxes project explored alternate representations for information (Resnick, et.al. 2000). Another activity, the Sensory Design Workshop, explored the idea of making sensors and representations more closely°. Building on this work, I will demonstrate that these questions are crucial to the visually impaired community.

3.3 Existing Practice

Although I made the conscious decision to work outside of a school setting, the practice and approaches towards schooling the visually impaired informed me and provide a background context in which to place my activities. Having no experience working with the blind, I found it necessary to explore the existing practice and discuss approaches with current practitioners.

3.3.1 Educating the Visually Impaired

The current prevailing trend is to integrate visually impaired children into public school systems. The reasons for segregation are primarily historical. The visually impaired used to be limited to vocational training in areas such as furniture making, basket-weaving, or piano tuning (as described by Fries, 1980). This has shifted for a number of reasons. In the United States, governmental regulatory policies have required schools to provide accessible versions of curricular materials. This

° The Sensory Design Workshop was organized and run by Bakhtiar Mikhak, Research Scientist and head of the MIT Media Lab's Learning Webs group.

has allowed the visually impaired to gain access to many of the same resources as their peers (exemplified recently by the Individuals with Disabilities Education Act). Additionally, many technologies have been created that make it easier for the visually impaired to interact with curricular materials. With accessible resources in place, visually impaired students are able to enter standard public schools and perform as their sighted peers. While many special schools for the visually impaired do exist, they usually focus on offering services for those with multiple disabilities.

However, those who integrate into standard school settings encounter many other difficulties. These students usually are engaged in after-school activities to learn Braille reading and mobility skills. The visually impaired students must also learn how to find fellow students who are willing to read them any texts that cannot be found in an accessible format (a problem that gets more critical as they move to university settings and specialized fields of interest). Efforts must be made to make other students more aware of the visually impaired student's needs (Krebs, 2000). These extra efforts consume considerable amounts of time and effort from the students and their families, often causing hardships. However, visually impaired students have been shown to be able to move through school-sanctioned curricula at the same rate as their sighted peers°.

Universal Design for Learning

The approach of Universal Design for Learning (UDL), proposed by the Center for Applied Special Technology (CAST), in part seeks to address the problems of handicapped students in schools. Speaking to curriculum designers and their goals, UDL proposes the following main points (from http://www.cast.org):

1) UDL proposes that children with disabilities do not constitute a separate category but instead fall along a continuum of learner differences.
2) UDL leads us to make adjustments for learner differences in all students, not just those with disabilities.
3) UDL promotes the development and use of curriculum

° Further discussion of these issues can be found in (Holbrook and Koenig, 2000).

materials that are varied and diverse, including digital and online resources, rather than centering instruction on a single textbook.

4) UDL transforms the old paradigm of "fixing" students, so that they can manage a set curriculum, into a new paradigm that "fixes" the curriculum by making it flexible and adjustable.

Much of the UDL philosophy resonates with the theoretical foundations presented earlier. Like CAST, I find it is crucial to make a concrete differentiation between tools that allow the visually impaired to access information and tools that the visually impaired can learn with (Rose and Meyer, 2000). I also agree that each learner is an individual, with unique learning styles, needs, and interests (as indicated by the second directive listed above). I have attempted to design the Bricket technology to fit the learning needs of the visually impaired, and my activities to allow learners to follow their interests with their own styles.

However, I am not presenting an argument aimed at curriculum designers, as UDL does. I have laid out a set of ideas to explore in this initial activity, and have guided a set of learners through that space. Their own interests and excitement have determined which ideas they have focused on, rather than any set of specific curricular goals I developed. My goal is to provide a new community with interesting entry points to the approaches to learning and skills to learn discussed earlier.

3.3.2 Assistive Technologies

There exist a wide variety of assistive technologies to make various everyday tasks accessible to the visually impaired. I define an assistive technology as any device that allows a disabled person to accomplish some goal or perform some task that is otherwise unattainable. These include things such as:

- an audible battery tester (manufactured by Ann Morris Enterprises - http://www.annmorris.com/)
- a talking thermometer (available from EnableLink - http://easycarts.net/ecarts/Enablelink/)

While these technologies are necessary in learning environments to

access materials such as textbooks or handouts, building a reliance on them without any understanding of their inner functioning can be inherently disempowering for learners.

Computers and the Visually Impaired

Historically, the visually impaired were able to access computers due to the raw-text based nature of their displays. The command line was highly accessible through the use of text-to-speech output, early attempts making even the first personal computers accessible (Vincent, 1982). However, the advent and subsequent proliferation of the Graphical User Interface (GUI) pushed more and more information into visual representations, shutting the visually impaired out. This focus on visual representation has moved to the web, where much information is still unavailable to the visually impaired (even with the Lynx text-only browser).

However, a visually impaired person entering the world of computers is presented with a wide variety of options. One path is to engage a command-line based operating system, most likely Linux. This option is appealing due to the simplicity of interacting with text-based display systems. Another path is to use the popular Microsoft Windows operating system. The GUI can be made accessible through the use of third-party software, and a vast majority of users are familiar with it°. Since speech output alone is sometimes inappropriate, many users also purchase a Braille printer, or a Braille display. Braille printers can emboss standard rolls of paper with special Braille fonts. Braille displays have one or two lines of actuated Braille dots, so any letter can be represented along the line. These lines change as the user moves through a body of text. However, more specialized products such as Braille note-takers are custom built for this population and are thus often more suitable. These note-takers fit somewhere between the worlds of computers and personal digital assistants (PDAs). They usually include a small Braille display. While some even run Windows CE, they are primarily meant as note-takers and communications device, rather than a full-fledged PC. However, for many users (visually impaired or not), these functionalities suffice.

° Unfortunately, the options for using an Apple Macintosh are quite sparse. However, the next version of the OS X operating system will include both a screen magnifier and a screen reader.

Accessing a GUI

Several assistive technologies exist that provide the visually impaired access to GUI-based computers. For those with limited vision, often a screen magnifier will suffice for them to interact with a computer. As the name suggests, these tools magnify the screen to make all the elements appear bigger. The user can control the scale of magnification, but otherwise the interaction is the same as a sighted user.

The primary mode of interaction for those who are severely visually impaired is a screen reader. These use speech-synthesis technology to speak the information on the screen. Common tasks such as mouse movement are tied to various keys on the keyboard (usually the numeric keypad). Intelligence is built into the screen reader that allows the keypad-mouse to move from button to button, so as not to force the visually impaired user to navigate over the confusing "empty" space between active GUI items they can interact with. These types of screen readers begin running at boot-time, existing between the operating system and any applications running.

The movement to build in accessibility to operating systems has been gaining momentum recently, across many platforms. The two main screen-readers for Microsoft Windows, JAWS and Window-Eyes, are both tightly integrated into the Microsoft Foundation Classes (MFC). The Java standard library of interface widgets, known as Swing, has been similarly tied to JAWS for Windows. On the Linux side, GNOME 2.0 will have similar built-in hooks for screen readers such as Gnopernicus and Speakup. SuSE (a distribution of Linux), in fact, already ships with support for SuSE Blinux – a screen-reader that also works with Braille output devices. For web content, the latest version of Macromedia's Flash authoring environment includes accessibility features. We are almost to the point where OS-level accessibility hooks become an assumption. Companies that wish to enter the US government market have driven much of this development, due to the fact that all US government software must adhere to strict accessibility standards. Unfortunately, those outside the US often cannot utilize these technologies because they are developed for the English language.

Outgrowths from the field of Human-Computer Interaction (HCI) have also engaged the issue of how visually impaired users interact with computers. The two most relevant to my work are the "earcon" and the "auditory icon", both of which strive to use non-speech audio to represent interactions with a computer. An earcon is a non-verbal audio message to the user about some kind of object on the computer (Blattner, et al., 1989). An example might be a sequence of tones, increasing in pitch, that play when a user selects the Open command in an application. Earcons must be learned over time, as they are usually abstract series of notes or tones. An auditory icon, on the other hand, attempts to use a natural or everyday sound to allow the user to intuitively discern its meanings (Gaver, 1986). In the same situation, that of a user selecting the Open command, an auditory icon might be a recording of a file drawer opening. Both of these ideas provide another aid to visually impaired computer users and influenced my thoughts on how to make my application accessible.

How the Visually Impaired Program

The access computers give to the visually impaired, and the powerful manipulations that programming allows for, have led to the development of an active online community of visually impaired programmers. The largest group is simply known as the "Blind Programming Site" (http://www.blindprogramming.com/). They provide speech-friendly tutorials, documentation, and tools. They also run an online discussion group called "blindprogramming" (http://groups.yahoo.com/group/blindprogramming/). This group is quite active, with over 400 members, and receives around 10 to 15 messages each day. The topics are usually questions about accessibility features of certain development environments, or general questions asking for programming assistance.

Their needs have led to the creation of various accessible development environments. The most comprehensive of these is Emacspeak, a set of extensions to the classic Emacs editing application for Unix (http://www.cs.cornell.edu/Info/People/raman/emacspeak/emacspeak.html). Emacspeak is open-source, and runs on various versions of the Linux operating system. It has been designed to let a visually impaired user

access everything from their desktop to the web. Beyond this, various editors and applications have reputations as being more speech-friendly than others.

The most targeted effort has come in the form of JavaSpeak, an Integrated Development Environment (IDE) from Winona State University's Computer Science Curriculum Accessibility Program (http://cs.winona.edu/csmap/javaspeak.html). JavaSpeak is a programming tool built to help undergraduate students learn how to program in Java, and to aid in their understanding of the ideas of computer science. Taking advantage of the fact that program code is highly structured, even though it is voluminous and hard to navigate, JavaSpeak allows users to browse multiple levels of representation (Smith, et al., 2000). They are currently in user testing with three visually impaired students in the Winona State University undergraduate computer science program (Francioni and Smith, 2002). The JavaSpeak tool provides a good number of ideas for making programming environments to support visually impaired learners.

These trends bode well for the visually impaired programmer. More and more research is pushing into the area of empowering the disabled to engage the world of programming. Similarly, a vast amount of corporate and volunteer effort is being made to tie in accessibility at the lower level of an operating system's foundation and GUI widgets.

Chapter 4 - Technology Design

The design of this computational construction kit was carried out with the help of a small group of visually impaired adults. Many methodologies exist for participatory design scenarios (Reich, et al. 1996), but all share the core of consulting with the community early and often°. These formed the basis of the simple rules I followed in my design process. Weekly meetings were used to solicit feedback on hardware and software prototypes, and to ground my own biased assumptions about the needs and preferences of the visually impaired community. This chapter presents a discussion of the rationale and implementation of the software and hardware created for this thesis.

The Bricket kit has two main technology components – the BricketLogo programming application, and the Bricket hardware. When a user downloads their code to the Bricket, it is first passed through a compiler, where it is converted into a byte-code format. The BricketLogo application then sends the bytes to the Bricket over infrared via an interface plugged into the computer's serial port. The Bricket then writes this program to its off-processor memory. When told to execute the program, a small interpreter on the Bricket steps through the instructions saved from the last download.

4.1 Software

° A brief history can be found at "Participatory Design History" <http://www.cpsr.org/conferences/pdc98/history.html>.

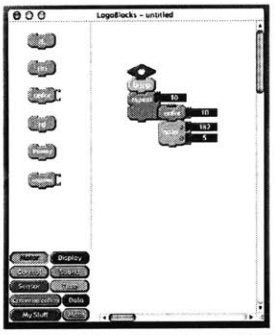


Figure 4.1 - LogoBlocks allows users to create programs by dragging visual blocks on a computer screen.

Creating a programming interface for the visually impaired requires rethinking the common practice of programming. As the reliance on visual metaphors for computer interaction has grown, so has the visual focus of programming tools and approaches. The tools are becoming more and more complex to manipulate, for even the sighted. Unfortunately, support structures for the visually impaired programmer have been largely ignored.

The Cricket can be programmed from the host unit in a variety of ways. As mentioned earlier, one can use a small text language based on LOGO, known as CricketLogo. CricketLogo is a small procedural scripting language, supporting standard features such as variables, procedure arguments and return values, control structures, in-code comments, and other commands to control its input and output capabilities. The Bricket uses an extended version of this language, called BricketLogo°. Another programming environment, called LogoBlocks, is provided for novice programmers. LogoBlocks is a visual language, where commands are blocks on the screen that snap together to create a program *(figure 4.1)*. The decisions made to create these programming environments informed my own implementation of the BricketLogo application.

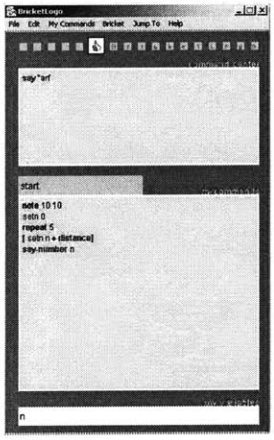


Figure 4.2 - The Bricket-Logo interface.

The BricketLogo application has three main interface areas *(figure 4.2)*. The first area introduced is the Command Center. A user can type of single line of code here, press enter, and have it immediately run on the Bricket. This area is meant for highly interative testing of ideas. The larger text area is for editing Commands that get downloaded to the Bricket. The third and smallest area, the Variables field, is a single line for defining a space-separated list of global variables.

4.1.1 Organizing Text

The visually impaired find large bodies of text difficult to navigate. However, while computer programs are often large blocks of text, they are highly structured with a known grammar. Taking advantage of this fact can make code more easily manageabl. If a visually impaired user is presented with options to navigate through code by method names, for

° Please refer to the appendix for a complete reference to the BricketLogo language.

instance, they can more quickly get to the one they wish to edit. This effectively replaces indentation as an aid to visual scanning. Smith, et al (2000) proposes a distinction between "syntactic structure" and "organizational structure" for their JavaSpeak application. They view syntactic structure as the inherent grammar of the programming language, while organizational structure is the manner in which a programmer breaks up the code to aid in their own comprehension and planning. My inclination was that organizational structures could be suggested that are friendlier to a visually impaired programmer.

However, many qualities differentiate BricketLogo from larger programming languages. Most notably, BricketLogo is a scripting language with a small command set. This allows for completeness in describing the language to users, rather than limiting exposure to relevant parts. Also, the size of programs created by users is limited to about 2kB of memory, and even that is much more than the usual page or two of code needed to build a complex project.

With these qualities in mind, I enforced certain types of organization to aid a visually impaired user in learning the BricketLogo language and editing their code. These build on some inherent features of the language, such as its small size and limited syntax, that make it easier to learn. The syntactic organizational rules are fairly small, leaving out things such as end-of-line markers, that can be cumbersome for the visually impaired. I decided to alter the notion of creating procedures and see how this new community of programmers reacted to it. Instead of editing one large text window, with multiple procedures making up a program, I used a model of teaching the Bricket new procedures one at a time. This builds on the tradition of teaching a LOGO turtle new words, a metaphor used in the early days of LOGO with an on-screen, or robotic, turtle. In BricketLogo, a user is only allowed to edit one procedure at a time, but is able to switch procedures via a menu, or utilize keyboard shortcuts *(figure 4.3)*. My hypothesis was that this would aid users in creating and editing programs. I also decided to refer to procedures as "commands". The breakdown is natural to the Bricket, since the Bricket already "knows" a number of commands (the built-in primitives), and working with an expansion module simply adds a set of

Figure 4.3 - The Bricket Command menu.

new commands. Thus the user is able to place themselves as the teacher of new commands to the Bricket, a pedagogical choice that establishs the roles of the learner and the tool (similar to the relationship between a learner and the LOGO turtle).

4.1.2 Accessibility

From an accessibility point of view, finding an appropriate screen reading solution presented my largest dilemma. After initially trying to build my own speaking interface, using Microsoft's Speech Development Kit (http://www.microsoft.com/speech/download/sdk51/), I turned to the Java Accessibility Bridge (JAB) and the JAWS screen reading software to handle screen reading for me. The integration of accessibility into the infrastructure of operating systems, as described previously, suggests that any developer attempting to build their own speakable interface is wasting their time. The decision to integrate accessibility at a low level of the system allows developers to make their applications accessible by simply hooking into the right places, and using the correct interface widgets. While that does force the developer to "play by the operating system's rules", it leads to a necessary level of standardization that benefits the visually impaired population°.

Java's Swing classes, used in conjunction with the JAB utility, provide a simple way to make accessible applications. While the underlying interface package (java.awt) does implement accessibility, the Swing classes standardize the implementation and are more comprehensive (the javax.swing package). Each Swing interface widget implements some interface of the javax.accessibility package, all based around the javax.accessibility.AccessibleContext class. This class provides methods to obtain the name, state, and content of any object that contains an instance of it. Thus by making a method call to any Swing component's setAccessibleDescription() method during initialization, a programmer can control how an interface widget is represented to a user via an assistive device, such as a screen reader. These assistive technologies then make successive calls to the object's getAccessibleContext() method to obtain that information and represent it in some understandable way to the user.

° Usability experts would argue that this level of standardization also aids those with sight.

Although the Microsoft Foundation Classes (MFC) do provide reliable accessibility through JAWS, choosing that technology would have restricted my options and slowed my progress. Primarily, I have no familiarity with their development environment, and thus development would have been delayed while I learned. Even though I knew that my study participants would be using Microsoft Windows and JAWS, I still thought it wise to consider the portability of Java as a good option to have. One of the main issues with assistive technologies is their high cost, so I thought it prudent to design for savings wherever I could. Thus, with only minor changes, one would be able to run BricketLogo with the free Linux operating system, providing that a suitably functioning screen reader or speech engine was present.

I found the most appropriate combination of technologies for this project to be a Java-based application tied to the JAWS screen reader on Microsoft Windows. From the Programmable Brick side, Java made sense due to the large body of code developed for the Cricket in Java. A compiler existed, as did serial communications code – both of which needed only minor modifications for the Bricket°. Thus the main task for me was to create an accessible interface. A Java Swing-based interface could use the Java Accessibility Bridge to work with JAWS. I also had the option of interfacing to Microsoft's Speech Software Development Kit to speak any other text I wished to. At a higher level, my familiarity with Java and the speed of prototyping it offers were not to be ignored.

4.1.3 Sound Interaction

Building on observations from the JavaSpeak project, I added features to indicate to the user the context of what they were editing. A user can select a menu item that will speak out the name of the command they are currently editing. Another feature lets the user find out what type of control structure they are editing (an if, repeat, or loop). These features serve not only as reminders, but also as a handy way to replace the act of scanning text. While earcons and auditory icons are engaging ideas, my research interests did not motivate me to use them.

4.1.4 Platform Considerations

° The main additon I made was to add support for strings.

While the pilot version of BricketLogo does rely on JAWS for speech, I have left in the earlier features that allowed it to be speakable via a free speech synthesis engine. These features take advantage of the Java Speech API (JSAPI) to interface to a number of available engines. My initial implementation relied on the free "JSAPI for SAPI" library by Jesse Peterson (http://www.jpeterson.com/rnd), which allows the developer to use JSAPI to work with Microsoft's free Speech API (SAPI). I initially attempted to rely completely on SAPI, hoping to avoid the cost of JAWS (around US$1000), but the speech engine would often crash while attempting to cancel events on the speech queue. Conversations online led me to believe there was a critical race condition somewhere in Java's implementation. I left the speaking features in place, with hopes that this would be fixed in the future. Having an application that does not rely on the expensive JAWS package would be preferable. While such a solution would hurt usability, in that I could not emulate the JAWS functionality fully, the cost savings makes it worth the steeper learning curve. This would also free me from my reliance on the Windows platform.

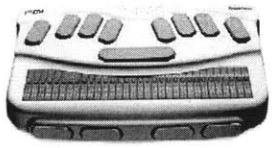


Figure 4.4 - The BrailleNote (http://www.braillenote.com).

A powerful new computational device, the BrailleNote, presents another appealing platform to develop an application for the visually impaired. Introduced by PulseData International in 2000, the BrailleNote is essentially a sub-notebook computer, running Microsoft Windows CE, with a Braille display and standard Braille keyboard for input *(figure 4.4)*. The device can also output speech and audio cues. Unfortunately, the cost of the BrailleNote was prohibitive for this study (around US$5000) and the anticipated time to learn how to use it would have limited the focus on the learning activity, for both my study participants and me. In the future, I imagine this to be a viable platform for BricketLogo, perhaps utilizing the Bricket's built in infrared communications port. It merits noting that Sun distributes a version of Java support for Windows CE through the Personal Java API (http://java.sun.com/products/personaljava/), so the existing code-base could be leveraged for this new platform.

4.2 Hardware

The latest Programmable Brick, the Cricket, is a programmable mini-

computer roughly the size of a 9V battery. It is built around a Microchip PIC processor. The Cricket can control two DC motors, using onboard motor-drivers, and take input from two resistive sensors, using onboard analog to digital conversion. Crickets can also communicate amongst themselves using two-way infrared communication. These basic functionalities can be extended via an expansion port. As mentioned earlier, peripherals (called bus-devices, or X-gadgets) can be created to speak a simple serial protocol and allow the Cricket to perform new tasks. Please refer to (Martin, et.al. 2000) for a through description of the Cricket architecture. The Bricket is an extended version of the Cricket.

4.2.1 The MetaCricket Designers Kit

The Bricket is example of the MetaCricket specification laid out in (Martin, et.al. 2000). The authors propose a model where "one can take an application built from the Cricket and several bus devices, and create a new design with this set of components laid out on a single board" (pg. 810). Thus at its core the Bricket is a Cricket, modified to suit the needs of a specific community.

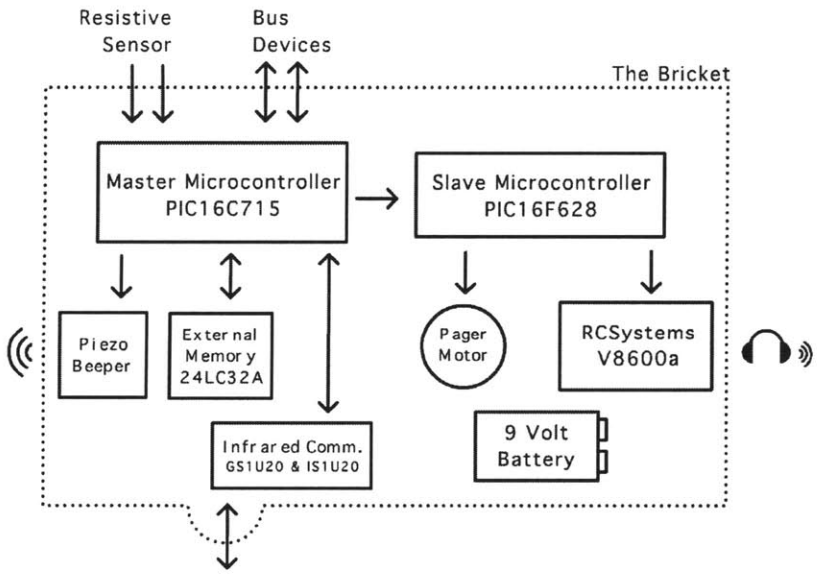


Figure 4.5 - A technical block diagram of the Bricket's main components.

The Bricket adds a slave microprocessor to the Cricket design *(figure 4.5)*. This chip manages the speech synthesis module and the pager motor.

While previous designs have adapted the Cricket for a particular activity or technology, the Bricket focuses on addressing the needs of a specific community. The "Thinking Tag" Cricket was initially created to facilitate face-to-face interaction in social settings (Borovoy, et.al. 1996). They were extended to perform participatory simulations, where students could be individual nodes in a group dynamics simulation, such as virus propogation (Colella, 1998). Both of these examples were targeted at specific activities for users of the modified Cricket. The Living LEGO City similarly created a Cricket car for use in a city-building activity with inner city kids in Pittsburgh (Rosenblatt and Mikhak, 2002). This activity encouraged reflection about the spaces the participants inhtabited, in addition to technical construction issues.

In adapting the MetaCricket idea to an existing community, my guiding principles were slightly different. In addition to designing for a set of underlying ideas to explore, my goal was to meet the needs of the visually impaired community. In my approach, there is a heightened sense of the required relevance, understanding, and ownership this community would need to adopt this tool for learning. In addition to the epistemological foundations discussed earlier, these principles guided my technology and activity development.

Redesigning the Cricket technological system to be accessible for the visually impaired involved numerous difficult decisions regarding the its functionalities and industrial design. Each feature was carefully evaluated on the basis of its usability and how it would fit into the learning goals explained previously. The following sections summarize the rationale behind those decisions.

4.2.2 Basic Functionalities

The Bricket adds some features to the Cricket, but also removes some of its basic functionalities. However, due to the modularity and flexibility provided by the bus-device system, a functionality was never truly removed. My decisions were primarily influenced by the desire to expand input and output modalities, as my activities were centered on those. Setting up the constraints of the hardware to focus on sensory input and

output was a conscious decision to guide the study participants' ideas towards this domain.

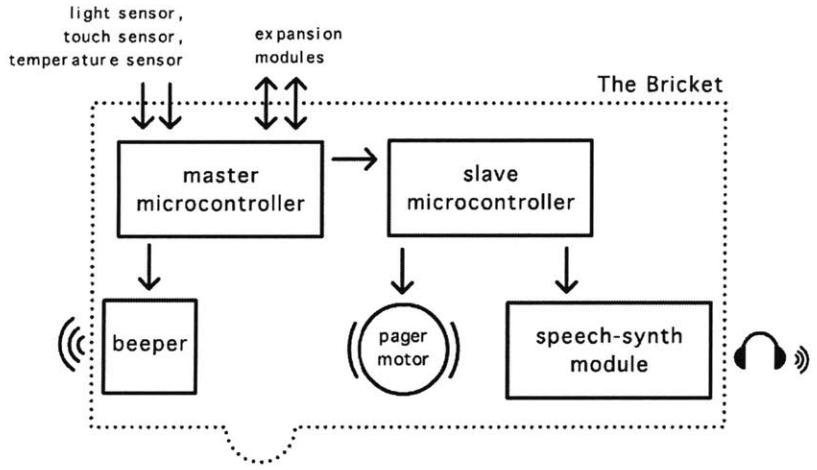


Figure 4.6 - The Bricket's input and output features.

A simple breakdown, organized by sensory channel and direction, describes the facilities available quite well *(figure 4.6)* :

Tactile – in – touch sensor, temperature sensor

Tactile – out – pager motor

Audio – out – speech synthesis, beeper, MIDI

Audio – in – voice recognition

Visual – in – light sensor

The existing Cricket system provides for all of the above except for the speech-synthesis, pager motor, and voice-recognition capabilities.

My initial plans called for making two types of Brickets - the speaking Bricket (Bricket II), and the silent Bricket (Bricket I). As the name indicates, the speaking Bricket would include speech-synthesis abilities, while the silent Bricket would not. I pursued this line due to the relatively high cost of the speech-synthesis component. However, as I discussed potential projects with my advisors, it became apparent that the speech facility was a necessary attribute. Thus the silent Bricket design was never completed. However, many of the lessons learned during its design are applicable to a more user-friendly version of the Cricket. The following section describes the audio and tactile output facilities in detail, as they governed the majority of my design decisions.

Audio Output

Speech synthesis is the primary output modality of the Bricket. Recent developments have made it possible for embedded systems to include speech synthesis capabilities. For this application, speech seemed to be an obvious choice to provide an expressive output channel. Empowering the user to programmatically generate any phrase to be spoken dramatically expanded the ability to interact with the Bricket itself.

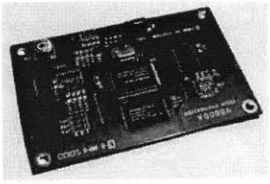


Figure 4.7 - The RCSystems V8600A

The Bricket generates speech via a RCSystems V8600A board *(figure 4.7)*, a high-level modular speech synthesis board (http://www.rcsys.com/v860x.htm). The Bricket's slave processor controls speech synthesis. Words are relayed to the V8600A serially, via ASCII text encoding. This high-level protocol specification made it quite easy to integrate, including the fact that it required the same 5V power supply as the rest of the Bricket components. The V8600A outputs an un-amplified mono audio signal that I connected to both channels of a standard stereo headphone port.

I chose to use the V8600a because of its low cost relative to the features it brings. Costing roughly US$130, it is affordable for embedded projects, even though it was by far the Bricket's most expensive component. For the quantities I was concerned with, I determined that it would have been costlier to develop my own circuit board design around the basic chips the V8600A utilizes (the RC8650 chipset). More importantly, it would have been a large time sink to develop and debug them. If I were to scale the quantities with the same technology, I would integrate the chipset into the main Bricket PCB design, allowing for a smaller overall industrial design.

Basic speech synthesis is controlled by a number of Logo commands. These include simple terms such as say, say-number, and shh°.

Tactile Output

Deciding that I wanted to focus on computational devices, rather than mobile robots, I removed the two motor drivers of the Cricket. Engaging in motor-involved activities would have pushed me to focus more on building LEGO brick fluency. Motors require gearing, and the LEGO gearing system facilitates creating gear-based mechanisms like no other.

° Please refer to the appendix for a complete reference to BricketLogo.

While many visually impaired children do have experience with LEGO bricks, due to their precise tactile design, I did not want to encounter that as a potential stumbling block with my specific participants. It can be argued that the capability should have been left in to support any ideas that children may have, but I would have been able to build and provide an external motor-driver bus-device quite quickly, if anyone had desired it.

Not wanting to lose the crucial tactile output capability, I added a small pager motor to the Bricket. Removing the motor drivers left the Bricket without any ability to exert a physical force on the user, a need that I found essential as a basic functionality. Motors vibrate, and controlling that vibration can allow for a fairly high-resolution tactile output.

The small pager motor I integrated into the Bricket returned this tactile output to the user. The Bricket's secondary processor drives the pager motor. Using the PIC16F628's built in pulse-width modulation, I hoped to be able to provide for at least a few levels of vibration. Unfortunately, I was unable to find a pager motor that performed well outside of the on-off realm. Left with no appealing options, I determined that this binary functionality would suffice. The pager motor is controlled via two simple Logo commands - motor-on and motor-off.

4.2.3 Extended Functionalities

The versatile bus-device system of the Cricket allowed me to extend the Bricket's abilities through add-on modules, each with their own specific functionalities. In addition to using existing devices, I developed a Voice Recognition module in co-operation with Casey Smith (another graduate student in the Lifelong Kindergarten Group). What follows is a brief description of each of the bus-devices used.

Voice Recognition

The voice-recognition bus-device, created for this thesis, allows the user to recognize up to 15 words. The board is first trained with the words to recognize by pressing the appropriate on-board buttons and repeating a word twice. The words are assigned numbers based on their order of

entry. The Bricket can then query the bus-device to find out the number of the last phrase recognized.

While voice-recognition does fail to recognize quite often, the failure mode is fairly predictable. It is based around the Voice Direct 364 chipset (from Sensory, Inc.) (http://www.sensoryinc.com/html/products/voicedirect364.html). Although the vendor claims it is speaker-dependent, with 99% accuracy, we found it to be rather speaker-independent, with a much lower rate of accuracy. However, the misses could be avoided by choosing a better set of words (with less similarities between them).

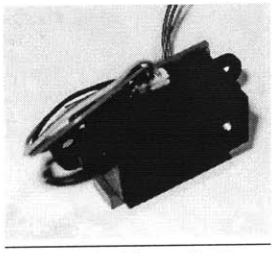


Figure 4.9 - The optical distance sensor bus device.

Optical Distance Sensor

The existing optical distance sensor allows the user to read the distance to an object and operate on it like any other sensor value *(figure 4.9)*. It reports a number between 0 (far away) and 255 (close). Utilizing the GP2D02, from Sharp Electronics, it can detect distances between roughly 10 and 80 centimeters.

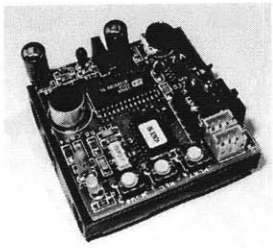


Figure 4.10 - The voice recorder bus device.

Voice Recorder

The existing voice recorder bus-device lets the user record a series of messages (up to a minute, in total) and then play them back *(figure 4.10)*. Recording is accomplished using buttons on the device. Each message is given an index according to the series they are recorded in. The Bricket can send the voice recorder an index of a recording to be played back (through a speaker plugged in to the bus-device). The device is built around the ISD2560S, from Winbond Electronics.

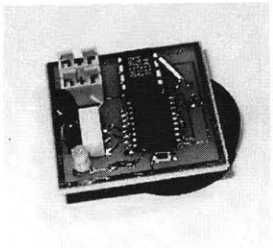


Figure 4.11 - The real-time clock bus device.

Real-Time Clock

The real-time clock bus-device lets the user keep track of the actual date and time *(figure 4.11)*. The Bricket can query the device to determine the hour, minute, second, day, week, month, year, or day-of-week. The board has its own power supply, so after the date is set once from the Bricket, it remembers the correct date and time even when not plugged in. Casey Smith developed the real time clock.

4.2.4 Industrial Design

The creation of any object for the visually impaired requires careful consideration of industrial design issues. Due to my limited experience in the area, I decided to work with an industrial design student to develop a case. Not only would the project provide a good experience for an undergraduate design student, but I also felt that I had a great deal to learn from someone with that background. I was lucky enough to engage an energized student named Jeff Easter, from Carnegie Mellon University's design school. We began by evaluating the current design of the Cricket, then iterating through a number of physical designs to establish a criteria for the redesign.

The Design of the Cricket

The Cricket, in its current form, was in no way suited for working with the visually impaired. The two main issues were the size and the various connection plugs. The Cricket was optimized for size, focusing on the goal of empowering users to embed computation in everyday objects. Because of a visually impaired user's reliance on solely tactile means to connect modules, this small size did not make sense for the Bricket. This is not to say I wanted a backpack sized object, but I felt that it needed to be comfortable to manipulate with two hands.

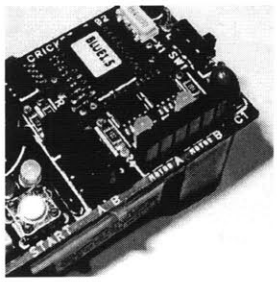


Figure 4.12 - The Cricket's sensor and motor plugs.

Rethinking the Cricket's connections to the outside world was a much more complex issue. I was able to identify two main problems – shape and feedback. The shape of the sensor and motor plugs are almost identical *(figure 4.12)*. Sighted users often plug sensors into motor plugs, and motors into sensor plugs. In addition to this, the plugs for the two sensors and the two motors are grouped quite close to one another. This leads to many users plugging a sensor or motor slightly off-center from where it should be – a problem that requires close visual inspection to debug. This leads to the other main issue, which is the fact that there is no non-visual feedback to let the user know when they have correctly plugged something in. It should be noted that these points are not true of the bus-device connector, which only be inserted in one orientation *(figure 4.13)*.

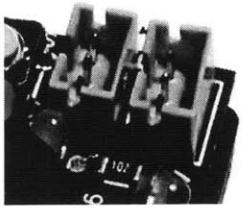


Figure 4.13 - The Cricket bus connector.

Criteria and Methodology for a Redesign

The previously described evaluation of the Cricket's design led to a basic criterion for an accessible redesign. We would consider a few main issues to guide our attempts. First, the case would have to clearly indicate where connections go in some tactile manner. Second, any interactive elements had to be keyed and provide some feedback. Third, the overall size would have to allow for two-handed operation with ease.

At this point I had decided which functional capabilities to include in the Bricket, and had completed an initial PCB design *(figure 4.14)*. From this we were able to generate an exact list of the elements that needed to integrated into the case:

- two sensor plugs
- two bus-device plugs
- a headphone jack
- a 2-position on/off switch
- a toggle "run" switch
- a window for infrared communications

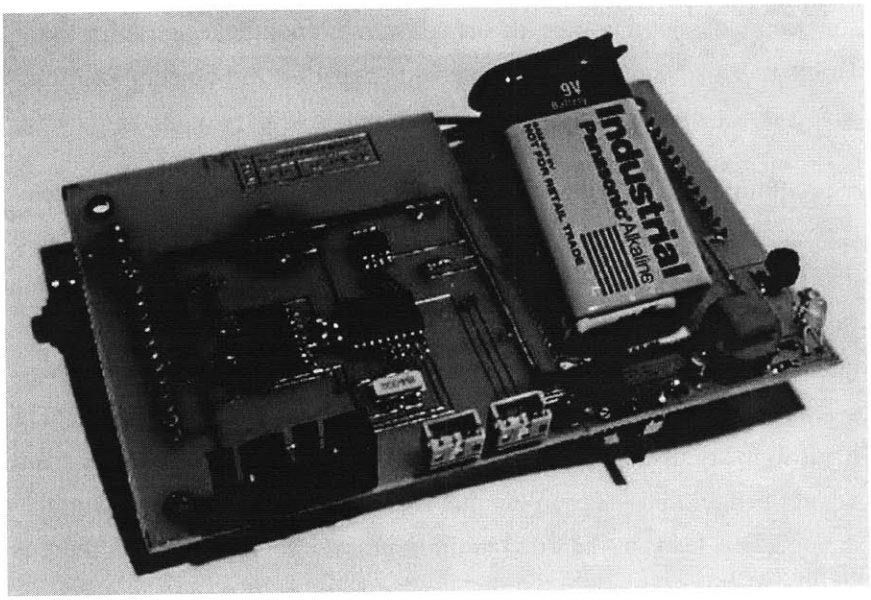


Figure 4.14 - The Bricket PCB.

The Bricket v1.0

The final design we decided to use with the initial trial group was built on the criteria defined above. Jeff created the design using LEGO bricks

(figure 4.15), which were then placed into a vacuum former to create a plastic case. Extensive effort was then spent in cutting away the excess plastic and carving out the appropriate holes for sockets and switches *(figure 4.16)*. The second layer of the design was molded in a similar way, and then glued over top of the base mold. The colors of the two molds were picked to be high in contrast. Jeff designed and created the prototypes and manufactured the final design.

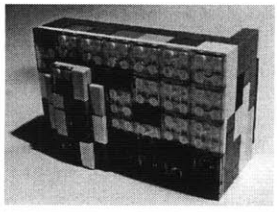


Figure 4.15 - The mold for the Bricket case, made of LEGO bricks.

The interactive elements were distributed to facilitate usability. All inputs to the Bricket are placed along the "top" side *(figure 4.17)*. We found that this made it easier to hold the Bricket with one hand and plug in with the other. The headphone plug, being the only output plug on the Bricket, was placed on the short end in order to separated from the others. We found that this reduced the incidence of the headphone cord coming in the way of inserting and removing plugs. The infrared communications window was similarly segregated from the other elements and covered with a rounded window, to suggest its function. A secondary molded layer was placed atop the primary mold to indicate the location of the various plugs. The user's hand naturally follows the embossed bumps around the edge from the sensors to the headphone plug.

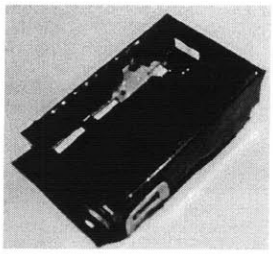


Figure 4.16 - The inside of a completed Bricket case.

The connectors for the sensors were redesigned to indicate when they were placed correctly. We tried numerous keyed connectors, where the male plug latches into the female only in one direction, and eventually decided on a commercially available design, which also required a small latch to be pressed to remove it *(figure 4.18)*. The bus connectors, while they are keyed, do not provide explicit tactile feedback upon proper insertion. However, we found users were able to discern when they were correctly inserted by attempting to pull it out – if it came out easily then it was not correctly placed. This design seemed to suffice and saved us from retrofitting the existing bus devices with new connections.

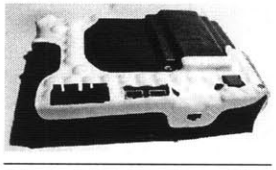


Figure 4.17 - The top of the completed Bricket .

Increasing the overall size turned out to be a moot point, due to the fact that the speech-synthesis module dictated it. We ended up with the "sandwich" design *(figure 4.19)* due to this constraint. If I had decided to use the RC8650 chipset instead of the V8600A development module,

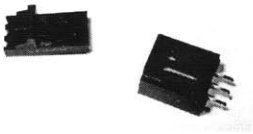


Figure 4.18 - The Bricket's sensor connector.

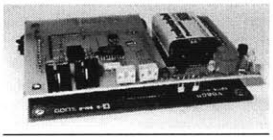


Figure 4.19 - The internals of the Bricket. Top upper PCB is the Bricket, the bottom is the V8600A.

we would have had more freedom with the overall size. However, the benefits of using the V8600A were far greater, as discussed earlier.

The battery was nestled inside of the case, in a notch carved out of the main component PCB. The back of the case was covered with a LEGO base plate, which slid into place from the short side of the Bricket. The LEGO studs were exposed, leaving an easy way to attach sensors and bus-devices, if desired.

Bus-Devices

Jeff's initial prototypes created a design language followed up on to create cases for several bus-devices. The following is a sample of cases for various bus-devices built from the criteria established previously:

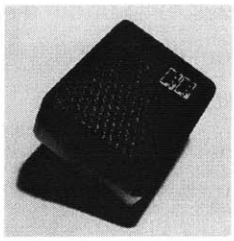


Figure 4.20 - A case for the real-time clock bus-device.

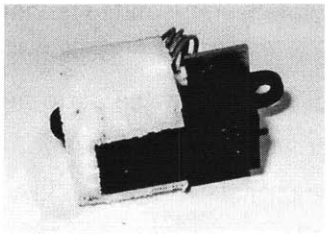


Figure 4.21 - Half of a case for the optical distance sensor.

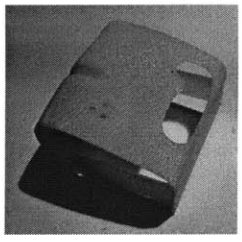


Figure 4.22 - A case for the voice recorder bus-device.

Chapter 5 - Activity Design

I designed and carried out a small study to test out my technology and activity ideas. Based on my own interpretations of the theoretical foundations explained earlier, I decided to do my study as a series of themed sessions with individual children in their home. The participants would keep the Bricket between sessions, hopefully continuing to build and explore. I planned to have them get together for one or two larger gatherings to compare creations and build a small interacting community. My study was carried out over 5 sessions of 3-4 hours each. This totaled to roughly 15 to 20 hours of time with each study participant. While I designed the study for a group of 6 learners, logistics limited my study to 3 participants°.

The introductory activities were designed to explore various technologies and approaches. The goal was not only to familiarize the user with the Bricket system and its basic functionalities, but also to convey some underlying ideas that would be returned to later. These focused sessions then led to a more open discussion of a final project, which I hoped would build on some of the ideas and technologies introduced. This approach allowed me to gauge the individual's interests in various aspects of the system, and then facilitate the creation of some construct along one of those axes of interest. The following is a description of the plans I had for these introductory sessions. They were, of course, carried out with some significant amount of variation, but the underlying theme

* Please refer to the discussion of logistics in section 7.1.1 for further information.

remained constant.

5.1 Session 1: Saying Hello

The initial Bricket activity was designed to immediately suggest assistive uses for the technology, but at the same time honestly expose some of its limitations. We began with the interactive Command Center area of the interface, where any command typed can be immediately executed on the Bricket. Thus if we typed `say "hello`, the Bricket would speak the word "hello" with its speech synthesis module. The syntax for this command is simple and the effect both immediate and obvious. We would then plug in the distance sensor module and use a command such as `say-number distance` to have the Bricket speak the distance it read. At this point we had a distance-sensing device that was useful and relevant to the visually impaired community, and simple to create.

From there my plan involved exploring procedures, touch-sensor input, and thresholding. I would introduce the notion of teaching the Bricket new commands to repeat the distance over and over. We then would use a touch switch and the `if` construct to have the Bricket speak the distance only when the switch was pressed. Inevitably, we would arrive at the point of wanting to have the Bricket say something when an object was close in front of it, leading to the idea of thresholding. Through these initial creations, the unreliability of the distance sensor would become obvious, an issue I planned to explore more fully at another time. I would end the session by leaving the user with the goal of getting more comfortable with the BricketLogo language syntax by creating other simple programs around the `say` command.

5.2 Session 2: International Bricket Standard Time

The second activity was designed to introduce the idea of having the Bricket on and around for prolonged periods of time, and to use basic constructs not discussed in the first session. These included the `loop` and `repeat` commands, and the idea of recursion. This focused around using the real-time clock module. One of my goals was to suggest the idea that the Bricket was something that could be interacted with over a long

period of time. It is something that exists as part of the child's world, rather than a construct that only exists at certain times when I was there to play with them. This activity was also driven by the desire to create a constraint that pushed them to work with the Bricket outside of the times I was there. While this is a bit of subterfuge, the logistics demanded it. The overall goal was to try and cultivate an attitude towards the Bricket like that towards a toy (or more appropriately, like the tools I described previously).

Using the clock involved the introduction of a number of new commands to control the retrieval of the date, time, year, etc. While I was worried that this would confuse the issue by presenting even more "magic" words to remember, my hypothesis was that the functionality of the clock would be meaningful and thus was willing to handle any confusion raised by the new commands.

5.3 Session 3: Reinventing the Wheel

After two sessions I hoped that a sufficient level of familiarity with the programming language and environment would be established, and I could then engage issues of sensing the world. Valuing what the learner brings to the experience, I looked to the existing tools of the community to ground this activity. Thus I wrapped this exploration in the theme of recreating assistive devices that the participants already used. I had a number of ideas to suggest, some based on available commercial assistive technologies°. My examples included:

- a device that alerts the user when a glass is full
- a distance sensing cane
- a device to leave voice message for others
- a speaking weather station

The final two are based on prototypes I created as example projects when first developing the Bricket. I would present these to the learner and solicit other ideas.

The underlying goal of this session was to begin to engage the participants in a discussion about how they felt about the design of the tools they use every day. I wanted this discussion to lead to reflections on

° This list was generated in the spirit of the "Twenty Things to Do with a Programmable Brick" (Resnick, 1993), which was itself inspired by "Ten Things to Do with a Better Computer" (Hillis, 1975), which was inspired by the original "Twenty Interesting Things to Do with a Computer" (Papert and Soloman, 1971).

their own designs in comparison to existing products.

5.4 Final Projects

Building on these three sessions, I then planned to have each participant build a more involved project, following up on a personal interest or idea. I planned to be an active partner in this decision process to guide them to a project that I thought would be compelling from a learning point of view, but would also be fun. I would take the opportunity to introduce new technological capabilities (through more expansion modules) to the Bricket system in order to test out their accessibility and relevance. I also hoped to have the development of new devices driven by the children's interests.

Chapter 6 - Case Studies

This chapter presents three stories of objects created by participants in my study°. These case studies document the creation of objects that reflect the participants' exploration of ideas. Psuedonyms are used to protect the participants' identities.

My metric for success is how much I can demonstrate that the participants have explored the topics of interest I laid out earlier. Thus these stories focus on the ideas of handling and interpreting unreliable sensor data, manipulating sensory input data to make intuitive and meaningful representations, and creating and managing large programming structures to develop a highly interactive device. I demonstrate the learners' understanding of these concepts by using direct quotes and by discussing the objects they made. By describing the artifacts they have built, and the process of building, I showcase the physical embodiment of the ideas they explored. This can be seen as a "constructionist" approach to evaluating the learning activities, due to its focus on their creations and how they are talked about as being inherently reflective of the participants' thinking. I also focus on how their thinking changed during the course of the study to demonstrate their own interpretations of the concepts.

Each case begins with a bit of background on the participant and how some of their early creations established their styles of learning and

° This study was fully approved by the MIT Committee on the Use of Humans as Experimental Subjects (application No. 2812).

areas of interest. The major portion is a discussion of their final project. This focuses on how the process of building led them to interact with the ideas underlying their creations. The first object is a digital cane, able to sense and warn its user when an obstacle is in their path. The second object is a "tricorder", capable of sensing a myriad of things about the real world and representing them to the user. The third object is an interactive toy, programmed with a personality that interacts with its user.

6.1 Case 1 : The Digital Cane

Jim is a 14 year-old boy. Jim was born completely blind.

From the beginning of this study, Jim said he wanted to "invent something new". He was excited to be able to create things that he couldn't have before. I attribute this in part to his father, who is an engineer by trade. It appeared to me that Jim's enjoyment of building things had been rubbed off from his father. It turned out that they would often work on Bricket projects together, establishing a pattern of play that will likely last beyond this study's completion.

Jim decided that he liked the idea of redesigning the cane, his most important assistive device. A cane is a device that assists the visually impaired by extending their reach to help avoid objects in their path. The cane is introduced to some visually impaired children as soon as they begin to walk, and rapidly becomes their most necessary assistive device. Educators for the visually impaired who focus on mobility training speak to the importance of the cane by noting how its early use establishes an attitude of independence, rather than one of reliance. The tapping and sweeping from side to side becomes almost a subconscious part of walking. It can replace a small part of the scanning aspect of vision, providing an extended tactile modality to discover objects at a close distance. Using a cane also frees the sense of hearing to focus on landmark location, instead of local events.

Jim's first idea for his digital cane was to simply duplicate a project we did in the initial session, where we programmed the Bricket to speak out when the distance sensor found a close object. After I created a longer

bus-device connection cable, we were able to mount the distance sensor three-quarters of the way down one of his canes. We ran the wire up the cane (attaching it simply with tape) and plugged it into the Bricket, which he was wearing on his belt with attachable belt-clip.

We soon found that the simple program for thresholding we had used earlier proved unreliable for actual navigation. The distance sensor returned both false positives and false negatives quite often. This dilemma set up the major focus of Jim's digital cane - strategies to deal with inaccurate sensor data. This, in fact, led us to a major area of study in robotics, known as sensor fusion.

I introduced Jim to the idea of sensor fusion by comparing it to how he uses his own senses together. For instance, he sometimes uses both his cane and his hands to help guide him down a corridor he is unfamiliar with. At its most basic level, sensor fusion suggested a number of techniques for dealing with erratic sensor data. The most immediately relevant seemed to be the use of complementary sensors, redundant sensors, and computation algorithms. Jim and I drew on these in our attempts to discover a good way to get more reliable data out of the sensors.

The use of complementary sensors allows one to check a sensor reading against another reading of the same quantity with a different type of sensor. For Jim's project, this called for the use of another sensor that would tell us the distance to an object. Unfortunately, a sonar module wasn't available, and a laser range finder proved to be too expensive (for the resolution we required). We decided that for the digital cane, the complementary sensor could be the cane itself. If the distance sensor reported an object just outside of Jim's reach, he could lean forward and discover if anything was actually there by using his cane. Thus he had a complimentary check to verify the reporting of close objects.

Another idea, redundancy, suggests duplicating the uncooperative sensor, in hopes that two are better than one. At first, this seemed silly to Jim, who presented the reasonable argument that "if one doesn't work good, why would two?" His theory was that "if you have one sensor giving me bad distances, and add another, I'll just have two sensors

giving me bad distances". To find out if his inclination was correct, we designed a test. We created a construction that, when actuated by a touch switch, speaks out the distance returned by two distance sensors mounted just above one-another. We would point at objects that we knew the distance to, and count how many times each sensor reported a bad number. As it turned out, it was almost never the case that both sensors were wrong. This was enough to convince Jim that he should try using two, because often the two values were close to each other and an average could thus be relied on to eliminate most bad readings.

This kind of mini-experimentation was a marked departure from the earlier way Jim had been building. He often decided on one way of doing things, and then simply did that way until it worked, or until it didn't. Jim liked attempting to implement his whole idea at once, rather than creating a small part, trying it out, and trying it again. The iterative approach we were taking to make the digital cane was influencing the way he approached problems.

Programming simple algorithms was the main method we used to process the sensor data. One of the behaviors that Jim noticed in our little test was that "bad stuff comes out of nowhere" – that bad readings are often unpredictable and unrepeatable. He wondered why it "can't just remember what it saw before". That being the spark, we created a simple averaging algorithm to remember the last ten sensor readings, and use an average of them to monitor for objects in his path. This didn't seem to work, because it would take a noticeable amount of time for the average to purge itself of clear readings when something new entered in front of it. After changing it to average only three samples, it worked quite a bit better. Used in conjunction with an average of two distance sensors mounted in the same direction, this proved quite effective at reporting accurate distance readings.

Comparing his digital cane to other ones that are commercially available, Jim was quite excited by his own design. He was able to tailor its design and customize its features. In fact, he changed the behavior to speak the distance every few seconds, rather than only speaking when an object was near. He found this was "more useful", and felt more

"like using a regular cane". Regarding existing commercial enhanced canes, Jim said

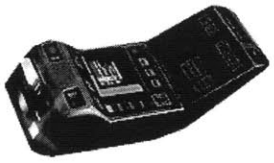
> They're too heavy – I tried one once and it felt so much heavier than my regular cane. With this I can have the whole thing on my belt. That way I don't feel the weight in my hand.

Despite the cord connecting the cane to the Bricket on his belt, Jim still liked his own design better, for an established criterion. This critique of commercial devices shows an understanding of the design decisions that went into their construction. In creating his own digital cane, Jim established metrics for the design that he was able to apply to similar constructions.

6.2 Case 2: The Tricorder

Carl is a 12 year old boy. Carl has severe visual impairment, and can only make out the difference between light and dark areas.

Throughout the introductory sessions, Carl was fascinated by trying out the various basic sensors, and walked around listening to their readings many times. He expressed disappointment that he could only plug in two at a time (even though I had provided three types of sensors)°. Rather than wanting to record data, he wanted to represent an instantaneous reading back to himself through one of the Bricket's outputs. He liked plugging things together and just seeing what happened, a style that played well with the Programmable Brick's focus on interactive, iterative design. Carl used the Command Center part of the interface (which allows a user to type a command and run it immediately) quite a bit. He would have the Bricket speak whatever it read on the sensor, then plug in a different sensor and try again.



° Characters on Star Trek use the tricorder to gather data about the environment around them.

This provoked me to ask him if he had even heard of the "tricorder" from the Star Trek television show. The name is a bit of a misnomer, in that it records more than three types of data. A tricorder is best described as an all-around sensing unit°. Characters on Star Trek carry it with them, using it to measure almost anything imaginable (and many things unimaginable). The wearable nature of the Bricket, and Carl's

° A sensor expansion bus-device does exist, but retrofitting it to work with the Bricket's connectors would have required a new circuit board design. Moreover, from a pedagogical standpoint, it is often fruitful to leave constraints as they are, in order to breed creative solutions.

desire to sense everything in the world, led me to suggest he make a tricorder. Here my own biases came into play, as I have wanted to build one since I was roughly 12 years old. However, Carl's enthusiasm for the idea ensured that it would be a project that both he and I would be excited about.

Creating the tricorder brought up many issues of how to accurately represent data in intuitive ways. While there are numerous inputs (touch, distance, light, temperature) and outputs (pager motor, speech synthesis, beeping), there are no obvious mappings to be made between them. This becomes even more complex when one realizes that two of the outputs use the same sensory modality - sound. Managing to represent the myriad of sensors into two modes of sensory output (auditory and tactile) required reflection and manipulation of the data.

Carl's initial explorations of representing the sensors focused on literal mappings, but that strategy did not work as he progressed to more sensors. Comments like "I want the buzzer to be the light" led to one-to-one mappings. His first attempts made use of statements like note sensora 5, which would play a note of varying pitch corresponding to the amount of light sensed. This directly mapped input to output, without any adjustments in-between. These strategies had to be modified when he moved to using the motor command, which is binary. Carl expected the motor to work the same way, but was disappointed to discover that there are two motor commands, motor-on and motor-off, rather than the opportunity to set the power of the motor. I explained to him that the motor is "binary" - that it can only turn on or off (much like the switcha and switchb commands). Thus he would have to try another method to use it. This led him back to our first exercise, in which we used a threshold to have the distance sensor tell him when an object was close.

These technological limitations led Carl to begin performing permutations on sensor data to format them for the specific output modality he wanted. The thresholding case, just discussed, is the most basic example. He initially mapped the light sensor to the pager-motor, but decided that the distance sensor made more sense as an input for a tactile output channel. He said he made the change because he wanted "it to be like

when I run into a wall", so he could feel the distance just like he would by reaching out his hand. This was the first intuitive mapping he made, and appeared to change his mindset towards the others.

After a few attempts, Carl ended up being most happy with the light sensor being mapped to the beeper. However, instead of the one-to-one mapping he had initially programmed, he created a series of five ranges that played different notes. If the sensor reported a value between 0 and 50, it would play a fairly low tone; between 50 and 100, it would play a slightly higher pitched tone; and so on. This represented a shift in his thinking about what the sensors were – they became continuous elements that he could discretize to pick out areas that were important to him. Carl's final mapping played out this new approach. He had two thresholds on the temperature sensor to have it say when something was hot or cold. His manipulations of the sensor data left him with the idea that the "numbers can be played with."

Here it can be seen that the Bricket led Carl to some of the same concepts that are explored in the area of multimodal output synthesis, and multimodal data visualization. The human-computer interaction field has been investigating this area for years, attempting to come up with frameworks for analysis and heuristics for implementations (as in Andre, 1993). Carl's ideas about representation shared a common base with many of these researchers' approaches. He found intuitive mappings between inputs and outputs in different modalities, and was able to understand and act on multiple outputs that were received at the same time.

6.3 Case 3 : BricketBot

Lisa is a 14 year old girl. Like Carl, her visual impairment is quite severe, in that she can only distinguish between large patches of light and dark.

From the very beginning, Lisa was fascinated by the interactive nature of the Bricket. She enjoyed making it talk, and having a conversation with it. Among other things, this led Lisa to usie speech almost exclusively for all her output. She said that it was "nicer to talk" than

the other options. Noticing this in the first session, I told her about Jim's first project, in which he acted out a play with the Bricket. He would say a line, and then hit a switch, triggering the Bricket to say an appropriate line back.

When thinking about her final project, Lisa returned to her interest in interactivity, and decided to make an interactive assistant. My own existing interests in creating robotic characters were influential here, as I became quite excited when she brought up the idea. We eventually decided on naming it "BricketBot".° Lisa wanted BricketBot to :

- "wake me up in the morning"
- "tell me what the weather is outside"
- "sing a song with me"
- "tell me a story"

With these tasks in mind, we set about creating the individual skills that BricketBot would need.

The various skills drew well on projects we had done earlier. The wake-up call was something Lisa had already created, and liked. That was integrated without many changes. Telling a story was quite simple, but required a good deal of coding to speak just a short story. Singing a song was difficult, in that it never sounded right to Lisa (mainly due to the fact that we could not control the Bricket's pitch sufficiently*). Her idea to have it tell her the weather borrowed an idea from my earlier suggestion of making a "weather station". This was accomplished simply by extending a sensor cable to tape a temperature sensor to her window.

During the creation of BricketBot, we encountered many of the underlying ideas behind creating interactive characters. The first time I added a hello message to BricketBot, Lisa reacted that "she wouldn't say it that way!" – Lisa had a clear picture in her mind of the type of person BricketBot was (beyond the fact that it was female). Anything outside of that image would ruin the interaction for her. This idea of consistency is one of the cores of creating animated characters (virtual or not). Animated characters must be consistent in their actions, or they risk destroying the tenuous relationship with the audience. Similarly, Lisa insisted that

BricketBot had to "act like BricketBot".

Integrating the features Lisa wanted into one construction proved to be technically challenging. Her initial plan was to have a number of touch sensors, each of which would trigger a different skill. This, of course, was limited by the Bricket's limitation to two sensor plugs. However, since we were using the clock bus-device already, for the wake-up call, I suggested we use that again somehow. Lisa liked the idea and proceeded to create an input handler that, when triggered, checked the time and performed different tasks based on what part of the day it was. This turned out to be an inter-related series of commands, all triggered by a when statement. The when acts like an interrupt routine – when some condition becomes true, it stops execution of the program to begin running code to handle that condition. During morning hours, Bricket-Bot would tell Lisa what the temperature was outside, or say a simple hello message (depending on which touch sensor was triggered). In the evening, BricketBot would either tell a short story (about itself), or sing a song (from the band "Destiny's Child").

This level of control of execution flow demonstrated a high level of programming proficiency. Lisa quickly grasped the importance of modularity in her coding style, and was able to reuse some of her code from previous projects to make BricketBot. Her final program had 15 commands°, more than the sum of the other two projects presented here. She realized that this required her to have concise and accurate command names, and limit the commands so no two did the same thing. Managing this level of code modularity proved a bit difficult, and Lisa complained that sometimes she would forget "who commanded who to do what". This suggests that some kind of high-level flow-of-execution representation would be useful for more extended projects.

Lisa was able to create something that was personally meaningful, tailored just for her. As with Jim's cane, this level of customization is not possible with existing assistive devices.

Chapter 7 - Reflections

The projects presented in the previous chapter paint a fair picture of the overall projects created during my small study. The foci of the various projects ranged from more technical programming issues, to questions of how to represent information, to theoretical issues of how to create good interactions. My exploratory approach left ample space for the participants to find their own path through the ideas I presented. For instance, while programming fascinated Lisa, Carl often found it exasperating.

The activities and technologies I developed in conjunction with these learners demonstrate that the visually impaired community is able to create their own assistive devices. These children I worked with were familiar with computers and screen-reading software, but did not have programming experience. They began with a tool that they used primarily for communicating, and quickly made it a tool for creating and building.

7.1 Problems that Arose

However, we did have many problems along the way. What follows is a summary of issues that arose around the topics of logistics, software, and hardware.

7.1.1 Logistics

Locating a suitable group of study participants proved quite difficult. I was ready to begin my study in September of 2001, but was unable to begin until January of 2002. I initially contacted the Perkins School, just outside of Boston, but my desire to conduct a study in a more informal setting led me away from collaborating with them. Also, many of the their students have multiple disabilities, which raised difficult issues for conducting a study with the technology I had developed. Through the help of the National Braille Press (headquartered in Boston), I was able to contact some teachers that work with visually impaired students in public schools. Unfortunately, the scheduling with the students they suggested contacting did no work out. It turned out that someone I had contacted earlier, whom I discovered online through a company that makes assistive devices for the visually impaired, was able to put me in touch with a few interested families that they knew personally.

The decision to work outside of any formal, pre-organized setting made it logistically quite difficult to conduct my study. School's regularity provides a structure within which I could have guaranteed that my participants would have been available at specific hours every week. Additionally, there are numerous weekend support and training activities planned for visually impaired students, especially those that are in public schools. These activities often made it difficult for me to schedule time with the participants. Many times I would have plans for a weekend up until the day before, when I would receive a call canceling because some other event had been scheduled. This difficulty also prohibited me from getting all the participants together for group discussions.

7.1.2 Software

For the participants of this study, learning the text-based BricketLogo programming language was an arduous undertaking. I had initially planned to provide a few sample programs, and a printed Braille reference. However, the sample programs proved inadequate and the Braille reference material was difficult to use. I responded to this by a help-center into a subsequent revision of the BricketLogo application. This

help-center allowed the user to get short descriptions of each command by navigating through a hierarchy of drop-down menus. I also installed a larger and more relevant set of example programs that the users could open and inspect for ideas. While these strategies did resolve some of the issues, they did not take away the fact that it takes a bit of time to get used to programming in a text language.

Tangible Programming

A number of researchers have proposed tangible programming environments, where attaching physical objects to each other in some way creates a program. These physical objects represent commands in a language that, when executed, controls some device or creates some behavior (Suzuki and Kato, 1993). Building on this work, a collection of tangible programming bricks was developed to control the Cricket (McNerney, 2000). These were small LEGO bricks, each representing a command, which could be stacked to create a program. This program could then be downloaded to a Cricket. McNerney suggests one application to be a programming interface for the visually impaired (pg 62), but did not pursue this goal.

The idea of a tangible programming environment, while it is appealing to the visually impaired audience for many activities, did not suit the plan I developed for a number of reasons. Applications for tangible programming usually require a very limited vocabulary and a small set of control structures. As soon as either of those grows, creating programs with the tangible bricks would become overly cumbersome. However, I do see tangible bricks as an excellent entry point for beginning programmers. Unfortunately, I think that the ceiling of what one could create would be reached rapidly. In fact, creating a high ceiling was not one of McNerney's goals – he focused on reducing the age barrier to creating with Programmable Bricks, increasing communication while programming, and creating novel interfaces for household devices (McNerney, 2000). Since I would have had my study participants quickly move to text-based programming, I decided that the development effort required to work with the tangible bricks was not worth the gains they might have provided.

Accessibility

The primary dilemma in making BricketLogo an accessible application was guaranteeing 100% speakability. If any interface item was inaccessible to the screen reader, it was effectively non-existence. If my speech generation engine at any time crashed, the entire application would be useless. These factors pushed me to rly on JAWS and the Java Accessibility Bridge for speaking my interface. These proven technologies relieved me from having to develop a robust solution, which would have required large amounts of time and testing.

At the same time, relying on costly technologies is still unacceptable if other solutions exist. The release of FreeTTS, a Java-based open-source speech synthesis solution, presented another option (http://freetts.sourceforce.net/). The implementers were able to remedy the previously mentioned low-level race-condition in the JSAPI, allowing me to solve the bug that would cause the speech engine to crash. I created an initial package of BricketLogo using the FreeTTS engine, and experienced a high level of reliability. Further testing would be needed to ensure full functionality, but FreeTTS seems to be a better solution for the level of speech control that I required. In addition, packaging the speech with the application would simplify install procedures (which sometimes took up to an hour to debug).

7.1.3 Hardware and Industrial Design

The frequent consultations with my advisors during the development of the Bricket hardware proved quite useful, in that there were no major unanticipated hurdles to using the Bricket. That is not to say that it was perfectly designed, by any means, but rather that the problems were ones that could be managed.

On the whole, the industrial design decisions proved to be quite user-friendly to the visually impaired. The sensor plugs were suited to the children's smaller hands, and the Bricket case reached usability goals. The earlier concern in regards to a size well suited to two-handed operation turned out to be misplaced, as most of the users plugged things in while the Bricket was on a surface of some sort. In fact, users would

take advantage of a completely overlooked detail – the fact that all the components connected with wires. The wires proved invaluable, in that the users were able to trace them with their hands. This greatly assisted the process of explaining and discussing the various components we plugged in. If there had been some kind of wireless connection, for example, it would have been quite difficult to explain the interactions between the modules in the systems they designed. This problem was encountered often when ensuring the Bricket had line-of-sight with the interface while downloading.

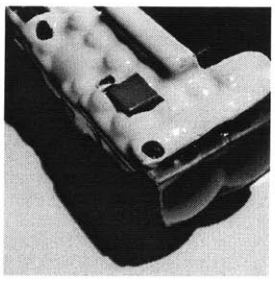


Figure 7.1 - The Bricket's run button proved to be hard to use.

The most significant problem was the :run" button, which starts a program on the Bricket (figure 7.1). The run button on the Cricket is a simple push-button, used to start of stop a program, depending on the state of the Cricket. This is a troublesome point because there are two ways that can happen – the programs can simple finish executing, or a use can stop it mid-program. I anticipated this being problematic, as there was only a visual indication to let the user know if the Crickets was ruining a program or it had stopped. I decided to add audio feed-back indicating when a program had started and when it had finished. These changes, however, proved inefficient to surmount the interface problem. The users more often just turned off the Bricket to ensure that the program had stopped running. An ideal solution to this problem would be an actuated switch, who's state could be toggled by either the user or the Bricket's program upon termination. Unfortunately, I was unable to locate one with an appropriate size and power requirement.

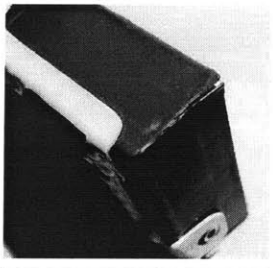


Figure 7.2 - The Bricket case began to crack over time.

Another serious problem occurred with the battery. The design did not provide an easy means to pull out and replace the batteries. First the back cover had to be removed, then the speech synthesis board had to be pulled apart from the main component board. Only after that could the battery be replaced. This is difficult because of the brittle nature of the ABS plastic that makes up the case. It bends a slight amount, but cracks along edges where it has been stretched during the vacuum forming process (figure 7.2). A better solution would be to have some sort of door that would allow easy access to the battery from the side.

Another limiting factor is the speech synthesis module in the Bricket.

The current solution provides easy integration, but is costly and large. A new option might be one of the myriad of chips being produced for cell-phone like devices. Speech synthesis is rapidly becoming a feature that companies wish to integrate into these products. One example is the single-chip solution offered by WinBond electronics (http://www.winbond.com/). As this technology matures and becomes available, it might present an ideal solution for size, cost and power consumption. I contacted the company for a sample, but they did not arrive in time to investigate their functionality.

7.2 Open Questions

This small study brought up a number of further questions. These revolve around how visually impaired children learn, and how to build an appropriate programming system for them.

7.2.1 Concepts and Approaches

The objects created by my study participants raised the question of whether there are sets of concepts and approaches that are more likely to be familiar to a visually impaired learner. This community's daily interactions already develop certain senses more than others (most sighted people are impressed by the visually impaired's developed senses of touch and hearing). My study has provoked me to ask whether there is a certain class of ideas and concepts that are similarly more developed in the visually impaired.

In particular, this study singled out the concept of negative feedback as such an issue. Unlike open loop control systems, negative feedback allows for self-correcting mechanisms. Fred Martin's PhD dissertation presents thorough examples of students exploring the idea of feedback with a Programmable Brick (Martin, 1994). His works demonstrates that, even after being introduced the idea of negative feedback as an important control system for mobile robot navigation, students continued to rely on inaccurate dead-reckoning position data.

In marked contrast to Martin's examples, the learners I worked with immediately embraced negative feedback systems as a natural way to

implement control systems. This suggests to me that because of their visual impairment, negative feedback systems became part of their everyday lives. The routine of performing some act, expecting tactile feedback, and performing it again is precisely this idea. A concrete example of this can be found in their use of the cane. As a visually impaired person sweeps a cane back and forth, their arc and position changes based on what they find. If they encounter some object, their gait and sweep changes to discover the range of the obstacle, and adjustments are made to bypass it. This is a negative feedback control system, but the key difference is that they are part of it - using some device to sense, and then acting on that data to correct for some goal. This close relationship to the concept leads me to propose that the idea of negative feedback is more likely to be developed in visually impaired learners. It also presents another model of introducing sighted students to the idea of feedback - one that places them in the control loop by replacing one of their senses with an electronic sensor.

Feedback is just one example of a concept that seemed to be developed by the visually impaired learners I worked with. It strikes me that further research into these issues would shed light on how to design better learning tools for the visually impaired.

7.2.2 New Languages for Visually Impaired Programmers

I created this programming environment for visually impaired by making evolutionary changes to an existing language. I made no major changes to the syntax of the programming language itself. The commands I added were to enable specific new technological capabilities. The important change I did make, limiting the user to editing one procedure at a time, was based on a desire to make the text more navigable.

The issue of creating a programming environment for visually impaired learners from scratch suggests a more revolutionary approach. Perhaps even the nature of the language should be questioned. A thorough survey of adult visually impaired programmers' habits might lead to guiding principles for a more intuitive programming language for the visually impaired. As more and more visually impaired people turn to

computation to create and learn with, such a language could become very important. Perhaps successful computer researchers who happen to be blind are the best developers to build such a language, because of their intimate knowledge of the domains in question.

However, there are problems with creating a new language for this community. Visually impaired developers would be immediately isolated from their peers, and a community would have to develop around such a language. These issues are key to the adoption of a tool. Such questions were too much to tackle in this thesis, but are currently unaddressed in the research community, which focuses on making evolutionary changes to current programming languages.

7.3 Conclusion

This thesis presents an example of technologies that allow the visually impaired to create computational artifacts and activities that allow them to explore relevant domains of knowledge. I have documented the process of developing for this community's needs, and the learning topics that were discovered to be important to the participants of my study.

Bringing the constructionist approach towards learning activities to this new community proved quite successful. In particular, focusing on activities with physical constructions proved appropriate. It quickly became apparent that because of their visual impairment, the learners I worked with felt comfortable engaging with tools in a tactile way. This interaction lends itself towards building physical objects. While constructionist activities do not all focus around tangible objects, this audience seems to find those most resonant with their interaction patterns.

The case studies present a closer look at exactly what ideas I explored with my study participants. Their constructions show a willingness to adopt the Bricket technology as a creative tool to build with. Building their own computational devices introduced them to some basic ideas of computation and engineering. They cam up with numerous ideas that are similar to commercial offerings, such as

- an audio note-taker
- a remote control for the television

- a speaking watch

All of these are assistive devices designed for and sold to the visually impaired community. The Bricket computational construction kit we worked with gave these learners an inclination of how the various pieces of electronics in these devices are put together and controlled. However, they were able to do so in a way that explored some foundational ideas of computation, changing their attitudes towards what computation is.

References

Andre, E., W. Finkler, W. Graf, T. Rist, A. Schauder and W. Wahlster (1993). *WIP: The Automatic Synthesis of Multimodal Presentations* In: M. Maybury (ed.), Intelligent Multimedia Interfaces, pp. 75-93, AAAI Press, 1993, Also as DFKI Research Report RR-92-46.

Blattner, M., D. Sumikawa, and R. Greenberg (1989). *Earcons and icons: Their structure and common design principles.* Human Computer Interaction, 4(1), pp. 11-44

Borovoy, R., M. McDonald, F. Martin, and M. Resnick (1996). *Things That Blink: Computationally Augmented Name Tags.* IBM Systems Journal 35, os. 3&4, 488-495.

Brown, J.S., A. Collins, and P. Duguid (1989). *Situated Cognition and the Culture of Learning.* Educational Researcher, 1989, 18, pp. 32-42.

Colella, V.S. (1998). *Participatory Simulations: Building Collaborative Understanding Through Immersive Dynamic Modeling.* Master's Thesis, MIT, Cambridge, MA.

Dewey, J. (1938). *Experience and Education.* New York: Collier Books.

Francioni, J.M. and A.C. Smith (2002). *Computer Science Accessibility for Students with Visual Disabilities.* Proceedings of 33rd SIGCSE Technical Symposium on Computer Sceince Education, Northern Kentucky, February 2002, pp. 91-95.

Fries, Emil B (1980). *But You can Feel It.* Binford & Mort : Portland, Oregon.

Gaver, W. (1986). *Auditory Icons: Using sound in computer interfaces.* Human Computer Interaction, 2(2), pp. 167-177.

Hillis, W.D. (1975). *Ten Things to Do with a Better Computer.* Unpublished memo available from MIT Artificial Intelligence Laboratory, Cambridge, MA.

Holbrook, M.C and A.J. Koening, Ed. (2000). *Foundations of Education, Volume I : History and Theory of Teaching Children and Youths with Visual Impairment.* Second Edition. American Foundation for the Blind, New York, New York.

Krebs, C.S. (2000). *Beyond Blindfolds: Creating an Inclusive Classroom through Collaboration.* RE:view v31 n4 p180-86.

Lave, J. and E. Wenger (1991). *Situated Learning and : Legitimate Peripheral Participation.* Cambridge, UK: Cambridge University Press.

Martin, F., B. Mikhak, and B. Silverman. *MetaCricket: A designer's kit for making computational devices.* IBM Systems Journal (Vol. 39, Nos. 3 & 4)

McNerney, T. (2000). *Tangible Programming Bricks: An Approach to Making Programming Accessible to Everyone.* Master's Thesis, MIT. Cambridge, MA.

Papert, S. and C. Solomon (1971). *Twenty Things to Do with a Computer.* Artificial Intelligence Memo 248, MIT Artificial Intelligence Laboratory, Cambridge, MA.

Papert, S. (1994). *The Children's Machine.* New York: Basic Books.

Piaget, J. (1963). *The Origins of Intelligence in Children.* New York: Norton.

Reich, Y., S.L. Konda, S.N. Levy, I.A. Monarch, and E. Subrahmanian (1996), *Varieties and Issues of Participation and Design.* Design Studies, 17(2):165-180.

Resnick, M. (1993). *Behavior Construction Kits.* Communications of the ACM, vol. 36, no. 7 (July 1993)

Resnick, M., R. Berg, and M. Eisenberg (2000). *Beyond Black Boxes: Bringing Transparency and Aesthetics Back into Scientific Investigation.* Journal of Learning Sciences.

Resnick, M., A. Bruckman, F. Martin (1996). *Pianos Not Stereos: Creating Computational Construction Kits.* Interactions, Vol 3. no. 6.

Rose, D. and A. Meyer (2000). *Universal Design for Learning – Associate Editor Column.* Journal of Special Education Technology. Volume 15.1.

Rosenblatt, M. and Mikhak, B. (2002). *The Living LEGO City: A Computationally Enhnaced City Building Activity.* In preparation for submission to the 2002 ICLS conference.

Smith, A.C., J.M. Francioni, and S.D. Matzek (2000). *A Java Programming Tool for Students with Visual Disabilities.* Proceedings of ACM Assets 2000, Washington, D.C., November 2000, pp. 142-148.

Schön , D.A. (1987). *Educating the Reflective Practitioner.* Jossey-Bass, San Francisco.

Suzuki, H. and H. Kato (1993). *AlgoBlock: a Tangible Programming Language, a Tool for Collaborative Learning.* Proceedings of 4th European Logo conference, pp. 297-303. Athens, 1993.

Vincent, A.T. (1981). *Computer-Assisted Support for Blind Students: The Use of a Microcomputer Linked Voice Synthesizer.* Presented at CAL 81: Symposium on Computer Assisted Learning (Apr. 8-10 ,1981).

Appendix A - BricketLogo Reference

A.1 Motor Commands

motor-on
Activates the Bricket's internal pager motor.

motor-off
Deactivates the Bricket's internal pager motor.

A.2 Timing Commands

resett
Resets the elapsed time counter to zero.

timer
Reports value of free-running timer. Time is counted in milliseconds.

wait duration
Delays for a duration of time, where duration is given in tenths-of-seconds.

get-day
Reports the current day of the month from the real-time clock bus-device.

get-month
Reports the current month of the year from the real-time clock bus-device.

get-year
Reports the current year from the real-time clock bus-device.

get-dow
Reports the index of the current day of the week from the real-time clock bus-device.

This reference relies heavily on a CrikcetLogo command reference created by Bakhtiar Mikhak.

get-hr
Reports the current hour of the day from the real-time clock bus-device.

get-min
Reports the current minute of the hour from the real-time clock bus-device.

get-sec
Reports the current seconds of the minute from the real-time clock bus-device.

set-datetime day month year dow hour minute
Sets the time of the real-time clock bus-device to the specified day, month, year, day of the week, hour, and minute.

set-time hour minute
Sets the time of the real-time clock bus-device to the specified hour and minute.

set-date day month year dow
Sets the time of the real-time clock bus-device to the specified day, month, year, and day-of-week.

A.3 Sound Commands

beep
Emits a short beep.

note pitch duration
Plays a note of a specified pitch and duration. Increasing values of the pitch create lower tones. The duration value is specified in tenths-of-seconds units.

say "a_phrase_to_say
Says the specified words. An underscore is equivalent to a space.

say-number number
Says the number specified. These number have to be between 0 and 999.

skip
Makes the Bricket stop whatever it is saying and move on the next thing to say.

shh
Makes the Bricket stop whatever it is saying and anything is was going to say afterwards.

volume number
Sets the volume of the Bricket's voice to specified number, which has to be between 0 and 7. 0 is quiet and 7 is loud.

speed number
Sets the speed of the Bricket's voice to the number specified, which has

to be between 0 and 7. 0 is slow and 7 is fast.

dial number
Emits the appropriate dialing tones for the number specified (between 0 and 9).

voice number
Changes the Bricket's voice (between 0 and 2).

articulation number
Changes how the Bricket generates its speech.

expression number
Changes how the Bricket generates its speech.

frequency number
Changes how the Bricket generates its speech by altering the frequency.

pitch number
Changes how the Bricket generates its speech by altering its pitch.

tone number
Emits tones.

reverb number
Changes how the Bricket generates its speech by adding in a reverb effect.

A.4 Sensor Commands

sensora
Reports the value of sensor A, as a number from 0 to 255.

sensorb
Reports the value of sensor B, as a number from 0 to 255.

switcha
Reports "true" if the switch plugged into sensor A is pressed, and "false" if not.

switchb
Reports "true" if the switch plugged into sensor B is pressed, and "false" if not.

resetdp
Reset the value of data pointer to 0.

record value
Records value in the data buffer and advances the data pointer.

recall value
Reports the value of the current data point and advances the data pointer.

erase number
Sets the value of the first number elements of the data array to zero and then sets the data pointer to zero.

A.5 Control Commands

loop [body]
Repetitively executes the body code indefinitely.

repeat times [body]
Executes the body code for times repetitions. times may be a constant or a calculated value.

if condition [body]
If the condition is true, the cricket executes the body code. Note: a condition expression that evaluates to zero is considered "false"; all non-zero expressions are "true".

ifelse condition [body1] [body2]
If condition is true, executes body-1; otherwise, executes body-2.

waituntil [condition]
Loops repeatedly testing condition, continuing subsequent program execution after it becomes true. Note that condition must be contained in square brackets; this is unlike the conditions for if and ifelse, which do not use brackets.

stop
Terminates execution of procedure, returning control to calling procedure.

output value
Terminates execution of procedure, reporting value as result.

A.6 Number Commands

+
Addition operator.

-
Subtraction operator.

*
Multiplication operator.

/
Division operator.

%
Modulus operator (remainder after integer division).

and
Logical bitwise and operation

or
Logical bitwise or operation.

xor
Logical bitwise or exclusive or.

not
Logical not operation. Use only with boolean values (1 or 0).

random
Reports a pseudo-random number between 0 to 32767.

A.7 Multitasking Commands

when [condition] [body]
Launches a parallel process that repeatedly checks condition and executes body whenever condition changes from false to true. The when rule is "edge-triggered" and remains in effect until it is turned off with the whenoff primitive. Only one when rule can be in effect at a time; if a new when rule is executed by the program, this new rule replaces the previous rule.

whenoff
Turns off any existing when rule.

A.8 Infrared Communication Commands

send value
Transmits a value via infrared.

ir
Reports the byte most recently received by the infrared detector. Note that the Blue Dot crickets do not clear the infrared buffer. Thus the ir primitive reports the most recent byte received.

newir?
Reports true if a new byte has been received by the infrared detector since last time ir was used, and false if not. It does not effect the content of the infrared buffer.

Appendix B - Program Code

The following pages present the BricketLogo programs created by the participants. These are the programs that ran the three constructions described in the case studies.

These are written here in the BricketLogo file format. Thus they are presented as Logo code, rather than the representation the children worked with. I have added tabs for formatting to ease readability.

B.1 The Electronic Cane

```
global [ my-d d1 d2 d3 ]

to start
  setmy-d 0
  when [ switcha ]
  [ speak my-d ]
  loop [
    set-them
    fix-them
    speak my-d
    wait 11
  ]
end

to set-them
  setd1 ( distance-red + distance-blue ) / 2
  wait 1
  setd2 ( distance-red + distance-blue ) / 2
  wait 1
  setd3 ( distance-red + distance-blue ) / 2
end

to fix-them
  set my-d ( d1 + d2 + d3 ) / 3
end
```

B.2 The Tricorder

```
global [ light temp ]

to start
  if switcha
    [ buzz-distance ]
  if switchb
    [ tell-me-light ]
    start
end

to buzz-distance
  if ( distance > 140 )
    [ motor-on
    wait 5
    motor-off ]
end

to tell-me-light
  setlight sensora
  if ( light > 0 and light < 50 )
    [ note 10 100 ]
  if ( light > 50 and light < 100 )
    [ note 10 90 ]
  if ( light > 100 and light < 150 )
    [ note 10 80 ]
  if ( light > 150 and light < 200 )
    [ note 10 70 ]
  if ( light > 200 and light < 255 )
    [ note 10 60 ]
end

to tell-me-temp
  settemp sensorb
  ifelse ( temp > 94 )
    [ say "its_hot ]
    [ say "its_cold ]
end
```

B.3 BricketBot

```
global [ which-one ]

to start
  alarm-clock
  check-stuff
end

to alarm-clock
  when [ get-hour = 6 and get-minute = 30 ]
  [ say "wake_up_lisa
  note 20 10
  note 10 70
  note 5 55
  note 10 70
  say "wake_up_lisa
  ]
end

to check-stuff
  if ( switcha )
    [ set which-one 1 ]
  if ( switchb )
    [ set which-one 2 ]
  pick-time
end

to pick-time
  ifelse ( get-hour < 12 )
    [ morning ]
    [ night ]
end

to morning
  if ( which-one = 1 )
    [ weather ]
  if ( which-on = 2 )
    [ hello ]
end

to night
  if ( which-one = 1 )
    [ story ]
  if ( which-on = 2 )
    [ song ]
end

to weather
  say "the_temperature_is
  say-number sensora
end
```

```
to hello
   say “hello
   say “my_name_is_bricket_bot
   say “lisa_and_i_are_friends
end

to story
   say “my_name_is_bricket_bot
   say “i_am_a_friendly_robot
   say “lisa_made_me
   say “and_i_like_to_sing
   say “but_i_wish_i_could_dance
end

to song
   song1
   song2
   song3
end

to song1
   note 10 60
   note 5 55
   note 10 61
   note 5 48
   note 10 44
   note 7 30
   note 6 100
   note 7 96
   note 5 80
end

to song2
   note 6 70
   note 8 85
   note 4 97
   note 4 100
   note 8 100
   note 10 110
   note 10 140
   note 2 10
end

to song3
   repeat 5 [
      note 4 70
      note 4 53
   ]
end
```