

**Circuits to Control:  
Learning Engineering by Designing LEGO Robots**

by

Fred Garth Martin

B.S., Massachusetts Institute of Technology (1986)

S.M., Massachusetts Institute of Technology (1988)

Submitted to the Program in Media Arts and Sciences  
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1994

© Massachusetts Institute of Technology 1994. All rights reserved.

Author .....  
Program in Media Arts and Sciences  
April 29, 1994

Certified by .....  
Edith Ackermann  
Associate Professor of Media Arts and Sciences  
Program in Media Arts and Sciences  
Thesis Supervisor

Accepted by .....  
Stephen Benton  
Chairman, Departmental Committee on Graduate Students  
Program in Media Arts and Sciences

**Rotch**  
MASSACHUSETTS INSTITUTE  
OF TECHNOLOGY

**JUL 13 1994**

# **Circuits to Control: Learning Engineering by Designing LEGO Robots**

by

Fred Garth Martin

Submitted to the Program in Media Arts and Sciences  
on April 29, 1994, in partial fulfillment of the  
requirements for the degree of  
Doctor of Philosophy

## **Abstract**

The nature of an undergraduate degree in engineering has undergone significant change since the end of the second World War. There is more theoretical content and less hands-on project work, reflecting both rapid advances in the state of scientific theory and educators' ideas about what engineering students need to know.

Since the 1960's, engineering educators have been aware of problems with this new curriculum: students graduate with the ability to analyze clearly presented problems, but little or no background in doing design, which is the central work of a practicing engineer. In the past, employers accepted that design skills—including the ability to transform under-specified and messy design situations into actual problems to be solved—would be learned on the job, but this has become increasingly less acceptable in today's global economy.

The focus of this thesis is an analysis of the situation through the development and evaluation of a model class experience for undergraduate engineering students that addresses the deficiencies in the traditional education. The model course has been developed with and tested on MIT undergraduate students over the past four years. It consists of a month-long intensive design workshop in which students are responsible for the conception, design, implementation, debugging, and competitive demonstration of an autonomous robotic device.

The core work is the task of developing and testing this design-rich learning environment with the goal of discovering the characteristics of the setting which most powerfully encourages students' learning. The methodology employed is the implementation of a "living laboratory" in which a series of design environments (i.e., workshop design classes) are successively developed, tested, and evaluated. The evaluation is based on a variety of observational tools, including interaction with students during the progress of their projects, student written reports and journals, and analysis of the actual products of the students' work—robotic hardware and software systems. The purpose of the evaluation is to understand the issues that the students face in accomplishing their design task, in order to ascertain what and how they are learning, and to improve the materials and the classroom environment in the future.

The outcome of this work is several-fold. Most importantly, it is a re-evaluation

and further understanding of the role of design work in the undergraduate engineering degree program, with a focus on specific ways to build empowering experiences into the undergraduate curriculum. Secondly, it reveals that students have pre-existing conceptions of systems and control that make it difficult for them to deal with sensor noise and other erratic phenomena in their robot designs. Thirdly, it develops a set of technological tools for learning—a kit optimized for students to work on robotic design projects. While the particulars of this technology may become outdated in a few years, the more important nature of its interactive qualities and theory behind its design will not.

Thesis Supervisor: Edith Ackermann  
Title: Associate Professor of Media Arts and Sciences  
Program in Media Arts and Sciences

This work was supported by the National Science Foundation (grants 9153719–MDR and TPE–88050449), the LEGO Group, and the Nintendo Corporation.

# Doctoral Committee

Thesis Advisor .....  
Edith Ackermann  
Associate Professor of Media Arts and Sciences  
Program in Media Arts and Sciences

Thesis Reader .....  
Pattie Maes  
Assistant Professor of Media Arts and Sciences  
Program in Media Arts and Sciences

Thesis Reader .....  
Seymour Papert  
LEGO Professor of Learning Research  
Program in Media Arts and Sciences

Thesis Reader .....  
Donald Schön  
Ford Professor Emeritus and Senior Lecturer  
Department of Urban Studies and Planning

# Contents

<b>Preface</b>	<b>13</b>
<b>Acknowledgments</b>	<b>15</b>
<b>Overview</b>	<b>18</b>
<b>1 Background</b>	<b>20</b>
1.1 Engineering Education . . . . .	20
1.1.1 The Grinther Report . . . . .	21
1.1.2 The MIT Report on Engineering Design . . . . .	25
1.1.3 Computer Assisted Instruction . . . . .	26
1.1.4 Contemporary Trends . . . . .	26
1.2 Design Research . . . . .	28
1.2.1 Design as a Formal Process . . . . .	28
1.2.2 Design as a Negotiational Activity . . . . .	29
1.2.3 Epistemological Pluralism . . . . .	30
1.2.4 Design Communities . . . . .	31
1.2.5 Design in University Education . . . . .	33
1.2.6 Design at MIT . . . . .	33
1.3 Educational Technology . . . . .	37
1.3.1 Constructionism . . . . .	38
1.3.2 LEGO/Logo . . . . .	42
1.3.3 The Programmable Brick . . . . .	43

<b>2</b>	<b>Introduction</b>	<b>48</b>
2.1	A Robot Design Competition . . . . .	48
2.2	Relationships for Learning . . . . .	49
2.2.1	Structure and Freedom . . . . .	50
2.2.2	Accountability . . . . .	52
2.2.3	Versatility . . . . .	52
2.2.4	Teamwork . . . . .	53
2.2.5	Community . . . . .	54
2.2.6	Motivation . . . . .	55
2.3	Methodology and Evaluation . . . . .	58
2.4	The Main Themes . . . . .	62
2.4.1	Technology for Learning . . . . .	62
2.4.2	Ideal and Real Systems . . . . .	63
2.4.3	Design Styles . . . . .	63
<b>3</b>	<b>Technology for Learning</b>	<b>64</b>
3.1	Contest Design . . . . .	67
3.1.1	Strategic Diversity . . . . .	71
3.1.2	Challenge Level . . . . .	81
3.1.3	The Social Message . . . . .	84
3.1.4	Summary . . . . .	87
3.2	Hardware and Software Design . . . . .	88
3.2.1	Levels of Abstraction . . . . .	91
3.2.2	Observability . . . . .	96
3.2.3	Interactivity . . . . .	100
3.2.4	Transparency . . . . .	104
3.3	Sensor Design . . . . .	106
3.3.1	Collaborative Learning . . . . .	106
3.3.2	Motivating Understanding . . . . .	113
3.4	Summary . . . . .	118

<b>4</b>	<b>Ideal and Real Systems</b>	<b>120</b>
4.1	Introduction to Robotic Control . . . . .	121
4.1.1	The Omniscient Robot Fallacy . . . . .	121
4.1.2	Understanding Sensing . . . . .	124
4.1.3	Models of Control . . . . .	127
4.2	Case Study One: Monitoring Robot Position . . . . .	139
4.3	Case Study Two: Simulation as a Development Tool . . . . .	145
4.4	Analysis . . . . .	148
<b>5</b>	<b>Design Styles</b>	<b>153</b>
5.1	Affirmative Action for Bottom-Up Design . . . . .	154
5.1.1	Experimenting with the LEGO Technic System . . . . .	155
5.1.2	Programming Strategies . . . . .	160
5.1.3	Time Constraints . . . . .	161
5.2	Design Excursions . . . . .	162
5.2.1	Improving the Motor Drivers . . . . .	163
5.2.2	Developing New Sensors . . . . .	166
5.2.3	Musical Robotics . . . . .	167
5.2.4	Alternate Control Systems . . . . .	168
5.3	Redefining the Goal of Participation . . . . .	169
5.3.1	Mechanical Intelligence . . . . .	170
5.3.2	Deliberate Complexity . . . . .	173
5.3.3	The Talent Show . . . . .	175
<b>6</b>	<b>Conclusion and Future Directions</b>	<b>177</b>
6.1	Educational Technology . . . . .	179
6.1.1	Levels of Abstraction . . . . .	180
6.1.2	Robot Design at Other Universities . . . . .	181
6.1.3	Robot Design in Secondary Education . . . . .	184
6.2	Models of Control . . . . .	186
6.2.1	Artificial Life and Behavioral Robotics . . . . .	187

6.2.2	The Role of Sensing . . . . .	188
6.3	Design Styles and Engineering Knowledge . . . . .	189
6.3.1	Mysterious Phenomena . . . . .	189
6.3.2	The Role of Reflection . . . . .	191
<b>A</b>	<b>Robot Glossary</b>	<b>193</b>
<b>B</b>	<b>Contest Design</b>	<b>195</b>
B.1	King of the Mountain, 1989 . . . . .	195
B.2	Robo-Puck, 1990 . . . . .	202
B.3	Robo-Pong, 1991 . . . . .	207
B.4	Robo-Cup, 1992 . . . . .	212
<b>C</b>	<b>Administrative Considerations</b>	<b>219</b>
C.1	Granting Academic Credit . . . . .	219
C.2	Student Costs . . . . .	222
C.3	Kit Ownership . . . . .	223
C.4	Lotteries . . . . .	224
C.5	Non-MIT Participation . . . . .	225
<b>D</b>	<b>Technology Development</b>	<b>226</b>
D.1	The Remote Controller . . . . .	226
D.1.1	Hardware . . . . .	227
D.1.2	Software . . . . .	229
D.2	The Assembly Language Controller . . . . .	230
D.2.1	Hardware . . . . .	231
D.2.2	Software . . . . .	232
D.3	The C Language Controller . . . . .	234
D.3.1	Hardware . . . . .	235
D.3.2	Software . . . . .	238
D.3.3	Results . . . . .	242
D.4	Sensor Development . . . . .	242



D.4.1	Robot Detection . . . . .	245
D.4.2	Deploying the System . . . . .	246
D.4.3	Evaluating the System . . . . .	247
<b>Bibliography</b>		<b>251</b>

# List of Figures

1-1	Early Logo robot and button panel for direct manipulation . . . . .	40
1-2	Logo turtle and typical Logo drawings made using turtle graphics . . . . .	41
1-3	The commercial “ <i>LEGO tc logo</i> ” system marketed by LEGO Dacta USA . . . . .	44
1-4	The LEGO/Logo Programmable Brick . . . . .	45
1-5	The Programmable Brick system . . . . .	46
2-1	Cycle of project design and assessment . . . . .	59
3-1	Triangle of technology: contest specifications, computer control hardware and software, and sensor devices . . . . .	65
3-2	Visualization of solo strategy relationships . . . . .	67
3-3	Visualization of interacting strategy relationships . . . . .	68
3-4	Year and name of robot design contests . . . . .	70
3-5	Schematic view of robots employing wheelchair drive configuration . . . . .	73
3-6	Playing table for the <i>Robo-Puck</i> contest . . . . .	74
3-7	<i>Bertha</i> , a successful puck-fetching robot from <i>Robo-Puck</i> . . . . .	75
3-8	<i>Robo-Pong</i> game design . . . . .	76
3-9	<i>Robo-Cup</i> contest playing table . . . . .	79
3-10	Close-up of goal in <i>Robo-Cup</i> game . . . . .	80
3-11	Graphic used to promote the first LEGO Robot Design Contest . . . . .	85
3-12	Graphic used to promote 1991 <i>Robo-Pong</i> contest . . . . .	86
3-13	Three stages of robot-building technologies . . . . .	90
3-14	Session with Interactive C (page one) . . . . .	101
3-15	Session with Interactive C (page two) . . . . .	102

3-16	Mercury switch . . . . .	107
3-17	Rolling ball sensor in level position . . . . .	107
3-18	Rolling ball sensor in tilted position . . . . .	108
3-19	Modified rolling ball sensor . . . . .	109
3-20	One Sharp Electronics no. GP1U52X infrared sensor and eight infrared transmitter LEDs, with a U.S. penny for size reference . . . . .	111
3-21	Cadmium sulfide photocell . . . . .	114
3-22	Assembly instructions for reflectance sensor from <i>Robo-Pong</i> contest hand- outs . . . . .	115
3-23	Wiring diagram for upgraded reflectance sensor provided to students in <i>Robo-Cup</i> contest . . . . .	117
4-1	Photograph of <i>Groucho</i> . . . . .	133
4-2	<i>Groucho</i> 's strategy as played in <i>Robo-Pong</i> contest . . . . .	134
4-3	Idealized lead-acid cell discharge curve . . . . .	137
5-1	Motor Switch Board used for manual control of kit motors . . . . .	163
5-2	<i>Blind</i> unfurls its arm toward the puck . . . . .	171
5-3	<i>Blind</i> brings puck to the rim . . . . .	172
5-4	<i>Blind</i> loses control of the puck to <i>Bertha</i> . . . . .	172
5-5	Strategy diagram for <i>Mutton Jeff</i> in <i>Robo-Pong</i> contest . . . . .	173
6-1	Designing robot-building environments and studying learners' experiences with them . . . . .	178
6-2	The 1994 version of the Programmable Brick . . . . .	185
B-1	<i>Robo-Puck</i> table specifications . . . . .	203
B-2	<i>Robo-Pong</i> table specifications . . . . .	207
B-3	<i>Robo-Cup</i> contest playing field specification . . . . .	213
B-4	<i>Robo-Cup</i> contest goal specification . . . . .	213
C-1	Number of registrants by year, 1989 through 1994 . . . . .	224

D-1	The 1989 Remote Controller Board . . . . .	228
D-2	Converting analog light reading to digital pulse stream . . . . .	229
D-3	The 1990 Assembly Language-based Controller Board (actual size) . . . . .	231
D-4	The 1991 C-language-based Robot Controller Board (actual size) . . . . .	236
D-5	Page from the electronic surplus catalog of <i>Marlin P. Jones &amp; Associates</i> (used with permission) . . . . .	244

# Preface

Traditional educational methods rarely consider the role that design activity can play in the learning process. In the United States, kindergarten is often both the first and last place in formal education in which exploratory and constructive play is the central way that learning happens. Beginning in the first grade and continuing through secondary school into the university, school is predicated on telling (i.e., lectures and readings) and testing (i.e., worksheets, problem sets, and written examinations). The narrowness of this pedagogy is part of the reason our educational system is presently dealing with a crisis of relevancy and performance.

In the field of engineering education at the university level, there is a more specific problem. Students are required to learn more and more theoretical material as part of their education. This material has squeezed design-oriented material from the curriculum; at the same time, universities have moved away from design, as it has seemed too practical and lacking in scientific basis. As put by Daniel Whitney in a recent article, “Mathematical analysis replaced design, manufacturing disappeared from the curricula, and both faculty and graduates lost touch with how engineering, design, and manufacturing interact in the ‘real world’.” (Whitney, 1990)

Yet the problem goes deeper than the level of the content of a university curriculum. In 1961, a committee at the Massachusetts of Technology published the *Report on Engineering Design* in which it raised serious concerns with the predominating pedagogy of the day, characterized by a preponderance of “single-answer problems” (M.I.T., 1961). The Committee believed that the goal of an engineering education must be not only to provide students with necessary technical backgrounds, but to inculcate certain skills and dispositions, including things like an ability to work in the face of incomplete and contradictory

data (which are commonplace situations for practicing engineers). One of the Committee's overriding concerns was the effect of having the bulk of a student's education be performed by having him or her solve problems that had only one correct answer. The Committee believed that this sort of education could greatly hinder students from developing the type of ability they believed to be central to the practice of engineering design.

The very pedagogy criticized in the committee's 1961 report is still in full force today, largely due to practical reasons, like ease of evaluation and ease of teaching, that were even cited in the committee's report. Since the time of the report, however, the engineering education community has come to a greater recognition of the need for design experiences in the undergraduate curriculum. A concrete manifestation of this awareness is the recent addition of a one-term engineering design requirement to the program specifications of the Accreditation Board of Engineering and Technology (ABET) (Jones, 1991).

Most universities satisfy both the ABET requirements and their own sense of the need for a design activity with the capstone design course. These courses have the distinguishing characteristic of being taken by students at the end of their undergraduate careers—and possibly their formal educational careers, as most students enter industry upon graduation. As such, the premise of the capstone course is that *students must learn analysis before they can do "synthesis"* (i.e., design).

This curriculum fix, the capstone course, indicates a lack of depth in the understanding that most educators have about the role of design in the learning process. The development of designer's attitudes, called for by the MIT Engineering Design committee, will not be significantly affected by one or two terms of design activity at the end of an educational career. By relegating a design course to this position, educators may have students learn *about* design, but they miss the opportunity to have them learn *how to* design. Those who create more eccentric design courses have a greater sense that many students learn *through the act of* designing. Unfortunately, most university courses do not entertain this possibility, and the ones that do often have a marginal status in the university curriculum.

# Acknowledgments

I've had the great fortune of an incredibly supportive and wonderfully perceptive thesis advisor, Prof. Edith Ackermann. Edith was extremely generous with her time, and was able to find the strands of meaning in what initially seemed to me as only a mess of interesting issues. Throughout she was always encouraging, excited, and also firm when necessary. Edith, I am truly grateful to you for your kindness and wisdom.

I began working with Prof. Seymour Papert and his research group just after finishing my Bachelor's degree from MIT in 1986. Since that time, as I have matured in my thinking, I have been profoundly influenced by Seymour's ideas and those of the people he has brought together around him. Seymour would always find the little idea buried in a pile of other considerations that needed to be let out and given a chance to grow. Seymour, I will always be indebted to you for your gracious support and advice over these years.

About a year ago I asked Prof. Don Schön to join my committee, and he has since contributed more to my thinking than I ever could have anticipated. I remember one meeting with Don when he was trying to explain to me an important theme he saw in my work, and after going around in circles for a bit, I realized that I just didn't "get it." We met again a couple weeks later, and he brought the conversation back to the same point. This time, something clicked in my brain, and I suddenly had a whole new lens with which to look at my work. I hope that some of Don's insights come through in this document. Don, I want to thank you for both for your extremely valuable contributions and your warmth and friendliness.

At an earlier stage in the writing of this dissertation, there was a section that none of my committee really liked. Well, they believed there was something of value in this section, but it was largely obscured by a poor structure. I have Prof. Pattie Maes to thank for putting

her foot down and saying, “No Fred, this just isn’t working.” Fortunately she also had some good ideas about how to fix it, and looking back the surgery wasn’t too major. Pattie, I’d like to thank you for your determined attempts to keep me honest. Any shortcomings in this final document are surely my own responsibility.

This whole project would not have happened without the collaboration I shared with Pankaj (“P.K.”) Oberoi and Randy Sargent. When the three of us began developing materials and ideas for this project, I doubt that any of us knew how much attention it would receive and excitement it would generate in the student body. We each brought different talents to the project, including Randy’s technical prowess and P.K.’s enthusiasm and organizational ability. I hope that this document gives the reader some sense of the extent to which this project has been a joint intellectual effort; while I present my own interpretations and conclusions here, my perspective has been greatly influenced by conversations we had and decisions we made while working together. P.K., Randy: I had a great time working with the two of you, and I wish you the best in the future.

I have enjoyed many valuable relationships with my colleagues at the Media Laboratory. To Mitch Resnick: it’s been great working with you since the early days of LEGO/Logo. To Yasmin Kafai: your hard work and self-confidence is a constant inspiration to me. I appreciate how you have gone out of your way on my behalf. To Aaron Falbel: your ideas about learning always keep me on my toes. To Carol Strohecker: thank you for interesting conversations and helping me find an interesting title. To Paula Hooper, Michele Evard, and Alan Shaw: thank you for caring.

Three faculty members of the Electrical Engineering and Computer Science department have been staunch supports of this project from the beginning. Professor Leonard Gould gave us encouragement and helpful advice, Department Head Paul Penfield made necessary funds available, and Professor Jeffrey Shapiro helped us figure out that we really wanted to give away the project’s hardware and software technology (which we did).

As a sponsor of the work of the Epistemology and Learning Group and as a direct contributor to the Robot Design project, the LEGO Group has been a wonderful friend. As a company, they have provided valuable funding and even more valuable LEGO parts, shipped straight from Denmark; at a personal level I have enjoyed conversations and support



from Robert Rasmussen and Lars Bo Jensen.

For several of my years as a doctoral student, I was a graduate resident tutor at Bexley Hall (an undergraduate dormitory on the MIT campus). During this time I enjoyed a friendship with Prof. William Orme-Johnson, the housemaster at Bexley. Prof. Orme-Johnson took a genuine interest in my personal well-being and my work at MIT. Bill, I thank you for your thoughtfulness and perspective.

I always felt that Jacqueline Karaaslanian and Mai Cleary, the administrative staff for the Epistemology and Learning Group, were on my side. They more than did their jobs; they cared about me and my work, and were always a pleasure to be around.

Wanda Gleason has been my friend, confidante, and lover since before I began writing this thesis. She has been incredibly supportive and confident in me at times when I was down. Wanda, it has been wonderful to have you by my side during this endeavor. I thank you for everything.

My mother and father have encouraged me to do my best since I was a little child. They have always helped me discover what I really wanted to do, which is the work I'm presenting here. Mom and Dad, thank you and I love you.

Finally, I would like to acknowledge all of the students who participated in this activity and often gave it their best. This project would not have happened without your no-holds-barred participation.

# Overview

This dissertation is organized as six chapters, four appendices, and a bibliography:

- 1—Background.** The background chapter presents an historical context based on work in the fields of engineering education, design theory, and constructionist educational technology.
- 2—Introduction.** This chapter presents the motivations that shaped the development of the *LEGO Robot Design Competition*—the workshop course and student competition used as a basis for this research project.
- 3—Technology for Learning.** This chapter discusses the three aspects of the educational technology developed for the Robot Design project: the contest designs and specifications, the robotic hardware and software toolkit, and robotic sensor devices. This discussion illuminates key characteristics behind the design of this media, including features like the structure of a design space, and the interactivity, transparency, and level of abstraction of technological tools.
- 4—Ideal and Real Systems.** This chapter analyzes the students’ robots from a control-systems point of view. The most striking result is students’ recurring inclination to build robots that will only perform correctly in an ideal world, which is a far cry from the actual situations that their robots face.
- 5—Design Styles.** Students approach the design task with a range of design styles. Our intention in creating the workshop and its materials was to encourage playful, hands-on explorations of new ideas and phenomena. While partly successful, we found that this investigative style is uncomfortable for many students.

**6—Conclusion and Future Directions.** The workshop provides an unusual and evocative space for engineering students to face genuine design challenges, express personal creativity, and learn in unfamiliar ways. Extensions of the design environment could support deeper inquiries by university students, broader ones, or similar projects at other educational levels.

The appendices provide reference material for the reader who is interested in further details about the project's materials and history:

**A—Robot Glossary.** Each of the robot projects discussed in the dissertation is cataloged here, along with page references to the main text.

**B—Contest Design.** Full text for each of the robot contests discussed is presented here, along with additional motivations behind the contests' design.

**C—Administrative Considerations.** This appendix presents the guidelines that were established for awarding academic credit. Of particular concern was the desire for non-graded participation.

**D—Technology Development.** Technical details concerning the hardware and software developed for the project are presented here.

The final section is the dissertation bibliography.

# Chapter 1

## Background

This chapter presents three areas of work that are related to the investigations described in this dissertation. The first of these is a brief history of engineering education from the post-World War II period to the present. This is followed by a consideration of ideas in the design research community that are related to this work. Finally, there is a summary of the educational technology from which the robot-design tools developed as part of this dissertation owe their heritage.

### 1.1 Engineering Education

Since the end of World War II, an explosion of growth in the scientific and technical fields has occurred, creating new and powerful theory and technique for understanding and manipulating the physical world. Engineering education at the university level responded to this innovation by packing more and more theory and mathematics into the undergraduate years, usually at the expense of engineering practices and design-oriented material, which has been seen as “too practical” or lacking in scientific basis (Whitney, 1990; Banios, 1991).

After the war, it was apparent to many of the nation’s leaders that scientific and technical superiority was essential to national security, and government bodies were established to ensure that funds were available for this purpose. The Office of Naval Research and other military agencies supported and determined priorities for a large portion of the research performed in leading United States universities and endowed laboratories.

### 1.1.1 The Grinther Report

In 1952, the American Society for Engineering Education sponsored a periodic review of the state of engineering education. The review was performed by a committee of thirty engineering professors and administrators and was headed by L. E. Grinther, dean of the graduate school at the University of Florida. A preliminary report followed in one year (Grinther, 1954). It noted that “the art of engineering has come to depend greatly upon the basic science of engineering” and therefore faculty should have the PhD degree, and added that appropriate industrial experience is also important in any faculty. This recommendation was not followed; by the 1960’s, it would “become painfully obvious that engineering faculties had become strong in research but were generally unfamiliar with engineering practice, particularly design.”<sup>1</sup> By consequence, teachers did not “have the necessary industrial experience to introduce students to the many subtle, unstructured problems of designing, building, operating, and maintaining structures and machines.”<sup>2</sup>

The Grinther committee addressed the issue of the difference between academic research (for which ample funds were being made available by government agencies) and education oriented toward engineering practice. The committee recommended a “bifurcation” of engineering curricula, so that students wishing to pursue academic careers could take theoretically oriented classes, and students planning to go into industry could take engineering art classes.

In October of 1953, the preliminary report was sent to approximately 175 colleges for review. Faculty committees of 122 colleges responded. The next interim report summarized the overwhelming agreement of the colleges that the engineering curricula should not be subdivided into two stems. The idea of bifurcation was thus summarily dismissed.

The final report (Grinther, 1955) contained two important recommendations. The first stated that “those courses having a high vocational and skill content” should be eliminated, as should “those primarily attempting to convey engineering art and practice.” The second recommendation called for the creation of courses in the “six engineering sciences—mechanics of solids, fluid mechanics, thermodynamics, transfer and rate mechanisms (heat,

---

<sup>1</sup>Ferguson, *Engineering and the Mind's Eye*, page 159.

<sup>2</sup>*ibid.*

mass, momentum), electrical theory, and nature and properties of materials.”

Ferguson neatly summarizes the academic community’s response to this advice (Ferguson, 1992):

Thus, shop courses—intended to give students a visual and tactile appreciation of materials and basic processes, such as the welding, casting, and machining of metals—were rapidly dispensed with. Engineering drawing lingered a bit, primarily because many drawing instructors held academic rank and were difficult to fire, but the diminished status of courses in drawing and descriptive geometry was clear to all concerned. The “art and practice” courses—which described the individual components of engineering systems such as steam power plants, electrical networks, and chemical process plants and explained how the components were coordinated in practice, thus providing training in the way engineering had been and was being done—survived only until the Committee’s second recommendation could be put in place.

...

By no means were all engineering curricula changed immediately, but the gospel of change was unambiguous for the research-oriented engineering schools and for the schools that aspired to join the prosperous group.<sup>3</sup>

The final report by the Committee on Evaluation of Engineering Education was published in 1955. Because of “mounting discussion of the section devoted to the engineering sciences, it was deemed desirable to elaborate further on this section,”<sup>4</sup> and an *ad-hoc* Follow-up Committee on the Evaluation Report was appointed during the summer of 1955. Their report, published in 1958, presented a structuring of the engineering sciences into seven areas: mechanics of solids, mechanics of fluids, transfer and rate processes, thermodynamics, electrical sciences, nature and properties of materials, and engineering design and analysis.

While the first six of these subjects were presented in a traditional manner, reading like a course syllabus of topics to be covered, the engineering design and analysis section was more conceptual: “we are not trying to propose courses of specific factual content, but are proposing a very definite engineering philosophy and methodology that should underlie and be a part of all of the students’ intellectual experience at the college.” The report

---

<sup>3</sup>*ibid.*, pp. 160–161.

<sup>4</sup>“Report on the Engineering Sciences,” *Journal of Engineering Education*, volume 49, number 1, October 1958, page 36.

focused on the need for creativity in the engineering design process, in addition to the requisite analytical skills, and the relentless way that formal education drives out creative, synthesizing talent:

We say that the synthesizing ability is the last to develop and the first to be inhibited or destroyed. Closely following in the destructive process is the ability to make judgments and decisions. This is probably the result of devoting practically all of our formal education (and a good share of informal education time as well) to exploring, explaining, and exercising the process of analysis as a mode of thinking, arriving at the *one* right answer to the problem or task confronting us. We learn the *one* right way to spell cat, the *one* right way to answer two plus two, the *one* right answer to “Who won the Battle of Hastings?” and the *one* right deflection at the center of a beam, uniformly loaded and freely supported at the ends.<sup>5</sup>

The report noted that “the learning of these things is not wrong, but the learning of *only* these things is undesirable if we want to have competent creative engineers as well as competent creative citizens.” The MIT Report on Engineering Design, which would follow a few years later, would identify this same problem in formal education as the “single answer methodology.”

The Engineering Analysis and Design sub-committee called for a re-examination of *all* courses and curricula, to see how “evaluation and synthesis can be exercised in courses now primarily analytic.” They suggested giving students “exercises in creativity”: problems which have no “analytical or unique answer.” They recommended that these exercises be given to students at all levels of university education.

Sample problems were presented in a number of different specialties, not to be representative of all the engineering fields, but more to suggest a process. The sample problems encouraged students to invent mechanisms, plan city roads and project traffic patterns, and devise function circuit elements rather than have students analyze existing designs exclusively. The problems have a strong leaning toward “paper designs”: projects that can be completed and evaluated in the classroom, with little need for hands-on experimentation and construction of models. Thus, while the subcommittee believed engineering educators needed to do much better in developing creative ability in their students—a skill especially

---

<sup>5</sup>*ibid.*, page 80.

needed in design situations—they did not see an essential connection between practical experience with materials and the ability to perform design.

In March of 1959, L. E. Grinther published “A Survey of Current Changes That Are Modernizing Engineering Education,”<sup>6</sup> a follow-up report which summarized development since his committee’s significant work earlier in the decade. This report presented the now unambiguous trend in engineering education: more theoretical content in core technical and mathematics courses, and the “dropping of courses in engineering practice, art or other technical areas.” The courses in engineering science or analysis and design which replaced the practice and art courses were heavy on theoretical technique, rather than hands-on practice.

A number of comments made by faculty who responded to Grinther’s request for information during the preparation of the report were presented anonymously. A sampling of these short quotations illustrates the conviction behind the changes that were taking place:

“We believe that the future education for engineering in the university should be based on the engineering science concept, rather than on the traditional emphasis on art, technology, and skill.”

“The trend in the College of Engineering over the past five years has been to reduce the laboratory and to increase the engineering science offerings.”

“We have improved the stature of our course in engineering drawing while decreasing the amount of time devoted to the subject.”

“There has been a decrease in emphasis on application courses.”

“Our Civil Engineering Department has thrown out several arts type courses, has increased its mathematics content to that required in all other engineering curricula, namely, through differential equations.”

“Our 1958 curricula as compared with the 1953 version is basically marked by a considerable change from the ‘how to do it’ type of course to one more soundly based on mathematics and science.”

---

<sup>6</sup>*The Journal of Engineering Education*, volume 49, number 7, pages 559–572.



### 1.1.2 The MIT Report on Engineering Design

As early as 1961, the deleterious effects of this trend were beginning to be known. At the Massachusetts Institute of Technology, a Committee on Engineering Design studied the nature of engineering design activity, and assessed the state of engineering education at MIT and other universities (M.I.T., 1961). The Committee concluded that one of the goals of engineering education, in addition to the more obvious goal of providing students with a body of knowledge and skills germane to the solution of engineering problems, should be the development of a set of attitudes and habits that are important to the designer. Quoted from the report, these attitudes are the following:

1. Willingness to proceed in the face of incomplete and often contradictory data and incomplete knowledge of the problem.
2. Recognition of the necessity of developing and using engineering judgment.
3. Questioning attitude toward every piece of information, every specification, every method, every result.
4. Recognition of the experiment as the ultimate arbiter.
5. Willingness to assume final responsibility for a useful result.

The Committee raised serious concerns with the predominant pedagogy in effect in universities, which it termed the “single-answer question” method. This method, prevalent at the time of the report and still today, teaches by having students complete single-answer problems. As described in the Committee’s report:

Questions asked of students are overwhelmingly what will be called single-answer problems. This classification includes all problems which can be answered with numbers or functional relationships; in fact it includes all problems which have answers that which can be generally be agreed upon.

The Committee pointed out why the single-answer methodology is so popular: it is easy to grade; it is easy to teach particular methods to solve such problems; graduate students and other non-experts in engineering can teach courses based on this methodology. The Committee expressed deep concern about this pedagogy, however, based on the attitudes it had deemed important to the overall abilities of an engineer. In particular, it raised the following concerns about the effect of teaching predicated on the single-answer method:

1. Incomplete or contradictory data have little place in single-answer problems.
2. Engineering judgment is not required of either the student or the instructor, hardly a situation to encourage its development.
3. The very existence of an objective standard puts the instructor in an almost impregnable position, which only a few of the very bright students will dare to challenge. Skepticism and the questioning attitude are not encouraged by this situation. Neither the data, the applicability of the method, nor the result are open to question.
4. The single-answer problem usually suggests the infallibility of logic rather than the ultimate word of experiment.

The Committee concluded, “It seems clear that the single-answer question has a rather strong negative effect on attitudes we hope to teach our students.”

### **1.1.3 Computer Assisted Instruction**

The report fell on deaf ears. In the 1960’s, engineering educators became enthralled with the promise of computer-assisted instruction (CAI) as a means of improving students’ learning. In an article presented in a 1969 issue of *The Journal of Engineering Education*, Lawrence Grayson sums up the domain of uses envisioned by educators at the time:

Computer-assisted instruction is here understood to mean the use of computers on a time-shared basis to perform any instructional function—presenting material or problem situations, guiding a student’s thinking by answering his questions, assessing his performance, managing his path through a course by selecting the material to be presented or assigning tasks to be performed away from the computer, or any combination of these . . . With so broad a definition, it is necessary to differentiate the ways in which computers have been used, since the various aspects of CAI are in different stages of development.<sup>7</sup>

### **1.1.4 Contemporary Trends**

The CAI initiative did not yield the revolutionary changes its proponents anticipated. Recently, there is a much greater recognition within the engineering education community

---

<sup>7</sup>Lawrence P. Grayson, “Computer-Assisted Instruction and Its Implications for University Education,” *The Journal of Engineering Education*, volume 59, number 6, February, 1969, page 477.

of the need to provide design experiences to its students. Still, much of the methodology of the education has remained unchanged; most of the innovation to respond to this problem has been the creation of design courses with the specific agenda of teaching design.

In a survey of contemporary design classes at universities nationwide, as reported in the literature, two broad categories of classes are seen. The first is known as a “capstone design course.” It is a course given to students at the end of their undergraduate careers—i.e., seniors. These courses are characterized by presenting formal design methodology and having relatively solid justification in the university’s overall curriculum—they are well-established in the university community. Examples of this type of course can be seen in references (Jolda, Barber, & Lane, 1991; Magleby, Sorensen, & Todd, 1991; Rashid, 1991).

These courses might vary such parameters as the length of time involved (one or two semesters), the use of team projects versus individual projects, and having extended projects (on which a given student completes only a small part of the overall design) versus complete (start-to-finish) projects. However, all of these capstone courses share one feature: they are provided to the student who is at the end of his or her undergraduate education.

The reasoning for this placement in the curriculum is not often made explicit, but when it is, the explanation proceeds with a supposition that only at this point in their careers will students have the formal analytical skills they will need in order to competently perform design. This is really a special case of the more widespread belief implicit in much of formal education that students must “learn about” before they can “do.”

Most capstone design courses are concerned with the most practical issues that will face the student who graduates and goes to become a practicing engineer, and therefore teach design as it is believed to be encountered in an industrial setting. In this way, the university teaches design as a separate topic, rather than integrating design activity more generally into the learning activities engaged in by students.

The other type of design course is not categorically identified in the literature and hence does not have a name, though it might be called the “creative design workshop.” It is characterized by placing an emphasis on student creativity, teaching less formal design methodologies, and allowing a high degree of innovation and personal choice in student work. These courses are usually the “pet projects” of particular faculty members of the

universities, and often have an eccentric approach toward teaching design—focusing, for example, on topics such as bicycle design (Klein, 1991), amateur radio design (Anderson, 1991), or piezoelectricity (Breger, 1989). The university standing of these classes seems fragile, not justified in explicit terms.

It does seem evident that the potential impact a design course could have on a student’s intellectual development is lost when the student is given the experience at the end of an academic career, rather than as an integral part of one. In fact, in a recent proposal, the ABET organization is proposing a significant change in its requirements that will push universities to have integral design experiences in their programs, rather than having a separate design course requirement (Prados, 1992).

## **1.2 Design Research**

It is not surprising that there are a variety of opinions in the education community about the role of design in learning, for there is deep contention within the design research community about the nature of design activity itself. This discussion has immediate relevance to educators concerned about the role of design in education.

### **1.2.1 Design as a Formal Process**

It has already been noted that one of the reasons design activities were displaced from university curricula is that design and the study of design have been perceived as too informal and unscientific to be worthy of a place in a modern university. Not surprisingly, one line of research in the design community is the formalization of design—the making of a “science of design,” as it is put by Herbert Simon in his book, *The Sciences of the Artificial* (Simon, 1969).

Simon is one of the most influential of the design formalists. In a chapter of his book entitled “The Science of Design,” Simon calls for the university to re-embrace the study of design, not in the “soft, intuitive, informal, and cookbooky” manner it might have been known, but in a reformulation that emphasizes formalizable aspects of the design process: optimization algorithms like linear programming, control theory, and dynamic planning,

heuristic search, and search for best research allocation.

Simon's work has roots in ideas of the Artificial Intelligence community, and was written during a time when that community was openly confident that deep issues of human intelligence and thinking would soon be solved. As such, Simon's recommendations, which focus on formalizable and algorithmic aspects of the design process, seem logical.

Yet Simon and many others who continue this line of work take their arguments the further step: they argue that these optimization methods and other formal treatments circumscribe the whole of the design act. This is the belief upon which Simon pins his call for universities to reinstate design as a part of the science and engineering curriculum. If design is indeed a formal science, he is correct.

## 1.2.2 Design as a Negotiational Activity

Other design researchers and design historians see very different processes at work in designer's minds. Donald Schön, a design researcher, studies designers at work in a variety of domains—engineering, architecture, management, and others. An important theme in his work is the idea that designers continually deal with underconstrained, negotiational situations in which the design goal has yet to be specified in formal terms. As Schön explains, this task challenges the “technical rationality” upon which design formalism is based (Schön, 1982):

Technical Rationality depends on agreement about ends. When ends are fixed and clear, then the decision to act can present itself as an instrumental problem. But when ends are confused and conflicting, there is as yet no “problem” to solve. . . It is rather through the non-technical process of framing the problematic situation that we may organize and clarify both the ends to be achieved and the possible means of achieving them.<sup>8</sup>

Even after the end goal may be clearly understood, the process of then embarking on the “doing the design” is not simply a matter of choosing the appropriate algorithm or design technique and applying it. As Schön describes:

---

<sup>8</sup>*The Reflective Practitioner*, pp. 41.

There are more variables—kinds of possible moves, norms, and interrelationships of these—than can be represented in a finite model. Because of this complexity, the designer’s moves tend, happily or unhappily, to produce consequences other than those intended. When this happens, the designer may take account of the unintended changes he has made in the situation by forming new appreciations and understanding and making new moves. He shapes the situation in accordance with his initial appreciation of it, the situation “talks back,” and he responds to the situation’s back-talk.<sup>9</sup>

In this way, Schön is in essence arguing that professional designers proceed into a learning process as they engage in the act of designing—learning at least about the situation at hand, but possibly also about a deeper level of the nature of their domain of expertise as well. This is a special, privileged type of knowing, a “knowing-in-action” as Schön puts it, that cannot be simply reduced to formal terms.

### **1.2.3 Epistemological Pluralism**

In research with computer programmers, Sherry Turkle and Seymour Papert argued the need to acknowledge that these designers have distinct intellectual styles, which they called either the “planner” or the “bricoleur” (Turkle & Papert, 1990):

The bricoleur resembles the painter who stands back between brushstrokes, looks at the canvas, and only after this contemplation, decides what to do next. For planners, mistakes are missteps; for bricoleurs they are the essence of a navigation by mid-course corrections. For planners, a program is an instrument for premeditated control; bricoleurs have goals, but set out to realize them in a spirit of a collaborative venture with the machine. . . While hierarchy and abstraction are valued by the structured programmers’ planner’s aesthetic, bricoleur programmers prefer negotiation and rearrangement of their materials.

Turkle and Papert called for an “epistemological pluralism”—a recognition that the bricoleur’s style is a valid design methodology, not a stage in an intellectual progression toward the “formal” planner approach:

Our observations suggest that with experience, bricoleurs reap the benefits of their long explorations, so that may appear more “decisive” act like planners

---

<sup>9</sup>ibid., pp. 79.

when they program on familiar terrain. Also, of course, they get better at “faking it.” Still, the negotiating style resurfaces when they confront something challenging or are asked to try something new. Bricolage is a way to organize work. It is not a stage in a progression to a superior form. Interviews with computer scientists and their graduate students turned up highly skilled bricoleurs, most of them aware that their style was “countercultural.” Indeed, there is a culture of programming virtuosos, the hacker culture, that would recognize many elements of the bricolage styles as their own.

Turkle and Papert’s studies focus on the activity of computer programming, both as it is experienced by programmers and as it is presented as a cultural force, but their discussion applies to many aspects of engineering and design as it is typically presented in the university setting.

#### 1.2.4 Design Communities

Not only do individual designers learn through designing, but so do groups of engineers as a community. Walter Vincenti’s case studies in aeronautical history reveal important technical developments as both a collective design process and a communal learning process (Vincenti, 1990). One of his examples illustrates how the development of flying-quality specifications for American aircraft required a collaboration among designers, engineering researchers, instrument developers, and test pilots:

The problem [of flying-quality specifications] was initially *ill* defined—the engineering community did not know at the beginning of our period what flying qualities were needed by pilot or how they could be specified . . . that community learned to identify pilots’ needs and translate them into criteria specifiable in terms appropriate to the hardware . . .<sup>10</sup>

This example is but one of many that can be used to show that when individual designers or communities of designers are confronted with unfamiliar territory or new situations, they don’t solve problems with formal design techniques. In these cases, the design process is not reducible to an application of formal methods. As described by Ferguson:

Engineering design is always a *contingent* process, subject to unforeseen complications and influences as the design develops. The precise outcome of

---

<sup>10</sup>What Engineers Know and How They Know It, pp. 51; original emphasis.

the process cannot be deduced from the initial goal. Design is not, as some textbooks would have us believe, a formal, sequential process that can be summarized in a block diagram. Starting with a block called “Need,” such a diagram—which may comprise from a dozen to more than a hundred blocks—purports to guide (or at least to follow) the designer through the process of inventing and analyzing a new thing. Block diagrams imply division of design into discrete segments, each of which can be “processed” before one turns to the next. Although many designers believe that design should work this way, even if it doesn’t, it is clear that any orderly pattern is quite unlike the usual chaotic growth of a design. The vision at the heart of a design is often in a designer’s mind long before a need has been articulated. Second thoughts are admitted in the block diagrams along the “feedback” paths, but the reader should understand that the steps in the design process may all be going on at once.<sup>11</sup>

Louis Bucciarelli, an engineering professor who has observed engineering designers at work, points out the difficulties that arise when groups of engineers attempt to follow the formal design process charts that are often imposed on the design process (Bucciarelli, 1988):

Despite the appearance of continuity and order conveyed by the charts, the generally articulated perception of time is that there is never enough of it. All the machinery of scheduling, forecasting, and systems analysis can never fully define, hence control, the future. Uncertainty, no matter how comprehensive or detailed the charts become, is always in the air. Indeed, there is a limit on how much designing of design is worthwhile. The planning effort presents the same problems it is intended to solve.<sup>12</sup>

Bucciarelli also notes the crucial role of informal and unplanned social interactions in the design process:

Decisions come in forms other than hard and formal. A happenstance gathering in the hallway, a background conversation at a group meeting, an intense dialogue at a farewell party, and the like can be settings for decisionmaking . . . These “soft” decisions define the context of design, fix what alternatives and ideas will be entertained, which will be considered laughable or ignorable.<sup>13</sup>

---

<sup>11</sup>*Engineering and the Mind's Eye*, page 37.

<sup>12</sup>Louis Bucciarelli, “Engineering Design Process,” in *Making Time: Ethnographies of High-Technology Organizations*, edited by Frank Dubinskas, page 108.

<sup>13</sup>*ibid.*, page 111.



### 1.2.5 Design in University Education

This conflict over the nature of the design process is of much importance to educators wishing to endow their students with design ability, for they will shape their students ideas about the nature and quality of the design act as it “ought to be.”

A survey of design textbooks written for university-level coursework can indicate what sorts of ideas are being presented in undergraduate courses. Morris Asimow’s *Introduction to Design* (Asimow, 1962) is representative of a genre of work that presents design as a formal, algorithmic process. The book contains numerous exercises to insure that students learn the methodology and techniques presented, which no doubt are used in many industrial and research settings. One gets the impression that a course built around such a textbook would consist entirely of readings, lectures, and homework exercises.

In this attempt to teach students *about* design, one wonders if this sort of “design knowledge” is not unlike the formal problem-solving skills that, when taken in isolation, are of questionable pedagogic value. The capstone design courses at least go a step further, in which students learn design through the process of actually doing so.

Still, both approaches miss the opportunity to view *the act of designing as a learning process*. They assume that design is simply the act of applying one’s knowledge and skills to a problem situation to create a satisfactory solution. Yet there is ample evidence that professional designers are engaged in an important and fundamental learning process during design work. Schön’s catch-phrase of designing as a “conversation with the materials of a situation” alludes to the negotiational learning process that must be engaged in to become familiar with a particular design situation. The university should acknowledge these aspects of the designer’s work, and provide students with learning experiences of this sort.

### 1.2.6 Design at MIT

MIT as a university has a strong continent of design-rich courses, but these courses are not part of an overall plan to integrate learning about design or design-based learning into students’ education. Recently, the Electrical Engineering and Computer Science department established a formal design requirement, but it can largely be satisfied by courses already

in the core curriculum, which have become worth various amounts of design “points.” The requirement, while ostensibly recognizing the need for design-rich activities, imposes minimal impact on the structure of the undergraduate curriculum, and seems to skirt the issue of integrating design experiences into the undergraduate experience in a planned manner.

Nevertheless, many of MIT’s courses do provide valuable design experience. This section discusses several such courses, and is included to give the reader a sense of how the Robot Design project fits into the MIT undergraduate’s alternatives.

### **The Digital Design Laboratory**

In MIT’s Electrical Engineering and Computer Science department, the *Digital Design Laboratory* class (number 6.111) is unusual in that it is built around extended design projects of the students’ own choosing. The course format consists of a series of four laboratory projects, each increasing in complexity, followed by team design projects that take up the final one-third of the semester (Troxel, 1968).

The laboratory series is structured so that when the student has completed the four labs, he or she has gained practical experience in using the building blocks of modern digital design, including its lower-level components (e.g., AND gates and flip-flops) and its higher-level structures (e.g., finite state machines, programmable gate arrays, and micro-sequencers). With both conceptual and practical experience in using these elements of digital design, the student then embarks on an original design project.

A feature of the course is that it does not allow the use of microprocessors in the design projects (nor does it introduce them in the laboratory series). MIT Professor Donald Troxel, who created the course, explained that if you give students a programmable artifact, they tend to do a lot of programming, and that’s not the point of this course.<sup>14</sup> Students are encouraged to use certain lower-level types of programmable controller circuits, such as finite state machines and micro-sequencers.

Final projects are encouraged to be creative and different, and often involve some sort of evocative aesthetic component. Each year there are typically a number of musical projects

---

<sup>14</sup>personal conversation, 1992.

(e.g., audio samplers, synthesizers, a machine to read data from old piano scrolls) and video projects (e.g., video games, video telephones, video scramblers). In this way the projects often become avenues for students to explore or develop a pre-existing hobby or interest.

Students work in self-selected teams of usually two but sometimes three students on the final project. A course teaching assistant is assigned as a consultant and advisor to each student team. This mentor has several important roles in working with the students. First is ensuring that the project contains a proper amount of complexity: projects should neither be too simple, for they would not be challenging, nor too complex, for they would fail due to lack of time or debugging skill on the students' part. The TA's help the students organize the progress of their work, beginning with general block diagram designs, through implementation and concluding with debugging.

Perhaps most importantly, the TA's make sure that the students clearly specify the interface between each individual's part of the project. Rather than having the two or three individuals in a team work on all aspects of the project, the teams are organized so that each individual is responsible for a particular sub-section of the overall design (to which all members contribute); this allows students to be graded for their portion of the work fairly, and also forces the design to be done in such a way that large sub-sections of the project can be separately tested and debugged.

Students are expected to get their projects to work, though many fail to do so after two to three nearly continuous days of lab sessions near the final deadline. Many students find that debugging is harder than they expected: it's easier to conceptually design a circuit than it is to build it, and it's easier to build the circuit than debug it. Here the role of the TA in helping students formulate projects with the proper amount of complexity is critical: in general students lack hands-on debugging experience, and have a tendency to err on either side of what would be the proper level of project complexity. Grades are not severely affected by a lack of a full demonstration of the project if the TA makes the assessment that the design was completed and an honest attempt to get it functioning was made.

Among MIT students, the Digital Design course has the reputation of being a lot of work but also being a lot of fun. It is clear that the task of designing and building a complex system and (hopefully) getting it to work is an extremely satisfying experience.

## Introduction to Design

In MIT's Mechanical Engineering department, sophomore students are required to take the *Introduction to Design* course (number 2.70), but many students from other departments take the class because they want to.

The central activity in the 2.70 course is a design contest in which students build mechanical contraptions and pilot them in a confrontational contest event at the end of the course. (This is the course that served as an inspiration for the project upon which this dissertation is based, as discussed in the Introduction.)

The *Introduction to Design* course consists of a set of lectures and introductory activities loosely organized behind the premise of engaging students in the hands-on design project, which itself takes up fully the latter half of the semester. A “warm up” project serves to introduce students to machinery in the traditional mechanical engineering shop: the lathe, drill press, taps, dies, and other such paraphernalia. Some paper design projects require students to perform drawing skills, encouraging them to gain an appreciation for this aspect of the mechanical design process. Lectures consist of interesting but not clearly relevant presentations on trends in the mechanical engineering profession, safety concerns, and analytic techniques. Recitation classes allow discussions on design methods and approaches.

All of these are not so structured a preparation for the design project in *Introduction to Design* as is the laboratory series in *Digital Design Laboratory*. For example, many students do not make detailed mechanical drawings before building pieces of their project, though this is ostensibly encouraged by the inclusion of such activities early in the course. In fact, students are allowed to progress in their design of their contest machine as they wish, allowing a spectrum of styles from the cerebral's plan-plan-plan and then build, to the bricoleur's “play with the parts and see what emerges” approach.

In the end it is the hours spent building, thinking, testing, and designing in the shop which are the point of the course. Situated interactions with other students and faculty while working on one's project often become valuable points of learning. The contest itself is an event charged with excitement and anticipation, with four to five hundred in the immediate audience and an annual television audience of many, many more.

Students' design notebooks and other written work serve as the central mechanism for assigning grades. For student who did not "think on paper" while doing their design, the design notebook does not accurately represent the extent and depth of their involvement during the design process.

## **Computation Structures**

MIT's *Computation Structures* (course number 6.004), typically taken during the junior year of the Electrical Engineering and Computer Science program, centers around the construction and programming of a prototypical central processing unit (i.e., a microprocessor). Each student in the course builds and writes code for a standardized CPU as part of the coursework; the class evolved from a lecture/textbook/problem set course to include the CPU hardware aspect, an effective way of making the ideas of the course concrete, as well as providing a hands-on experimental component.

While students design small pieces of the CPU system as the course progresses, the overall design of the system is laid out at the beginning of the course. So while students learn *about* a complex system through the process of building its pieces and putting them together, they don't get the experience of *designing* that system.

It must be noted that the purpose of the 6.004 class is not to teach students design; it is to teach them about microprocessor architectures, a job that the course does extremely well. So this analysis of the 6.004 class is not a criticism of its intended shortcomings, but rather an observation and example of a hands-on project-based course in which students do some design, but not systems-level design.

## **1.3 Educational Technology**

The materials developed for the Robot Design project are derivative of the LEGO/Logo work done at the MIT Media Laboratory in the mid-1980's by Seymour Papert, Mitchel Resnick, Stephen Ocko, and Brian Silverman (Resnick & Ocko, 1990; Resnick, 1990). This work, in turn, evolved out of Papert's work on Logo, the children's programming language, which began in the 1960's. Papert's work took a deliberately opposite intellectual direction

from the work on computer-assisted instruction (CAI), which was just beginning to gather momentum at the time.

### **1.3.1 Constructionism**

Papert's motivation to create a computer programming language for children began when he was co-director of MIT's Artificial Intelligence Laboratory in the 1960's. Papert was a leader of the community of artificial intelligence researchers during a period of intense creativity and intellectual excitement. A.I. researchers had developed the Lisp programming language, and were engaged in writing programs to test, explore, and embody their ideas about the nature of human intelligence.

Papert wanted to bring this spirit of inquiry to the world of children. He believed that deficiencies in the mathematical fluency of children sprung from a paucity of challenging and engaging "mathematical stuff" in world of a child. In *Mindstorms* (Papert, 1980), his seminal book on the experiences of children programming with Logo, Papert makes an analogy between trying to learn a foreign language (say French) in the classroom versus learning it in a place where it is spoken (France). It's easy to learn to speak French while living in France because speaking the language is a natural, contextualized activity with real-life relevance—if you want to eat an ice cream cone, you must ask for it in French! In contrast, speaking in a classroom is based on role-play conversation, which cannot have the same personal relevance. Papert hoped to create an environment—the Logo programming language—where children could work with mathematical ideas with the same personal meaning as is speaking French in France.

As the Logo language was developed, it came to have at least two characteristics that distinguished it from other contemporary computer programming environments. The first was its interactivity, which it shared with Lisp, the language that Logo was based on. When a child was sitting in front of a Logo console, he or she could type a Logo command and the computer would execute it immediately. This was a completely different sort of interface to a computer than what was typical in those days, namely, batch mode programming. With the Logo approach, a budding programmer could immediately see the result of an interaction with the computer. This encouraged a whole different style of work on the

computer, one that was more oriented to exploration and play—children’s natural ways of thinking—than abstract symbolic thought.

The other feature of Logo projects was that they expanded the realm of the computer beyond data manipulation. Most computer interactions, including early Logo projects, were based on some sort of data transformation. Numeric or textual data would be processed by the computer and the result would be displayed for the programmer. For example, a Logo program might take lists of English nouns, verbs, and adjectives and string them together into nonsense sentences—a word play experiment.

Papert and his colleagues began experimenting with robots connected to computers running Logo. Rather than being fixed arms or XY-tables with Cartesian geometries, these robots were mobile robots that could be understood with a relative, robot-centric geometry. Children would first play with robot movement using button-boxes to control the robot’s motion. (Figure 1-1 shows an early Logo robot and the direct-manipulation button panel used to control it.) Then they would use Logo primitives to control the robots, typing statements like `FORWARD 50` to make the robot move forward fifty “steps,” or `RIGHT 90` to make the robot turn in place ninety degrees. By making sequences of these movement commands, children could cause the robot to move around in specific ways, or even to draw geometric patterns on the floor, as a robot would carry a pen to mark the path it traveled.

Papert noticed that children’s interactions with the Logo robots had a quality that was different from the other projects based on data manipulations. The children were able to *relate* to the robots in a way that they hadn’t with the data projects: they could imagine themselves as the robot, and literally walk themselves through a Logo program by moving about as the robot would. Papert felt strongly that the way children were able to think about the robot by using their bodies, what he called a “body syntonicity,” was a key to making Logo accessible to children with a broader range of intellectual styles. More children would become engaged in projects based on robot control than data manipulation, because they were able to “think with their bodies” in doing so.

The Logo robot became a defining feature of the Logo environment. It was dubbed the “turtle,” since early robots were shaped vaguely like turtles, moved sort of like a turtle would, and also to give a children (and adult researchers for that matter) a playful and

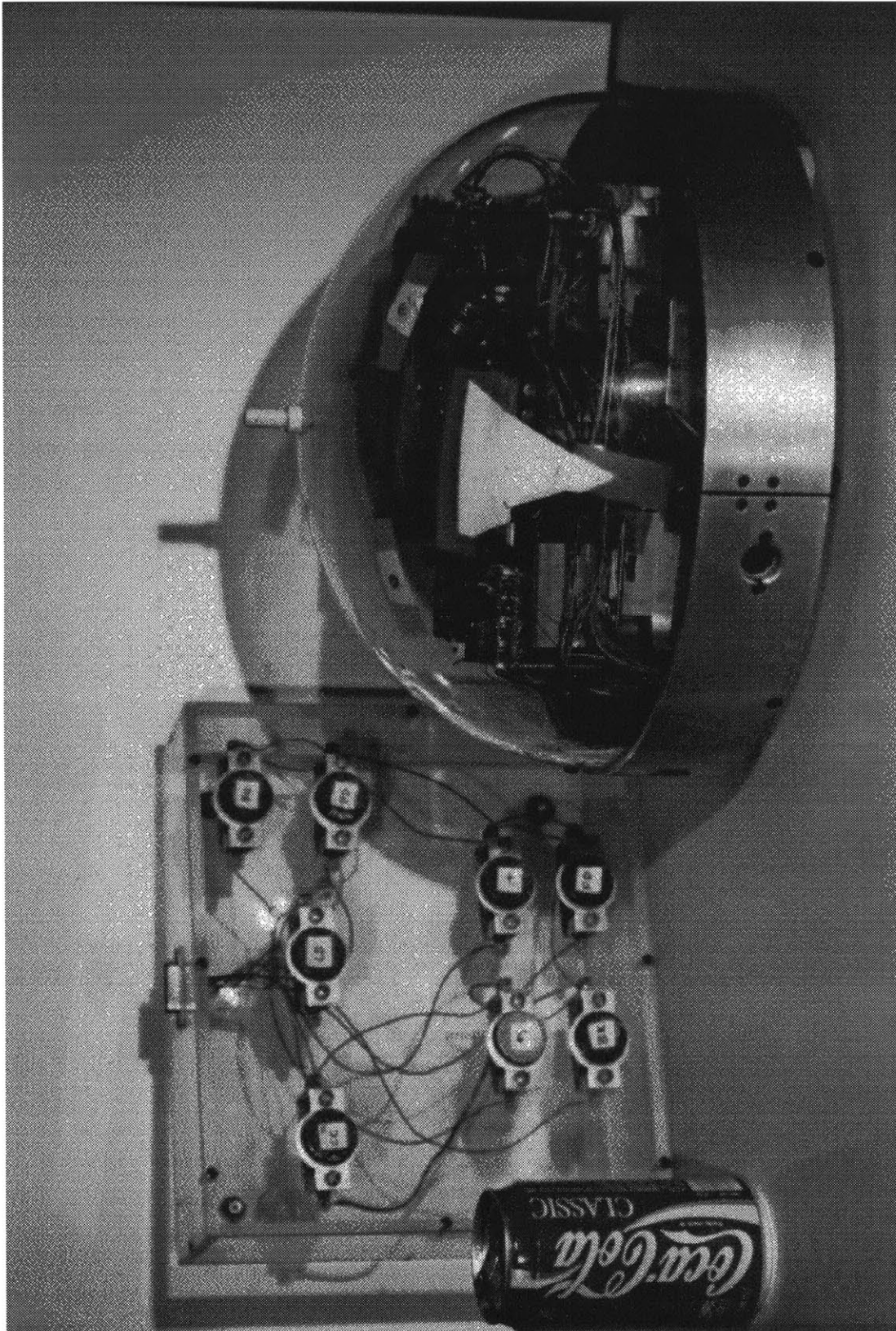


Figure 1-1: Early Logo robot and button panel for direct manipulation



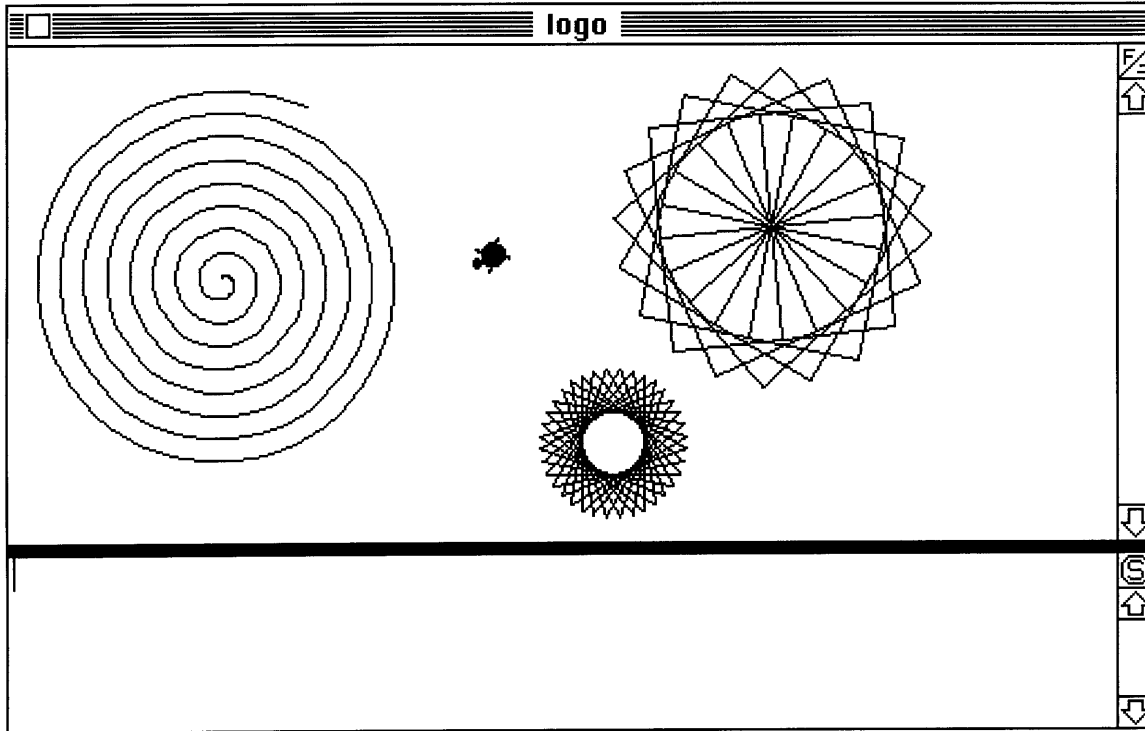


Figure 1-2: Logo turtle and typical Logo drawings made using turtle graphics

familiar object to hold in one's mind when thinking about how the device would behave.

As computers developed video display technology that replaced Teletype-based line printer interfaces, the Logo turtle moved “off of the floor” and “onto the screen.” This is to say that the physical electro-mechanical Logo robots were supplanted by iconic images of turtles on the video display screen. When a child gave a command to a screen turtle, it would move about and draw on the display screen rather than on the floor.

The screen turtles had certain advantages and drawbacks with respect to the floor turtles. Perhaps the most important advantage of the screen turtles was that they could be used on any computer with a video display, so they could reach many more children. Also, screen turtles could move very precisely and rapidly—there were no mechanical slippage problems—so children could easily create complex geometric displays (Figure 1-2 shows the Logo screen turtle and typical drawings made with it.)

On the other hand, screen turtles were more conceptually abstract than their floor turtle ancestors. Children had more difficulty understanding rotations from screen turtles—in some implementations of Logo, the turtle would snap immediately to its new position

rather than step through a series of rotations to accomplish a movement. Children couldn't get up and walk around a screen turtle. When the turtle were facing downward on the screen, children would often become confused about the meaning of "turning left" (counter-clockwise rotation) versus "turning right" (clockwise rotation).

Still, the screen turtles were a valuable invention that greatly accelerated children's ability to relate to computer programming activities. It was much easier for children to understand how to program a turtle to make a drawing than to use cartesian geometry, the "native" language of the computer hardware, as was typically available in early microcomputer versions of the BASIC language.

### **1.3.2 LEGO/Logo**

In 1971, Papert and colleague Cynthia Solomon published a short memo entitled "Twenty Things to Do with a Computer" (Papert & Solomon, 1971). Written in the days of the Teletype interface, this visionary paper suggested twenty computer-based activities that presupposed the availability of much richer user interfaces to the computational hardware. Several of the activities involved hardware Logo turtles, but several others imagined more sophisticated and versatile possibilities of the computer controlling other mechanical devices.

In the mid 1980's, Stephen Ocko and Mitchel Resnick, two researchers working in Papert's group, began experimenting with a electronic interfaces that would allow children to hook motors and sensors up to a computer running Logo. But there was an important difference between this work and the early Logo floor turtle experiments: with the newer work, children were able to not only write the programs to control electro-mechanical devices, but could actually build those devices as well. The vision laid out by Papert and Solomon's memo of fifteen years ago was finally being realized.

The LEGO/Logo project, as it became called, used a recently developed set of LEGO parts, named "LEGO Technic." The LEGO Technic set included not only the familiar plastic LEGO building blocks, but newer pieces like gears, beams, wheels, and motors, which enabled a LEGO builder to create animated and exciting mechanical projects. The range of devices that could be constructed with LEGO Technic was as wide as the imagination:

children created ferris wheels, toasters, elevators, walking robots, animated sculptures, and many, many other things.

Coincidentally, as the MIT researchers were building prototype interfaces that allowed the Logo language to control LEGO devices, the president of the LEGO company in Denmark read *Mindstorms*. Sensing a shared set of ideals about the role of children's play in learning and the value of constructive materials in children's hands and minds—whether they be the grammatic building blocks of the Logo language or the physical building blocks of a LEGO set—Papert's group and principals at the LEGO company arranged a meeting. A sponsored research project resulted, and Resnick and Ocko performed research and development that led to the commercialization of the LEGO/Logo system as a product for the educational market. (Figure 1-3 shows components of the system, including an MS-DOS computer, an interface box between the computer and the LEGO motors and sensors, and several representative LEGO models). The LEGO company has been selling the system, which they named *LEGO tc logo* ("tc" for Technic Control), since the late 1980's; currently, it is estimated that seven thousand schools in the United States have the materials and nearly one million children have used them.

### **1.3.3 The Programmable Brick**

Shortly after Ocko and Resnick had finished their work on what became the *LEGO tc logo* product, they began looking for new directions to extend the LEGO/Logo concept. One limitation of the commercial product was the matter that LEGO constructions needed to be tethered with wire to the electronic interface sitting aside the controlling desktop computer. The system tended to encourage the construction of stationary machines, like a merry-go-round, rather than mobile machines like a LEGO floor turtle. Researchers in Papert's group were also interested in exploring children's work with mobile creature-like robots that exhibited cybernetic characteristics.

Resnick and Ocko experimented with remote control technology in which the controlling computer broadcasted commands to a LEGO machine that carried an infrared- or radio-based receiver. The system was functional, but it too had its limitations: reliable, bi-directional communications (needed so the LEGO machine could report sensor data back

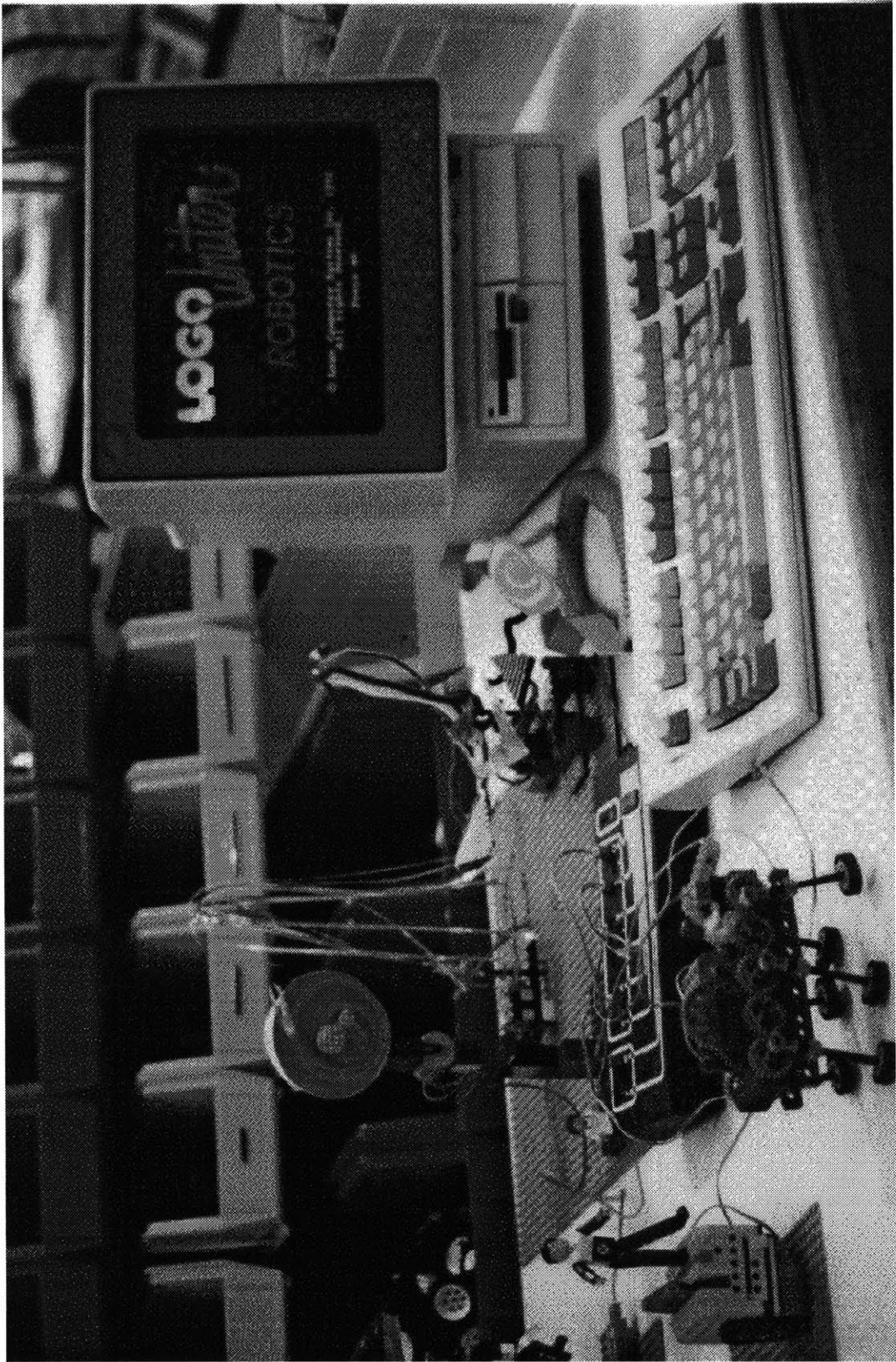


Figure 1-3: The commercial “LEGO tc logo” system marketed by LEGO Dacta USA

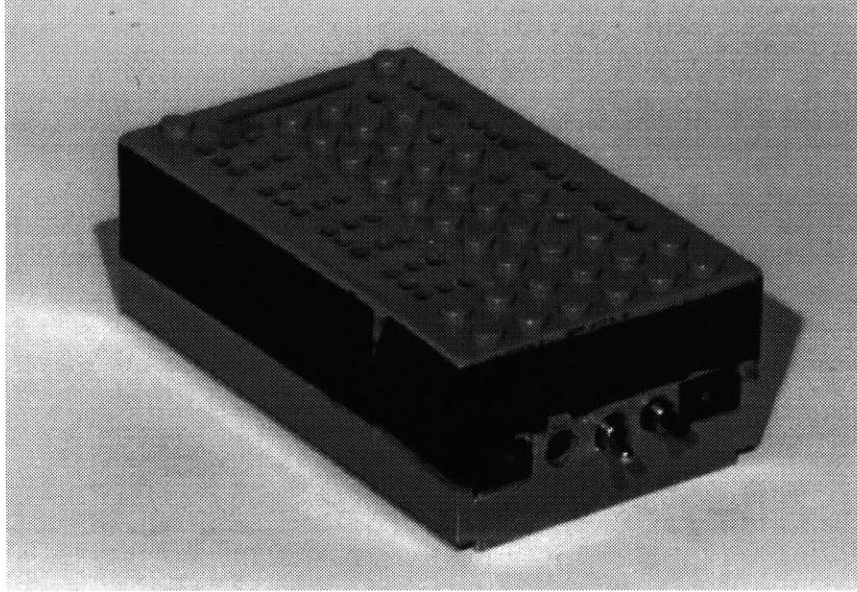


Figure 1-4: The LEGO/Logo Programmable Brick

to the host computer) were difficult to implement, and a system that would be suitable for classroom use, in which there might be a dozen or more simultaneous projects, would make the communications technology unwieldy. Additionally, it seemed like poor aesthetics for a big, desktop computer to be controlling a little LEGO machine. Why not build a miniature computer that could be embedded into the LEGO machine itself?

I joined Papert's team to assist in the development of the "programmable LEGO brick"—a hand-held LEGO box that contained an entire computer capable of running Logo. Two additional people joined the development team: LEGO engineer Allan Toft, who visited at our MIT lab for a year-long period, and Brian Silverman, chief scientist at Logo Computer Systems, Inc. (LCSI), who had done the software development of the Logo implementation used in the commercial *LEGO tc logo* product.

In about a year's time we created a prototype Programmable Brick and manufactured about a half-dozen copies of them. (Figure 1-4 shows a photograph of the Programmable Brick we developed). The Brick had outputs to control four LEGO motors, and inputs to receive data from four sensors. Existing LEGO sensors—a touch switch and a light sensors—as well as custom sensors could be used. The brick was based on a version of the 6502 microprocessor—the same device as was used in the Apple II series of computers, which were prevalent at the time. Brian Silverman ported the commercial version of Logo,

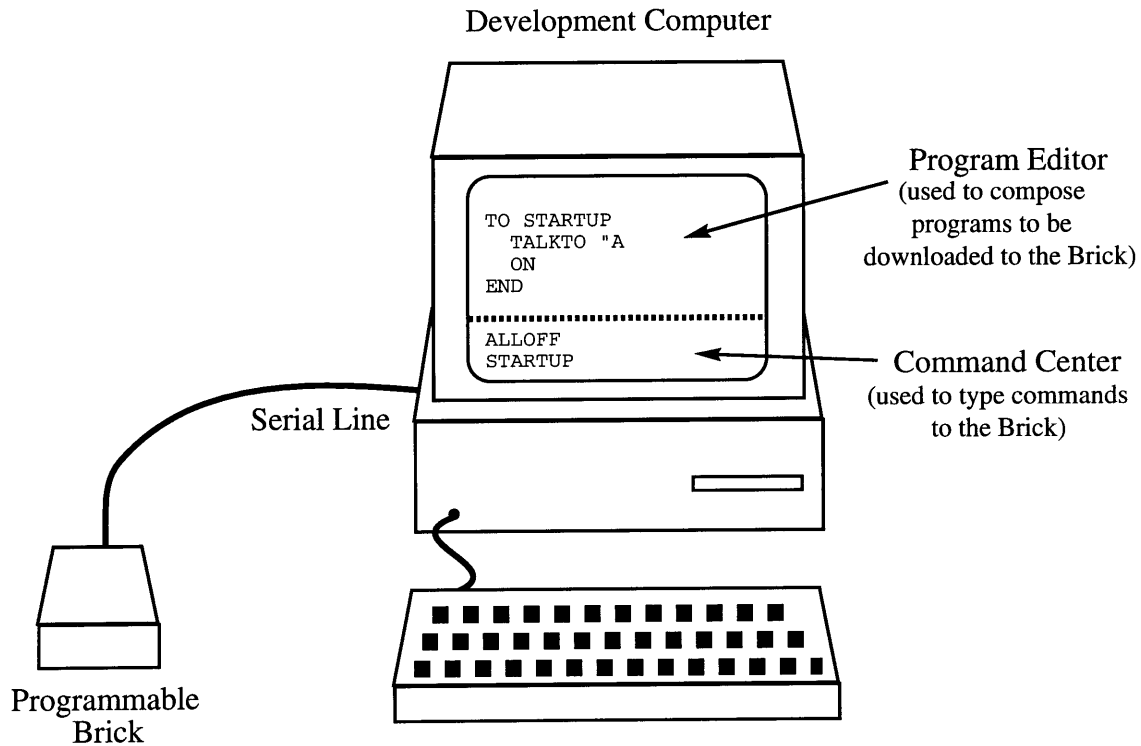


Figure 1-5: The Programmable Brick system

written for the Apple II computer, to run on our Programmable Brick.

Figure 1-5 shows how the Programmable Brick was used in operation. To program the Brick, it would be hooked up to a host computer (using a serial line connection). Then the user could type commands to the brick or download Logo procedures to it. After downloading, Logo procedures could be invoked by giving commands through the Command Center, or by pressing a button on the Programmable Brick, which would run a specially-named Logo procedure.

The user interface of the Programmable Brick system was based on the Logo interface which Brian Silverman and his colleagues at LCSI had developed for their commercial versions of Logo: the upper portion of the screen was conveniently available for editing programs while the lower portion of the screen served as an interface to the Logo interpreter. When the Programmable Brick was connected to its host, it functioned very much like the commercial *LEGO to logo* product, but with the additional capability that the Programmable Brick could be disconnected from the desktop computer to run programs on its own.

Researchers at the Media Lab, especially Professor Edith Ackermann, and I used the

Programmable Brick system with a small group of fifth grade students to explore ideas about cybernetics, feedback, and anthropomorphization; this work is described in (Ackermann, 1991) and (Martin, 1988). In using the Programmable Brick in these experiments, we had the opportunity to evaluate the design from a practical standpoint. The observations would later become considerations when we embarked on the design of similar technology for the MIT Robot Design project.

From a usability standpoint, the software component of the Programmable Brick system was a success. In a sense, this was a further validation of the usability of the same interface which LCSi had developed for its commercial Logo products: by keeping both the program editor and the command center on the display screen at all times, it was easy for children to use the Logo environment and move back and forth between writing and testing their Logo procedures.

However, the Programmable Brick had some hardware design deficiencies that ultimately became a significant liability. The internal memory of the Brick was lost whenever power was disconnected from the Brick (the computer brick used a separate battery brick), necessitating a procedure to reload the Logo language operating system. This clumsiness was compounded by a poor choice of the battery connector: when the power cable was jostled, the Brick would often seize up, requiring both the Logo operating system and the program the Brick was running to be reloaded.

The result of these design shortcomings was that the Brick could only be used if there were a "Brick expert" in the room. Our hope that the Brick could be used autonomously by a diversity of researchers and children was hence not realized, though it stood as a valuable model for a mobile robotics control technology that informed our work on the materials for the Robot Design project.

# Chapter 2

## Introduction

This chapter discusses the foundations of the research performed for this dissertation. The chapter is organized into four sections. The first section introduces the *LEGO Robot Design Competition*, the workshop class developed as part of this study. The second section discusses the educational goals and motivations that underlay the workshop's development. The third section presents the research methodology upon which the results are based, and the final section introduces the three central chapters of the work.

### 2.1 A Robot Design Competition

The model class environment that has been developed is in some ways an academic course, a workshop, an independent student activity, and a design competition.<sup>1</sup> The project has taken form as an activity held during MIT's Independent Activities Period (IAP), and is known to students both by its unofficial course number, "6.270," and its name, *The LEGO Robot Design Competition*. The project began as a programming competition modeled after MIT Professor Woodie Flowers' course for Mechanical Engineering students, *Introduction to Design* (course number 2.70). In Professor Flowers' popular course, students each receive a kit of scrap and surplus building materials and the specification for a task to be performed by mechanical contraptions to be designed by the students. Students spend the

---

<sup>1</sup>This project has been a collaborative effort resulting from the work of many individuals, including, most importantly, Pankaj Oberoi, Randy Sargent, and myself.



two-thirds of the term-long class designing and building machines to operate in the contest held at the end of the class (Flowers, 1987). Flowers' course is unusual in that it gives students wide creative latitude in the execution of their designs. Its high level of popularity amongst the MIT student body can be partly attributed to this feature, which allows the work of the course to adapt itself to each individual student's own learning style.

The format of the LEGO Robot Design Competition project is similar to that of Professor Flowers' mechanical design class. At the beginning of the project, students receive the entire robot-building kit, including plastic LEGO pieces for mechanical and structural work, parts for building a pre-designed microprocessor-based controller that will serve as the "brain" of their robot, electric motors, batteries, and parts to build robotic sensor devices. At the same time as they are given this kit, the specifications for the robot contest are presented. The students' task is to design, assemble, and debug a robot that will successfully accomplish the challenge presented by the contest specification.

The students are given a comprehensive set of course notes that explains how to use the specialized technology they were given in the robot-building kit; also, lectures and recitations are held to present and share ideas relevant to the design and construction of a viable robot. The most important learning, however, happens in small groups and the ever-present laboratory sessions, where students are engaged in building prototypes, testing their ideas, and sharing thoughts with each other and the project organizers.

At the end of the month-long Independent Activity Period, the contest is held. Students robots are on display in a competitive event that is witnessed by about four hundred members of the MIT community. It is an exciting spectacle with an atmosphere resembling that of the traditional collegiate football homecoming game. At the contests, students discover whether their robot can withstand the difficulties of performing in the "real world."

## **2.2 Relationships for Learning**

The development of the Robot Design project as an academic learning environment was guided by a set of pedagogical principles that together form a certain philosophy of education. This approach is founded on the constructionist theory of education espoused by

Seymour Papert (Papert, 1986, 1991). Constructionism is based on the premise that learners are best at creating knowledge and ideas inside their own minds when at the same time they are engaged in an act of building a personally meaningful artifact in the world.

As an elaboration of this principle, I would like the reader to consider a set of educational principles which underlay the Robot Design project as an academic course. The relationships formed by the participants in the Robot Design class—relationships with their work, each other, and the organizers of the project—are different from relationships formed in traditional academic settings. This section highlights these ways in which the Robot Design class differs from traditional academic environments.

### **2.2.1 Structure and Freedom**

In discussion of an educational setting, the word “structure” is often used as a barometer for the degree of guidance in the classroom environment. “Highly structured” situations imply that students’ work follows a narrow and precharted path; depending on the teacher’s pedagogical beliefs, this may be a good thing or a bad thing. It is usually assumed that such situations produce predictable and consistent results.

On the other end of the spectrum, we hear about “unstructured” learning environments, in which students are free to spend their time as they see fit on an on-going basis. It is generally presumed that the experiences of people in these sorts of environments are much more idiosyncratic than those in the highly structured ones.

Yet the word “structure” is really a misnomer for the sort of variation in different learning situations; one is really concerned with *freedom* and *accountability*. In the “highly structured” situation, the student has little personal freedom and is not terribly accountable for the result, since the learning experience has been precharted by the pedagogue. In the “unstructured” situation—which must necessarily have some sort of structure, however implicit it may be—the participant has a great deal of personal freedom, and also is personally accountable for the result, since the nature of the learning experience is of her or his own making.

Therefore, while the Robot Design project leans far to the side of personal freedom and self-accountability for the learning experience, this is not to say that it is “unstructured.” In

fact, it is anything but: it has its own consistent internal structure that specifically allows for learning to happen.<sup>2</sup>

In particular, there is the structure of the contest challenge provided to the students. The contest is a “problem space” which simultaneously is open-ended yet provides focus to the students.

## **Open-Ended Problem Spaces**

As noted by the MIT Committee on Engineering Design, the generally accepted method of giving students closed-form, “single answer” problems has a significant and detrimental effect on students’ ability to tackle complex and underspecified problems like those in the real world. It is therefore important to give students design problems which do not have a single right answer, but rather a multiplicity of possible good answers.

I prefer to call such situations a *design space* rather than a “problem,” to highlight the open nature of such a situation. A student then does not “solve the contest problem,” but rather constructs a model that satisfies the design space of the contest (to whatever degree of success).

As will be discussed in the Technology for Learning chapter, this premise became a driving concern in the development of the series of contest challenges.

## **Implicit Structure**

While the design space of the contest should be open-ended, it also must set limitations to focus the students’ intellectual activity. Much of the detail in the following chapter explain the way in which the contest specifications, the level of abstraction implicit in the hardware and software, and the nature of the robotics sensors all form a coherent set of constraints and opportunities that guide the students’ work.

---

<sup>2</sup>I would like to thank Aaron Falbel for his generous conversation with me on this matter.

### **2.2.2 Accountability**

The Robot Design project is designed so that students are on their own to organize their progress through their design project. The only hard milestones are the first day of the course, in which they get their kit, and the last two days of the course, which are the preliminary and final contest rounds. In between this period—i.e., the span of the entire project—students work at their own pace with little formal intervention from the course organizers.

The students are given some suggested milestones, and lectures and recitations presented ideas at what we hoped would be appropriate points along the path of the students' own progress. However, the overarching concept in developing the “course” as it were was to give students all of the materials they needed up front, and get them up and running with the specialized information they might need to know as soon as possible, in order that the control of their progress would be in their own hands. “Get them going and get out of their way” would be a way to summarize the approach. To accomplish this, we analyzed the shortest route between getting the kit of raw parts and getting over the barrier of understanding the basics of a robotic system—sensing, control, and programming. Then we charted a path of progress which would get the students to this point as soon as possible, after which they would be free to manage their own progress.

### **2.2.3 Versatility**

Because of the open-endedness of much of the course, students were free to embark on various sub-projects as they went about building their robots. Some of these became quite significant investigations as they became interested in various aspects of robotics technology. As such, the pedagogical model of the course exhibited an important versatility, adapting to the interests of students from a variety of academic levels and personal backgrounds.

Many of these design excursions will be discussed in other parts of this dissertation: the custom motor drivers built by a number of students in the *Robo-Pong* contest (Section 5.2.1), the burst of musical robots (Section 5.2.3), the custom sensor devices (Section 5.2.2), and the custom control systems (Section 5.2.4). Accomplishing these projects required some

mixture of traditional library research, experimentation with the materials that were already provided, and conversations with friends, course instructors, or other experts to get advice.

The framework of the course was to give out a bunch of interesting stuff to play with, along with interesting ideas to explore in doing so, and let the students “go play.” If some of them became interested in using a particular idea as a jumping-off point to delve into some associated matter—be it motivated by personal interest, a performance criterion for the robot design, or whatever—then it made the course even stronger if it encouraged them to do just this. Rather than holding students back, in effect saying, “Sorry, that isn’t a part of this course, you’ll have to do that another time or on your own time,” our project said, “Great! Glad to hear you’re interested in that. Here are some suggestions. . .” It is a sad fact that many learning situations are so constrained by their predesigned curriculum that there isn’t room to allow for individual interests like this.

#### **2.2.4 Teamwork**

There were two important ways in which students collaborated with other people during their design work. Not only was their relationship with each other marked by collaboration rather than competition, but so was their relationship to the project organizers characterized more as a peer relationship than as a teacher-student one.

##### **Among Students**

Participation in the Robot Design project was different from taking a traditional academic class in the important social dimension. In the Robot Design project, working in a team and living in a fascinated community was a significant part of the experience.

We encouraged students to explicitly think about their strategy for working together. Some teams planned to use a specialist approach, assigning work to each team member based on his or her previous background or expertise. Other teams planned for a generalist approach, in which each team member would share more or less equally in all aspects of the work. Usually teams shifted their strategies as the needs of the project and interests of the teammates became more concrete. For example, a mechanical engineering student might

have been elected to perform the LEGO design work, but not long after receiving the kit, all of the team's members realized that they have an interest in developing LEGO designs. Or a team which started out sharing all parts of the work realized that time is running short and that they had better specialize in order to work more efficiently.

We provided students with little supervision in managing their teams, leaving it up to the individuals involved to work out mutually satisfactory arrangements. This resulted in teams that were quite successful, teams that were adequate, and teams that did not hold together. Many students found that getting along with their teammates to be one of the most challenging aspects of the whole project. Students had different strategies for working together on the design project, but for nearly all participants, the challenges and triumphs of working closely with one or two other people became a defining part of their overall experience in the project.

### **Between Students and Organizers**

On numerous occasions, the students of the project developed ideas in partnership with its organizers. This sort of relationship between teachers and students is not normally highlighted in the academic setting, and had an important impact on the engagement style of the participants. For example, during one of the projects, a student led tutorial sessions to teach students about a favorite sensor of his that we did not officially support; more recently, the whole project is run by past students who have become teaching assistants and finally organizers.

### **2.2.5 Community**

Beginning in the 1991 contest year, the robot-building software was set up to run on the campus-wide Athena computer network. Since workstation clusters are located not only in various locations on the main academic campus, but also in off-campus living groups (i.e., dormitories and fraternities), students did not need to come to the electronic laboratory and computer cluster to work on their robot projects—they could do so in their living groups.

This happened to some extent in the 1991 contest year, but much more so in 1992. This

change is attributable to a difference in the format of the course notes between these two years. In 1991, the course notes were handed out in small segments during the progress of the course. For example, when students first got their kits, they received course notes covering only the contest specification and board assembly instructions. Later, they received handouts on building sensors and using the Interactive C programming language.

Beginning in the 1992 contest year, we were able to provide students with the comprehensive set of course notes from the very start of the class. Since students also received a complete set of electronic assembly tools with their kits, this meant that they could “set up shop” in their living groups and perform a large portion of the robot-building process there, including electronic assembly, LEGO design development, and programming.

As it happened quite a few teams opted to work in this fashion, especially teams that lived off-campus (across the Charles River) or in the west portion of campus (nearly a mile from the electronic laboratory). For these students, the trek to show up at lab wasn’t worth the effort if things were going smoothly, especially in January, the dead of Boston’s winter.

There was, however, an additional benefit that these teams received from working in their homes: the curiosity, interest, and participation of their housemates. Students described how their housemates would sit down and play with LEGO parts spread over a shared table, peer over their shoulders while they wrote computer code, and get excited watching the fledgling robots wander around the hallways. One year, three teams from one particular dormitory joined forces to build a replica of the contest playing table in their residence for their robot development efforts.

Not only did this type of forum provide students with valuable ideas and feedback, but it gave them a sense that their community cared about the work they were doing. In contrast to most academic endeavors which consist of studying concepts in isolation, this activity was one which they could relish in sharing with their peers.

## **2.2.6 Motivation**

Students’ participation in the Robot Design project was purely voluntary. As was mentioned, the project takes place during MIT’s Independent Activity Period (IAP), a one-month session between the fall and spring semesters. Student participation in all of IAP opportunities has

historically been on an optional basis.

## **Voluntary Participation**

IAP began as an academic experiment in the 1960's, with the purpose of giving MIT students an alternate experience to the traditional university term. IAP has evolved into a veritable smorgasbörd of learning opportunities: faculty, staff, and students organize and run sessions of widely varying formats to introduce both their vocational and avocational interests to others. An IAP activity may consist of a field trip, a lecture series, an all-day workshop, an evening outing, or just about any other format that activity organizers may conceive of and activity participants may find interesting.

It is out of this venue that the Robot Design project was born and bred; presently, the Robot Design activity is by far the largest IAP activity of all. It accepts over one hundred and fifty students, and expects each of them spend between twenty and thirty hours a week on the Robot Design project.<sup>3</sup>

## **Role of the Contest**

Later in this dissertation, the role of the contest in structuring the students' design experiences is discussed. Here, I would like to briefly consider the impact of the contest as a social event.

As a performance event, the contest allows students to display their robots—the fruits of their labor—to the MIT community in a public setting. Because of this, it's natural for students to make an extra effort to make their robots ready for public display, just as anyone puts extra work into preparing his or her work for some sort of performance—be it a concert, talk, or stage play. Students take pride in their creations, and want them to operate properly on the night of the contest, both for their own satisfaction and to impress

---

<sup>3</sup>I have some qualms about whether the all-consuming nature (in terms of time) of the Robot Design project is appropriate for an IAP experience. As an undergraduate who participated in several IAP's, the smorgasbörd aspect of it was one of the most important for me: that in a month I could partake of perhaps a dozen different learning experiences. Thus I am in part saddened that students who participate in the Robot Design project must necessarily give that up, since the Robot Design work consumes so much of the month's time. I reconcile this conflict by keeping in mind that participation in the Robot Design project is voluntary, so even though students who participate have a different sort of IAP experience, it is still a self-chosen one.



the audience.

Yet a competitive performance is not the only way that a public showing can be staged. It's a natural question to ask if a contest event is an appropriate way to conclude an educational activity; perhaps some students do not feel fully comfortable in an activity that is in some sense evaluated by naming "winners" and consequently "losers." This is an issue that we kept firmly in mind when assessing students' experiences in the class.

We did not want students to become overly serious competitors for two reasons. The first is that this attitude would be most detrimental to those who wished to simply enjoy a friendly, low-key competition. The second is that this would lead students to work in isolation rather than in collaboration with each, and would have a negative effect on their learning potential.

In light of this, we took several actions to encourage friendly rather than ruthless competition. We discontinued a policy of having substantial prizes (e.g., VCRs, scientific calculators, and cordless telephones) for the contest winners that was inherited from the Robot Design project in its first years. We established a rule stating that robots could not intentionally harm one another, encouraging the competitions to be more like sporting events based on finesse and speed (e.g., tennis) than brute strength and machismo (e.g., American football). We encouraged students to take a light-hearted approach to the competition and share ideas rather than working in isolation; in our experience, we told them, teams that were present in the lab, sharing ideas and discussing problems, were more likely to have a working and reliable robot than those that worked on their own, hoping to develop a "world beater" strategy that would catch the other designs off-guard.

We were mostly successful in creating an atmosphere of friendly sport. While there were always a few teams who put winning ahead of most other aspects of participation, generally speaking this was not the case. While some teams developed their robots in isolation, in an attempt to surprise their competition, most students worked in a public setting. Others developed robots that were "accessorized" in such a way that would be hard to imagine that winning were the primary concern—for these students, creating an interesting project with a design aesthetic meaningful to them was the overriding concern.

An important feature of the contests was the fact that they provided an objective

performance evaluation for the students to base an analysis of their work. This is in contrast to events like the talent show, where the rating of a panel of judges is the evaluation method; this can hardly be considered objective in the scientific sense. For our Robot Design contests, however, the judges were more like game officials whose task it was simply to assist in procedural matters of running the game and ascertaining that the rules were adhered to. For the cadre of MIT students involved, it became apparent that this sort of game had great appeal—both from a standpoint of participation and observation.

Because the contest required an *actual performance*, not simply a theoretical or proof-of-concept one (e.g., “There; it worked once, now I’m done”), interesting issues were forced to the surface during the design process. Problems like reliability, endurance, and overall robustness are things that students would not have faced, by and large, in developing a project that did not require both a public demonstration and one that had hard-and-fast performance goals. Further, it was these issues in particular that became central places of learning for many of the project participants: demonstrating that something worked “in principle” was a familiar task, but proving that it worked in repeated practice was something altogether different. In this manner, the project gave them experience comparable to design activities in the real world.

The motivational aspect of knowing that one has an excited audience to receive one’s work should not be underestimated. In recent years of the course, all of the project participants knew how popular the contest events were, and there is no question in my mind that this served as an inspiration for doing one’s best. Even when students might realize that their machine is not likely to be the overall contest winner, students still have the desire to get the machine to work as they intended it to, and thereby be a winner in their own minds.

## **2.3 Methodology and Evaluation**

The methodology used in this work is based on an iterative cycle of design, testing, and evaluation, as represented in Figure 2-1. In the first stage, a hypothesis about the essential characteristics of a design-rich learning environment is formed. As was mentioned, this hypothesis was based on Papert’s theory of constructionist learning.

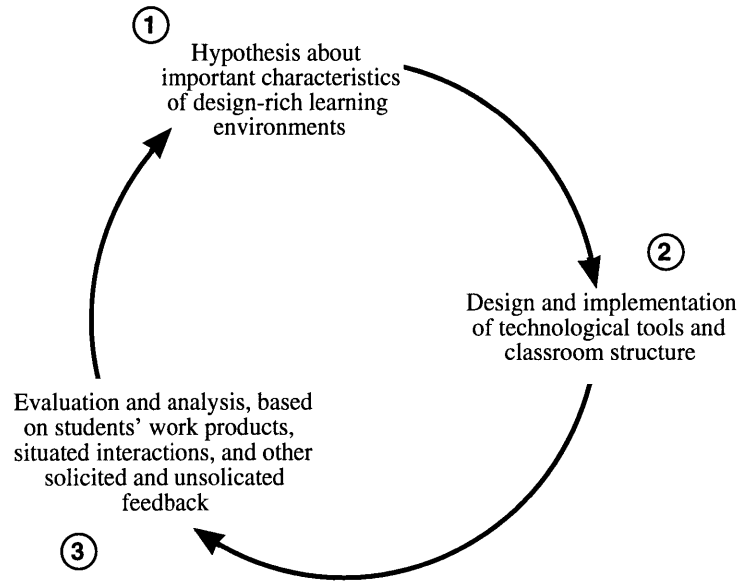


Figure 2-1: Cycle of project design and assessment

In the second stage, this hypothesis guides the formation of both a set of technological tools for learning and the classroom environment in which they are deployed. In this work, technology to facilitate exploration in the areas of engineering, design, sensing, and control was developed.

In the third stage, these materials are tested, evaluated, and analyzed. A variety of observational tools are used to form the evaluation: situated interactions with students during the course of their design projects, student journals and other solicited comments, unsolicited comments, video reports, and interviews, and analyses of the students' actual work products—robotic designs that reveal structural, software, and strategic decisions.

We have completed four iterations of this cycle of design and evaluation, corresponding to the four successive versions of the Robot Design course which were developed as part of this dissertation. In each of these versions, the impact of the materials provided to the students upon their learning was assessed, and these analyses fed into the design of the materials for the subsequent course.

The question of how to accurately assess what a person has “learned” through the progress of a particular experience is a difficult one. Traditional education bases its evaluation upon students' performance on a series of written tests, which are usually either of the “take-home” (i.e., problem set) or in-class (i.e., examination) variety. If a student is

proficient in his or her performance on these measures, then it is assumed that he or she has learned the material that is the object of the class.

This model is problematic from a constructionist learning theory point of view. The written examination is only one of what could be numerous “output modalities” for a student’s knowledge; in the traditional model, this mode is so heavily weighted in value so as to exclude others. Put another way, there is not a straightforward mapping between what a student knows and what he or she is able to express on a written test.

Further, this focus on written evaluation throughout the educational system forces the implication that all important knowledge or skills can be represented in this form. This in turn implies that any knowledge which cannot be represented this way is not worth knowing (at best, it is not worth evaluating). Yet it is precisely these alternative skills and ways of knowing—what Schön refers to as the tacit knowledge embedded in a designer’s process of work—which are the locus of investigation here. In other words, I am less concerned with what students can write or say about what they have learned than with a textured understanding of the difficulties they have faced, the styles of work they established, and the problems they have posed and solved for themselves.

These assessments of students’ activity are based on data collected from several means:

**Situated interactions with students.** During the course of their projects, I interacted with students extensively, answering their questions, helping them solve problems, and otherwise listening to their thoughts and concerns. This data was essential for a detailed understanding of the thought processes of individual students. As noted earlier, this style of interaction was more as between peers than as between teacher and student; often the problem puzzling the students was one we had not yet solved ourselves.

**Student journals and weekly video reports.** Weekly journals and self-guided video reports kept by students recorded the progress of their designs. This data was useful in understanding the broadness and commonality of students’ experiences.

**Concluding video interviews.** Most teams were interviewed near the end of the course in a relaxed setting. This data was useful for recording students’ own descriptions of

their experience (which did not necessarily match with other data used to observe it).

**Analysis of student projects.** Records kept on the students' designs—including copies of their control software, photographs of their machines, and discussions of the designs in the journal reports—is analyzed to infer the sorts of problem situations the students framed for themselves during the course of their designs, and how these challenges were resolved.

This data is used as the basis for forming arguments about learning, but also serves as feedback into the cyclical process of course design mentioned earlier. In this regard, the inferences made from these observations are either strengthened or challenged by seeing the results obtained from applying the working results into the course design. Thus, the final arguments to be presented in this work are based on their value as tested in the course itself.

It is worth mentioning that students often cannot verbally explain their learning process. In the concluding interviews, there were a number of students who were quite animated and involved during the progress of the course—excited about the new ideas they were experiencing and learning. But when it came to the interview, they were at a loss for words, unable to describe what had been valuable or interesting to them. Similarly, many students found it much easier to discuss the strategy their robot used to solve the contest than the strategy *they* had employed to create this robot.

For this reason I cannot overemphasize the importance of the contextualized interactions I had with students in my role as a course organizer. It was when working with students on problems they were dealing with at the particular moment that I gained the greatest understanding of the issues they faced and how they shaped the problem situation for themselves.

A related matter is the fact that sometimes learning happens over a long period of time. In one of the case studies discussed in Chapter 5, a student's work is tracked over a period that spans about fifteen months. It is only at the end of this period that the student in question revises his fundamental beliefs, after a series of demonstrated failures. The message of this example is that students' ideas are deeply embedded and it takes time for

them to change; sometimes the failure of one robot is not proof enough.

## 2.4 The Main Themes

The section introduces three themes that are the subjects of the subsequent three chapters.

### 2.4.1 Technology for Learning

Chapter 3, entitled “Technology for Learning,” explores the development and impact of the technology created for the Robot Design project. Taken in a broad sense, “technology” includes not only the tangible hardware and software materials that were given to the students, but the specifications of the robotic contest challenges. The chapter explores both the types of problems students encountered in their robot-building activity, and the progressive way in which we gained a deeper understanding of the structure that underlay the design environment of the robot-building task.

Several themes emerge from this study. The first of these relates to the way in which the contest game specification affected students’ work. We found that it was difficult to balance our desire for an open contest specification, which would allow many different types of robotic solutions, with a focused specification, which would encourage students to make machines that really worked.

Another issue relates to the properties of the technology as an educational media. Specifically, characteristics like *interactivity*, *modularity*, and *transparency* were found to have a profound impact upon students’ learning.

A third theme arises from unexpected difficulties in effectively using sensor devices. In two cases, complications in using sensors put the project participants and the organizers together in collaborating to find a way to use the devices. This “leveling of the playing field” had a valuable influence on the relationships of all of the project participants.

## **2.4.2 Ideal and Real Systems**

In Chapter 4, entitled “Ideal and Real Systems,” the students’ robots are analyzed from a control systems point of view. Of particular interest is students’ recurring inclination to build robots that will perform properly only under ideal conditions. Students repeatedly build robots that are not well-equipped to deal with the exigencies of the real world, but rather with the specifications of an idealized, abstractified world, a world that they would like to believe is a close representation of reality. This result points to limitations in the set of ideas about technological systems and methods that comprise the core of the engineering curriculum; what surprises many participants is that these ideas do not map well to the challenge of designing a robot to play in one of our contests.

## **2.4.3 Design Styles**

In Chapter 5, entitled “Design Styles,” I explain how the course attempted to encourage a bottom-up, playful style of design—the “bricoleur” style described by Turkle and Papert. We found that many students gravitated toward this style in the mechanical design aspect of their robotics work, but returned to the apparently more familiar top-down style, which one student described as the “computer science mind.”

This chapter documents a variety of design excursions, mini-research projects students performed as offshoots of their robot-building work, and ways that students re-defined the goals of their participation to suit their own interests. These projects illustrate the flexibility of the project in stimulating genuine participation from its students.

The data from this chapter is used to argue that regardless of students’ approach toward design, they encounter the central engineering design problem of reconciling the phenomena that they observe with the formal models of engineering science that they have learned. In some cases, it is easy for them to see the underlying principles, but in many it is not.

# Chapter 3

## Technology for Learning

This chapter presents the development of the technology used in the Robot Design project. An important concern throughout the development process was the notion that we were *designing for designers*: that the materials and scenarios we were creating were to be used by others to build with and upon, and to learn from as they worked as designers. Here, I explore the motivations, experiments, and assessments of the impact of the materials we created.

While this chapter is explicitly concerned with implementation details of the project at hand, that of a robot design competition, I believe that the lessons learned go beyond this particular domain. I encourage readers with other passions to consider how the methods and considerations I discuss here can be applied to your own areas of interest.

The technology of the Robot Design class can be categorized into three areas: contest specifications, control hardware/software, and sensor devices. Each of these categories influences the others in various ways; as illustrated in Figure 3-1, they form a triangle of interrelated concerns. Certain preoccupations guided our design of the materials in each of these areas:

**Contest Specifications.** Rather than setting a specific problem to be solved, the contest lays out a broader *design space* which shapes the students' engineering experience. To encourage students' creativity, the contest should encourage a variety solutions by providing multiple paths to viable solutions. By allowing students' robots to



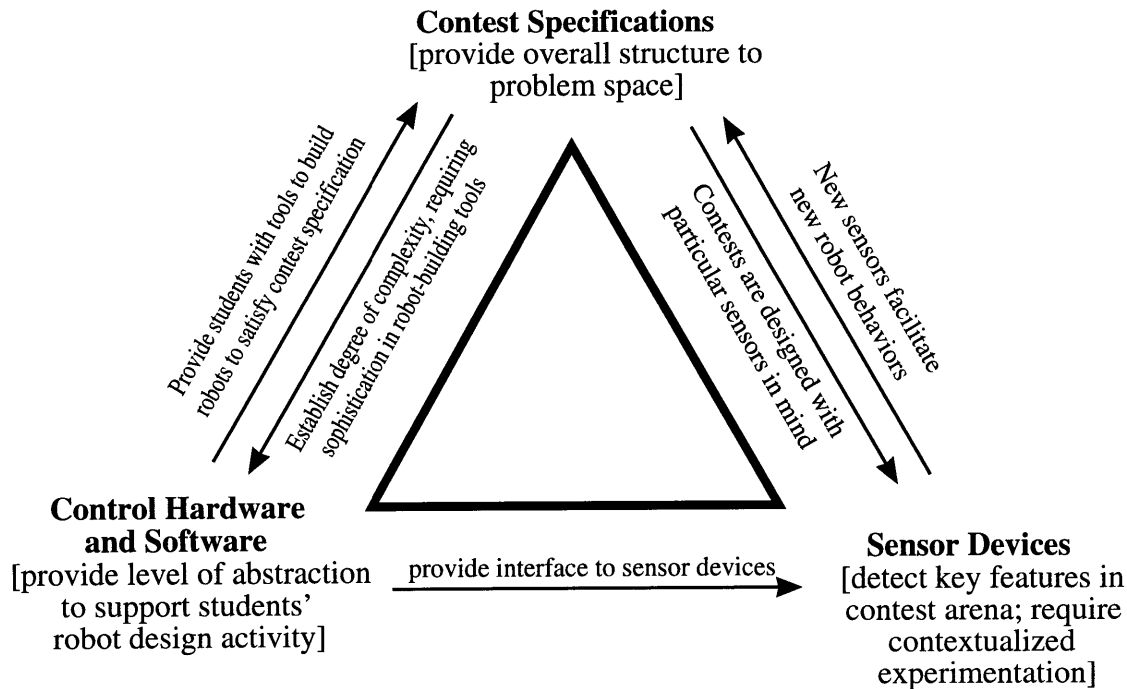


Figure 3-1: Triangle of technology: contest specifications, computer control hardware and software, and sensor devices

interact with one another, it may be possible that no single approach is the best. Some solutions may perform better against some opponents and worse against others.

Contests should promote a positive social message, and should strive to be inclusive of different personal styles.

Contests should provide the proper amount of intellectual challenge. No one is served by a problem that is too difficult to solve, and likewise a puzzle that is too easy may not bring out the best in those who attempt to solve it.

**Control Hardware and Software** The physical robot-building materials provided to the fledgling robot designers determines a *level of abstraction*. We provided a working microprocessor circuit, a high-level programming language, and a versatile mechanical construction kit so that students were able to focus on the strategic and conceptual aspects of robot design, rather than getting bogged down in low-level implementation issues.

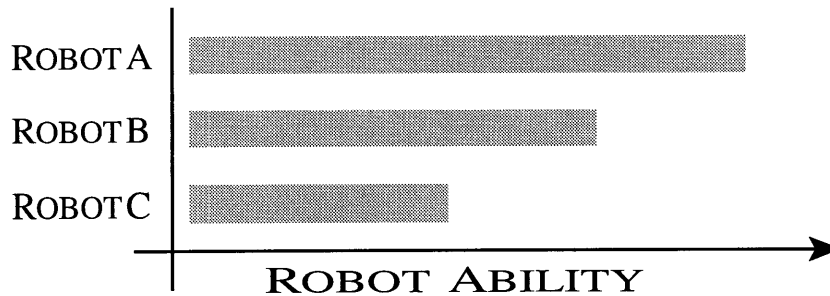
Determining the proper layer of abstraction is not a trivial issue; things should be

sufficiently “messy” so that creativity is encouraged rather than stifled. We found that the best balance was struck by providing “glass boxes” rather than “black boxes”: abstraction layers that students could peel away and look inside of when they desired. We wanted to encourage students to design interactively with the materials— to use bottom-up strategies—and we made considerable effort to make our materials easy to use, encourage experimentation, and require a minimum of unnecessary conceptual overhead.

**Sensor Devices** Sensors were one of the most conceptually challenging aspects of the technology. Students found them confusing; we found them ripe with potential. Being necessary to allow robots to perform any sort of interesting behavior, sensor development was a critical part of the progress of our work; in many regards the sensor development process was a microcosm of the development process for the project as a whole. Nearly all of the sensors we used led to instructive, unforeseen consequences that were rich in learning for both the students and ourselves (from both technical and pedagogical standpoints).

The three sections of this chapter that follow present each of these aspects of the course technology. The discussion that follows makes explicit the process of design, evaluation, and hypothesis-making that characterized our work. The reader will find that some parts of the story are retold three times, as through the lens of the designer of each of these three aspects of the technology. Rather than being redundant, I hope that this approach will illuminate the inter-relatedness of choices that were involved in the development of the materials, while allowing a presentation organized by the categories of the materials themselves.

This chapter presents a lot of detail that may not seem immediately relevant to the reader. I believe, however, that this detail is necessary in order to present the ideation process and reasoning behind decisions that were made, and to illuminate the lessons that were learned. The reader who is less concerned with the pedagogical role of these designer’s materials may wish to skip ahead to the concluding section of this chapter.



In the solo performance type of contest, a robot's ability can be reduced to a single parameter. Thus, if Robot B defeats Robot C, and Robot A defeats Robot B, then Robot A will necessary beat Robot C.

Figure 3-2: Visualization of solo strategy relationships

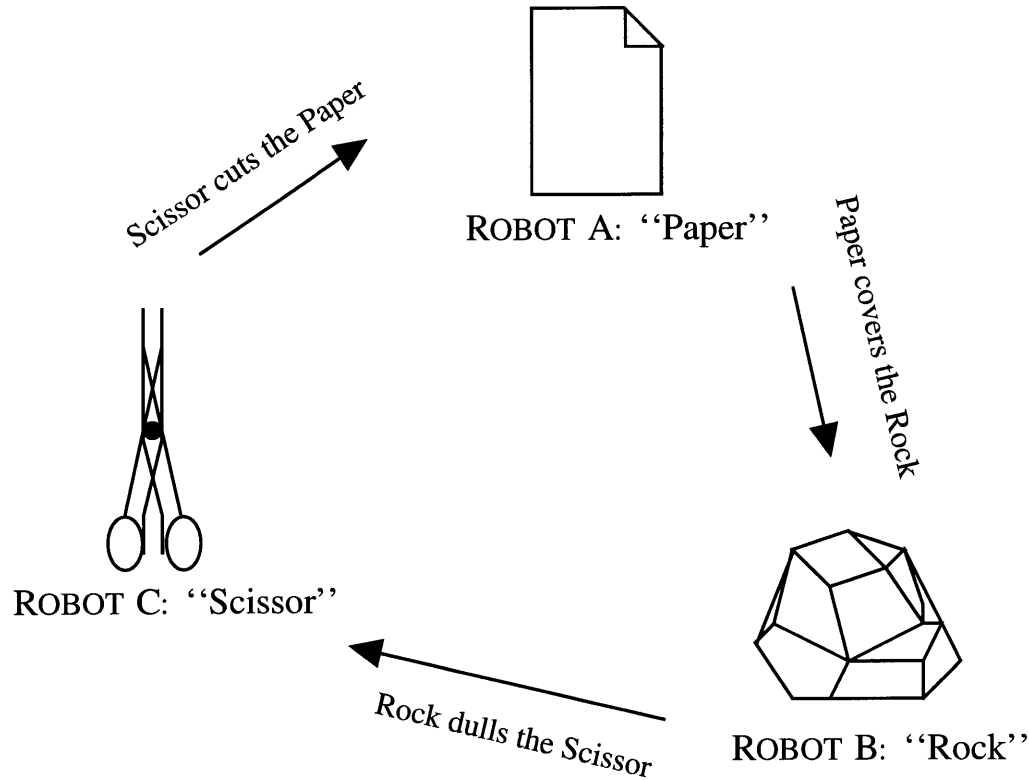
### 3.1 Contest Design

The design of the contest is a critical aspect of the learning environment because, perhaps more than any other factor, the contest sets out the scenario or design space that students will explore as they build their robots. If the contest is underconstrained, students will have difficulty focusing on essential aspects of the robot's functionality; if it is overconstrained, creativity will be stifled and projects as well as students' learning will suffer as a result.

The contest designer must keep firmly in mind the categories of problems that will be encountered, and hence must be solved, by those who participate in the contest. I prefer to call contest problems "situations" or "design spaces": problems which should be solvable given the materials available, but which are not obvious or one-dimensional. Many different types of solutions must be possible to any given contest situation.

There are two basic types of contest: individual performance runs and cooperative events. The cooperative events can, in turn, be divided into competitive and collaborative performances. The important distinction here is that individual performance runs represent a static, more well-defined problem statement, while cooperative events are more dynamic and unpredictable, since one cannot predict the actions of the other robots in the contest scenario.

The maze contest typifies the solo performance type of contest. Obstacle courses are another example. In this contest variety, performance is generally measured in terms of



In a competitive performance, each robot has its own unique strengths and weaknesses. Even if Robot A beats Robot B, and Robot B beats Robot C, it does not mean that Robot C cannot beat Robot A.

Figure 3-3: Visualization of interacting strategy relationships

least time or proportion of the contest that is successfully completed, reducing a robot's performance to a single parameter (Figure 3-2). While it may be the case that multiple solution strategies are viable, these strategies do not interact. Therefore, if strategy A beats strategy B, and strategy B beats strategy C, then strategy A is superior to C, by definition.

In a collaborative or competitive event, however, performance is determined in relation to one's collaborators or competitors. Each robot has various strengths and weaknesses, and there might not be a single best solution. As in the children's game of Rock, Scissor, and Paper (Figure 3-3), while the Paper can beat the Rock (by covering it), and the Rock can beat the Scissors (by blunting it), the Scissor can still win against the Paper (by cutting it). This idea applied to robot contests makes for a more exciting and diverse contest. Each design concept created by the students has its own validity, since it's not the case that any

one approach is obviously the best.

There are additional concerns that the contest designer must consider. To be a valuable learning experience, a contest should not be so difficult as to invite abject failure from even a small portion of its participants. This would discourage participation or leave an unsuccessful experience in the minds of many. Yet, a contest should not be so obvious as to suggest the same solution strategy to all; it should be sufficiently difficult to encourage and reward innovation and creativity.

To encourage fair play and an atmosphere of friendly competition, a competitive contest should be carefully designed so as not to allow an unsophisticated brutish design from bulldozing over all other competitors. If the competition is to have meaning, however, cleverly designed aggressors should be rewarded. (The reader will notice a tendency to return to the issue of aggression on several occasions in this chapter. I consider this issue important for personal reasons—I don't believe in aggression as a way of solving one's problems—and also because in these robotic face-offs, unintelligent robot interaction can easily stifle creativity.)

If a competitive contest is new—it's never been attempted before—then no one, including the contest organizers, know which sort of strategies are most viable or how various strategies may interact. The matching of respective designs leads to the greatest surprise. In this situation, those who are producing the contest have no special knowledge about what might work best; they are curious and inquisitive just as are the players. This encourages the free exchange of ideas between the project participants and its organizers.

Finally, a contest should be enjoyable to watch when played and should inspire fair play from all concerned. The game should be intelligible to the audience, so they can respect the diversity of the strategies on display, and engage their emotions in the game play, thereby rewarding the students' hard work.

A brief introduction to the progression of contests developed for the Robot Design project will serve as an orientation for the discussion and analysis of contest design parameters that follows. The reader who is interested in a full presentation of contest designs is referred to Appendix B.

Figure 3-4 summarizes the progression of the Robot Design contests from its inception,

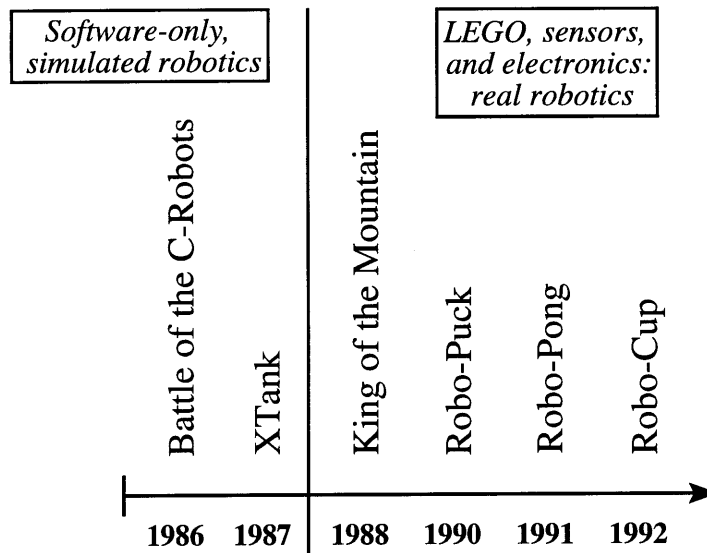


Figure 3-4: Year and name of robot design contests

in 1986, through end of this study, in 1992.<sup>1</sup>

**Battle of the C-Robots.** In this first year, and the subsequent year, the contest consisted of a software-only programming challenge in which a master computer program simulated the environment of the robots in the fashion of a video game. The *C-Robot* contest challenge consisted of writing a computer program to locate the other students' programs and shoot them. Thus, the video game characters were controlled by students' programs rather than being controlled by a game player's dynamic hand-eye coordination.

**XTank.** The second year of the project was in the same style of the first, however the video simulation was much richer.

**King of the Mountain.** This first of the hardware-based contests consisted of the challenge of building a robot to climb to the top of a large paper-mâché mound, in the fashion of the children's game that goes by the same name.

**Robo-Puck.** In this contest, three robots at a time competed to gain possession of a

<sup>1</sup>During the first two software-only contests, I was involved as an observer (*C-Robots*) and as a robot-programmer (*XTank*). My work as a contest designer begins with the *King* contest, and I include these brief observations about the earlier contests for completeness and to note their relevance in shaping the later ones.

physical hockey puck (which was modified to include an electronic infrared light source).

**Robo-Pong.** Building on the sports theme, *Robo-Pong* pitted two robots in a contest to transport ping-pong balls onto the other robot's side of the table.

**Robo-Cup.** Also based on ping-pong ball manipulation, each of the two *Robo-Cup* players had to extract balls from a feeder and deposit them into their respective miniature soccer-like goal, in an attempt to score more points than the opponent.

### 3.1.1 Strategic Diversity

A key to creating a rich learning environment lay in providing a contest specification that while giving guidance was open enough to encourage innovation. By inspiring students to create a multitude of approaches to solving the contest, we conveyed the message that there isn't one right way to approach a problem, and stimulated students' sense that originality and creativity are valuable engineering skills.

#### Robot Configurability

As early as the *XTank* contest, we saw that a successful contest allowed multiple solution strategies and a diversity of approaches. Even though this contest was purely software-based (the "robots" competed on a computer-simulated maze-like terrain), students were given wide latitude in the specification of their tank's properties, which led to adoption of different play strategies.

Students competing in the *XTank* contest were allowed to spend a certain number of "dollars" on parts to build their tank. This included choices like the size of the engine, the type and amount of armor, and the types of weapons the tank would carry. A typical tradeoff might be choosing expensive armor, which is light and would allow a tank to remain quite maneuverable, versus inexpensive armor, which is heavy and would impede a tank's agility (but would leave money available for other uses).

An example will illustrate the way that students personalized the flexibility of the *XTank* criteria. The object of the *XTank* contest was to destroy the opponent tanks, but one

participant took a non-violent approach to participation. He chose the lightest, smallest, and most maneuverable vehicle available (the “motorcycle”) and devoted all of his programming effort to the task of *avoiding bullets*. His strategy was successful—no other tank was able to shoot down his motorcycle. Since the contest rules awarded one point to each tank that survived the battle round (tanks also got a point for each opponent they destroyed), it was conceivable that his non-violent “tank” could place in the top few competitors overall. Unfortunately, his program had a bug in which the motorcycle would crash into walls and destroy itself. Still, it was never shot down in battle, and it did win survival points on a number of rounds.

Subsequent contests, based on physical LEGO-and-electronic robots, gave the students much greater freedom with respect to the configuration of their robots. Students were allowed to assemble the provided components in any fashion they desired. Even though they were more or less restricted to the components we provided in our robot kits, the more significant constraint was their own ability to configure these materials into viable mechanisms within the time allotted.

### **Simplicity versus Specificity**

The contests varied substantially in complexity, with the earlier contests being the simpler. Part of the motivation for increasing the complexity of the contests was the development of successively more versatile robot-building materials which supported students in developing more complex robots. Additionally, there was the desire to “outdo” our previous work each year. This trend, as will be noted, was not always successful in coaxing the best performances from the students.

The next several pages describe robots produced for each of the physical robot contests, illustrating how the contest specification led to varying degrees of variety in the students’ robots. As mentioned, the other factor affecting the robot designs was the capabilities of the robot kit that we provided; the impact of the kit is discussed later in this chapter (Section 3.2). Here, I examine the way in which the contest specification led to variation in the students’ work.



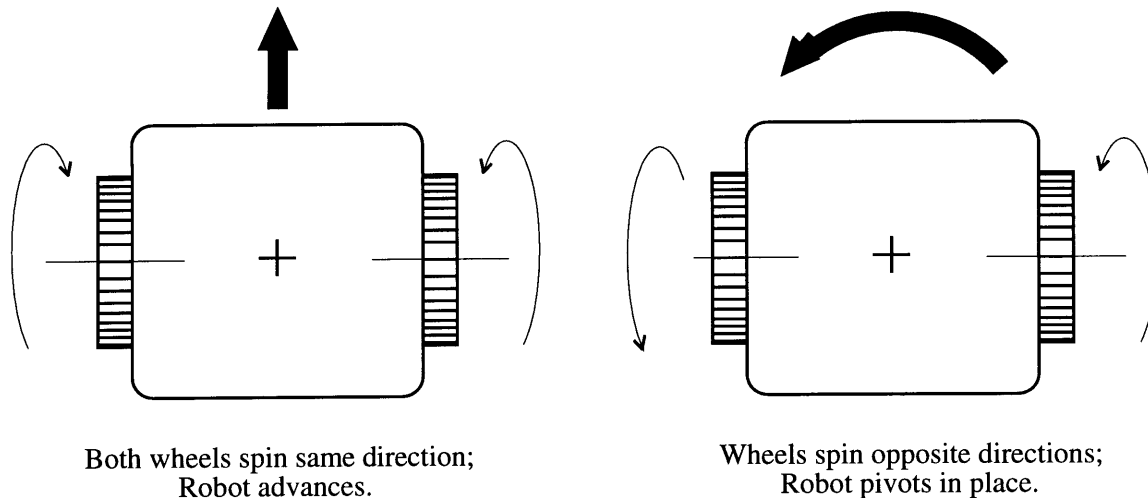


Figure 3-5: Schematic view of robots employing wheelchair drive configuration

**The King of the Mountain Contest** In the first year of physical materials, the *King of the Mountain* contest provided a simple challenge: build a robot that can climb to the top of a hill-like surface. The contest was deliberately kept simple because our technology was primitive, and the results reflected the difficulty of getting any robot at all to function. With a couple of exceptions, there was little differentiation among the final robots.

As we anticipated, nearly all of the robots had a “wheel chair” style drive configuration (Figure 3-5); this is the simplest and most effective drive configuration for a small mobile device. The main parameter that was varied within this design was the height of the robot’s body. Some of the robot-builders didn’t anticipate the effect of having a high center of gravity when the robot would be positioned on the incline of the hill; many of these “tall” robots were unstable and toppled over easily.

**The Robo-Puck Contest** The *Robo-Puck* contest was also simple, but the increased power of our technology led to greater variation in the students’ robot designs. In *Robo-Puck*, three robots played on a six-foot diameter circular rink and attempted to gain control of the puck, which was initially placed in the center of the rink (Figure 3-6). Students invented a variety of approaches to solve the *Robo-Puck* challenge. Though there was a preferred approach to solving the contest, the contest inspired a variety of different solution strategies.

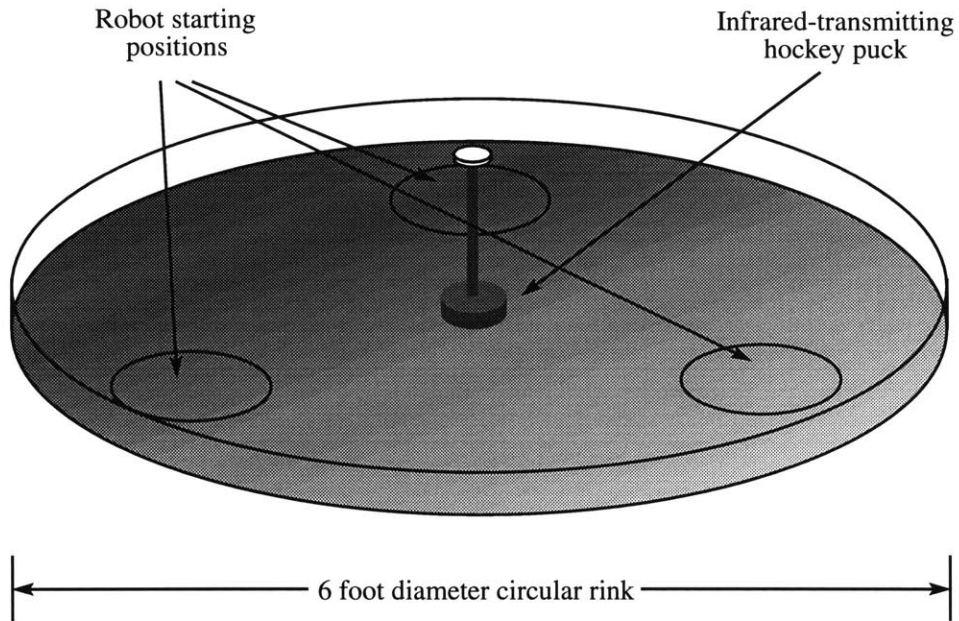


Figure 3-6: Playing table for the *Robo-Puck* contest

The most common design was a *puck-fetching robot*. In this design, the robot drove toward the puck and attempted to capture it. Several techniques were used to “grab” the puck: some robots had actuated arms that would close around the puck; others drove over the puck to bring it inside the robot’s own body; several had non-actuated arms that would trap the puck when the robot pushed it to a retaining wall. Any of these designs were potentially the simplest to implement, depending upon the complexity of the capturing mechanism. *Bertha*, a successful puck-fetcher design, is depicted in Figure 3-7.

One team created a dual-robot puck fetching design. One smaller puck-fetcher was nested inside another larger one. The plan was for the smaller, faster robot to capture the puck, whereupon the larger, slower robot would capture the first robot! (The infrared light from the sensor would not be shielded by the first robot, so the second one would just home in on it in the same way.) Unfortunately, neither of this pair of robots worked on the night of the contest.

Another team produced a robot named *Shotgun*, which fired a retractable claw at the puck. The claw closed around the puck and then the robot reeled it in, drawing it away from the opposing robots. This design was able to gain control of the puck much faster than any other robot, but it was not fully reliable: sometimes the claw missed the puck, and

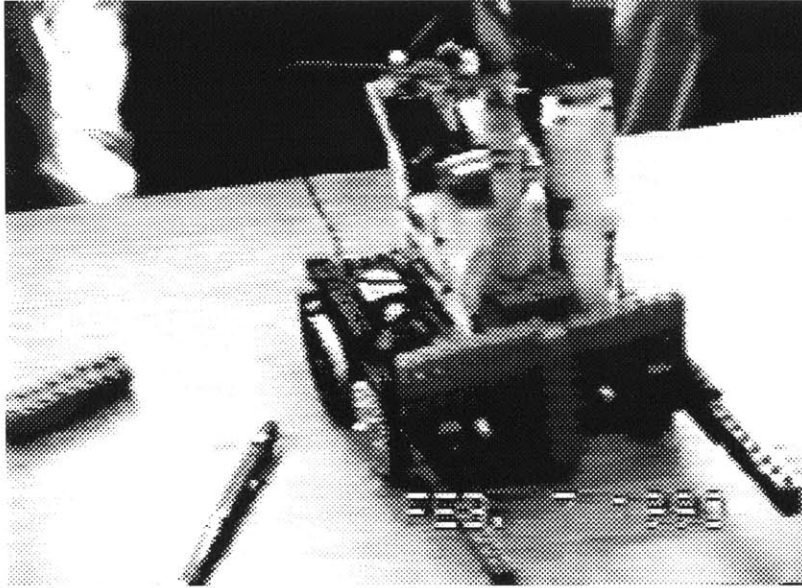


Figure 3-7: *Bertha*, a successful puck-fetching robot from *Robo-Puck*

the robot had no way to recover from, no less detect, this situation.

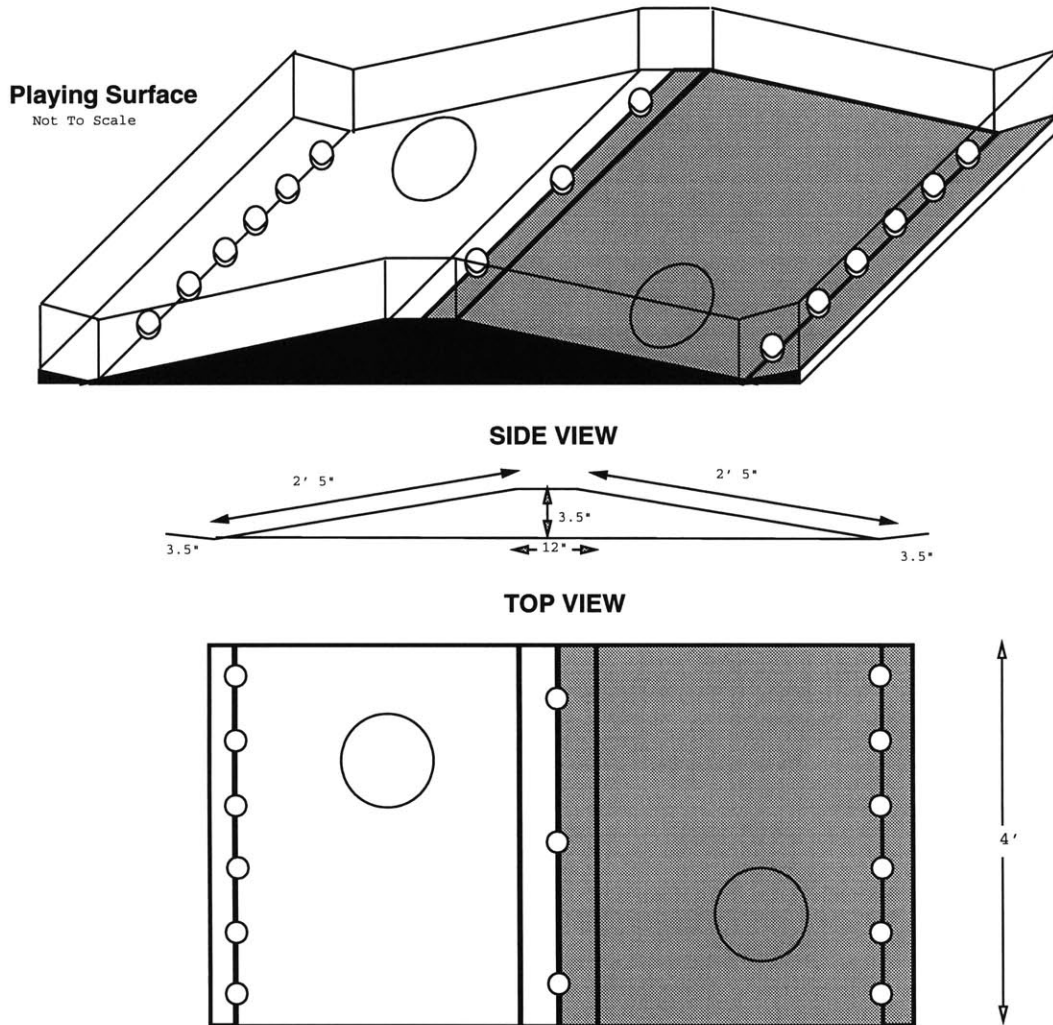
Two teams produced aggressive designs which hoped to overpower their opponents through force. One design used an extremely high gear ratio, so it moved very slowly but presumably with a great deal of power. It failed in that while it could push the other robots a bit, it couldn't force them to give up the puck if they had gotten to it first. Another aggressor design flipped down a wedge-like protrusion that it planned to drive underneath opponents, toppling them. But it was unable to reliably locate the opponents, and the robot ended up fruitlessly spinning in circles; it seemed there were problems with its puck-locating capability as well.

**The Robo-Pong Contest** The *Robo-Pong* contest was designed with specific attempts to encourage a diversity of solutions. While *Robo-Puck* was based on a single game object (the puck), *Robo-Pong* included multiple game objects (a total of fifteen ping-pong balls).

Figure 3-8 shows the final game design. The game was played on a doubly-inclined surface, a kind of hill, with a level center plateau. The goal of the game was to transport, hurl, or otherwise move ping-pong balls onto the other robot's side of the table. At the start of the round, each robot had six balls located at the base of its side of the slope (its *ball trough*); three balls were located on the center plateau.

# 6.270: The LEGO Robot Design Competition

## “ROBOPONG”



**OBJECT:** At the end of a 60 second round have fewer balls on your side than your opponent has on his side.

- Contestants begin in diagonally opposite circles marked on the table.
- Two Contestants and 15 balls
- 6 balls on each side of the table and 3 balls in the middle plateau
- 4.5 inch high clear plexiglass rim surrounding the playing surface
- The playing surface will be divided into a dark region and light region
- The playing surface may not be permanently altered or destroyed
- All evidence of an entry must be removed within 30 seconds after the end of the round
- Robots may not exceed a 1'x1'x1' Max at the beginning of the round

*(drawn by Pankaj Oberoi)*

Figure 3-8: Robo-Pong game design

By placing the balls on the center plateau, we hoped to encourage a greater diversity of strategies. We predicted that some robots would opt to play toward the center balls first, while others would initially attempt to remove the balls at the bottom of their side. The center balls also made it possible for a simple robot—one that just climbed uphill—to potentially win a contest round. In this way, the contest could be satisfied without great difficulty. By encouraging diversity and allowing simplicity, we believed *Robo-Pong* would be an ideal contest challenge.

The results of *Robo-Pong* were in line with our expectations: a greater diversity of robots than we had ever seen before. Successful *Robo-Pong* robots needed to be able to climb uphill and downhill, maneuver in the trough area, and coordinate activities of collecting and delivering balls. Overall, ball-collecting robots were the most popular design choice. This was not surprising in that they were also the design that required the least mechanical complexity.

There were two basic types of ball-collectors: *harvester robots* and *eater robots*. In the harvester approach, chosen by eighteen teams, robots scooped the balls into some sort of open arms and then pushed them onto the opponent's side of the playing field. Eater robots, constructed by eleven teams, were similar in principle to the harvester robots with the exception that the eaters collected in balls inside their body before driving over to the opponent's side.<sup>2</sup> Students believed the eater robots to be a safer design than the harvesters, since balls couldn't be returned to the robot's own side by the opponent. The eaters, however, ran the risk of failing to get onto the opponent's side before the end of the round.

Four teams successfully deployed *shooter robots* which catapulted balls onto the opponent's side of the table. We had placed a rule clause in the contest design which we hoped would encourage the development of shooters: if a ball were to leave the playing field over the opponent's territory, it would count as permanently scored against the opponent. Hence, shooting balls over the opponent's "head" was a sure-fire way to score; the opponent couldn't bring the balls back to your side.

However, the shooter concept required a fair bit of mechanical ingenuity to both con-

---

<sup>2</sup>David Chen, a Visiting Professor at the MIT Media Lab during 1991, helped categorize and tabulate the robot strategies in the 1991 class.

struct a shooting mechanism and a device to load balls into the shooter. Many more teams attempted the shooter design than eventually deployed one. The shooters that were finally fielded were sophisticated and pleasing to watch, but ultimately lost against aggressor designs, which could trap them easily, and effective collector designs, which delivered balls back after they had been shot across the table.

Two teams constructed dual robot designs. In one of these projects, one half of the robot acted as a ball shooter while the other half drove up to the center plateau and held up a shield to block the opponent. This was a very sophisticated design from both a mechanical and programming point of view; when it worked it was quite impressive. Unfortunately, recurring mechanical problems kept it from winning a single round during the actual contest.

The other dual robot design hoped to bring all of the balls to the opponent's side by having a small extension robot drive to the opposite end of the rear area. A chain built out of LEGO parts that linked the main robot and the extension robot would drag the balls out of the rear basin area and over the top, along with the center plateau's balls. This design did not compete as its creators couldn't get it to work in the final few days before the contest.

**The Robo-Cup Contest** There were several considerations behind the design of the *Robo-Cup* contest, largely in reaction to perceived flaws in *Robo-Pong*. One of the problems, discussed immediately following in Section 3.1.2, was that *Robo-Pong* was not sufficiently structured to bring out the best in the students. The other problem, discussed subsequently in Section 3.1.3, was that *Robo-Pong* was vulnerable to a simple aggressive strategy which could have overrun more sophisticated robots.

In order to strike a balance between these factors, the *Robo-Cup* contest made it significantly more difficult for a robot score points and hence be able to win. Figure 3-9 shows the *Robo-Cup* playing field, from a bird's-eye view. The task was to collect ping-pong balls from the ball dispensers and transport them to the appropriate goal. (The shaded areas represent regions that were painted with dark paint; robots could readily determine the difference in reflectivity between the dark paint and light paint surfaces. The edge of the shaded areas trace a path between the goals and ball dispensing devices.)

A close-up view of the goal is shown in Figure 3-10. The goal was divided into an

# The "Robo-Cup Soccer" Playing Field

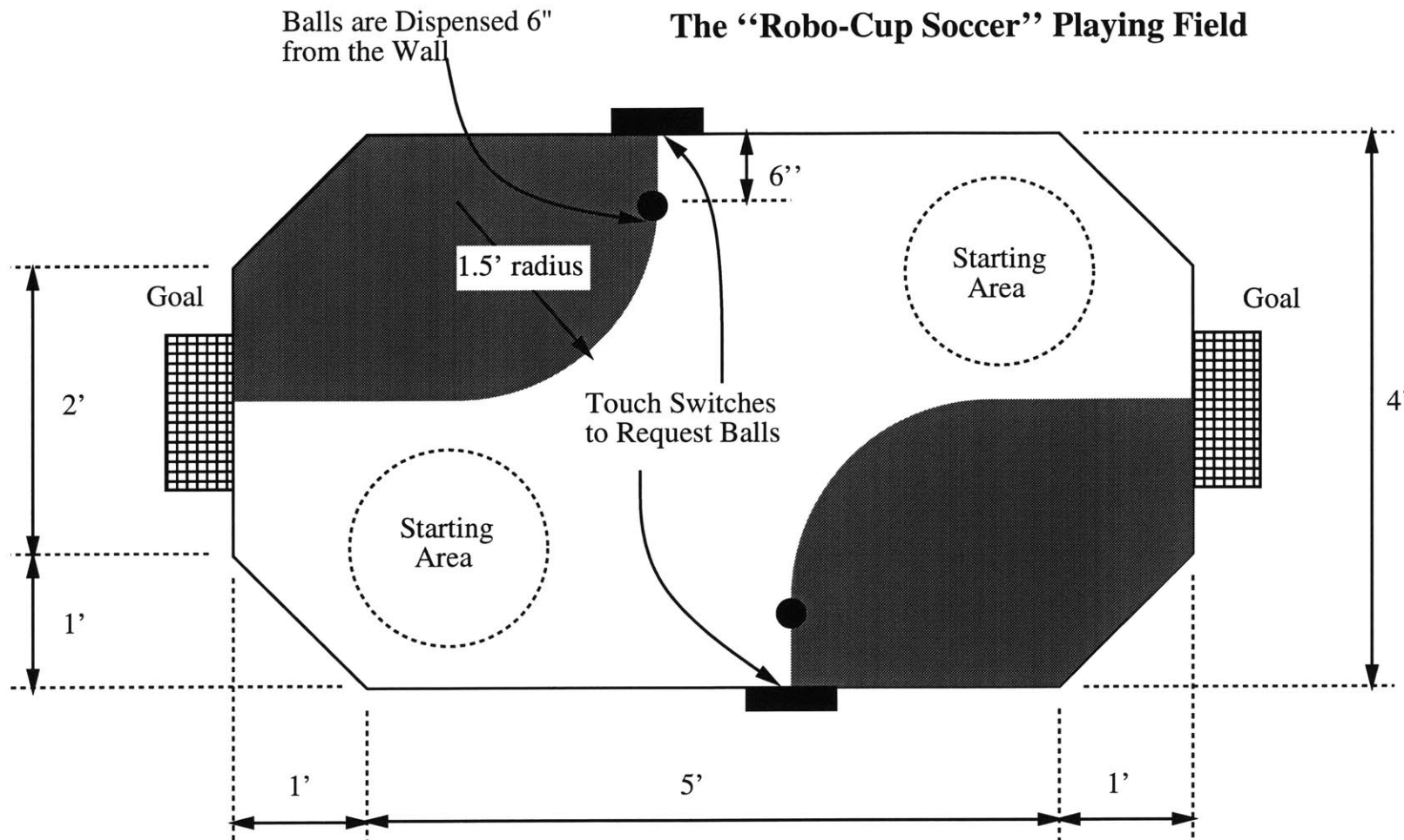


Figure 3-9: Robo-Cup contest playing table

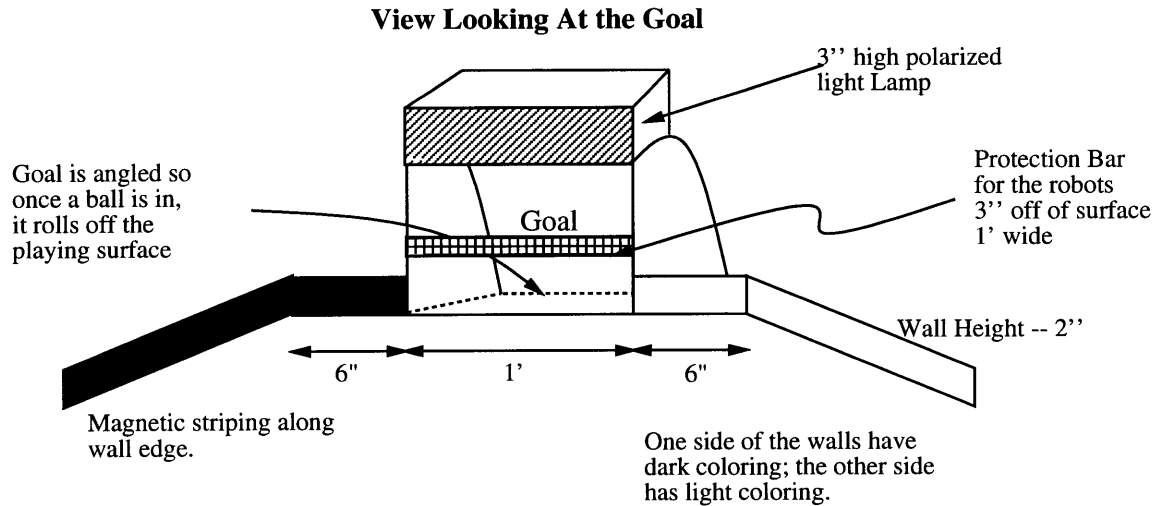


Figure 3-10: Close-up of goal in *Robo-Cup* game

upper and lower portion with a “protection bar” that served to prevent robots from falling into the goal. In dividing the goal into two regions, the bar served another purpose: a ball delivered into the upper portion of the goal counted as scoring three points, while a ball scored into the lower portion of the goal was worth two points. This point differential was made to encourage robots to shoot balls into the goal—if there were no differential, a designer contemplating a shooting robot would have no incentive to shoot the balls rather than just roll them in.

In order to score a point in the *Robo-Cup* contest, a robot had to perform a number of specific competences, including moving from its initial position to the location of the ball dispenser, depressing the ball request button, and transporting or shooting the ball into the goal. The result of our having imposed this series of behaviors was that students did not develop the diversity of strategies as in the past two years of contests. In retrospect, this was not surprising as *Robo-Cup* had a relatively high degree of structure when compared to the prior contests. Thus, in this regard, the *Robo-Cup* contest was lacking.

*Robo-Cup* robots could be categorized into two types, the first one being by far the dominant:

**Ball Carriers.** These robots caught the dispensed balls into some sort of holding chamber, shuttled over to the goal, and released the balls into the goal.



One important parameter that varied considerably among members of this class of design was the number of balls that a robot would collect before carrying them over to the goal. Some robots tended to the “all the eggs in one basket” approach: they collected balls from the dispenser until the round was just about to end before bringing them over to the goal. Others shuttled just two balls at a time. Most fell somewhere in between, making two or three trips in total.

**Ball Shooters.** Several teams fielded ball shooting robots. These were a more adventuresome design than the ball carriers in that the aiming problem was difficult—there was no obvious way of ensuring proper aim and consistent ball velocity.

One team came up with a quite innovative solution to the ball aiming problem. Their robot, named *Juicy Chicken*, had two components: a ball shooter and a separate mobile baffle unit that deployed itself at the goal. The ball shooter fixed itself at the ball dispenser while the baffle positioned itself at the goal. The shooter aimed the balls into the baffle, which was designed to guide balls into the goal, rather than shooting balls into the goal unassisted. Additionally, the baffle carried an incandescent lamp that the shooter unit would locate and use to aim. Ideally, this would mean that the shooter could aim quite effectively, as the baffle would position itself at the goal based on the surface markings of the table. Unfortunately, *Juicy Chicken* required too much mechanical sophistication for its designers to successfully implement within the time available. Also, the design had the unpleasant effect of often unintentionally entangling its opponent with the wire harness that joined the two halves of the robot; this typically resulted in both robots becoming disabled.

We concluded that *Robo-Cup* was successful in focusing students on the contest task, but this accomplishment was made at the expense of a diversity of solution strategies.

### 3.1.2 Challenge Level

Closely related to the issue of encouraging strategic diversity is the issue of targeting a contest’s complexity to provide an optimal challenge to the students. This issue came up after an analysis of the *Robo-Pong* contest.

We considered the *Robo-Pong* contest to have been quite successful; an interesting

variety of robots were built and the robot designers seemed to be well-challenged by the contest specification. There seemed to be a flaw, however, related to the fact that the *Robo-Pong* contest could be solved by a relatively simple robotic mechanism. That is to say, building a simple robot that was capable of knocking at least one ball onto the opponent's side was all but trivial.

As it happened, a number of robots with such a minimal level of functionality were able to qualify for competition. Quite a few of the final robot designs did not ever work dependably or reliably; they simply moved around enough to knock a ball or two over the center plateau and hence win a round. It seemed that because an erratic performance would win a round here and there, some students were not sufficiently challenged in the design task and ended up fielding robots that were not particularly competent. Thus while *Robo-Pong* had the important characteristic of being solvable, it perhaps erred too far in that direction.

There is an amusing robot development story to help make this point. One team of students planned to build a robot that would drive back into its trough, drop a pair of arms to either side, and sweep all of its balls plus the center balls over to the other robot's side of the table in one fell swoop. Unfortunately, they had difficulties both in implementing their idea and in working together as a team. Finally there were just a couple of days before the contest and their robot was far from working as they had hoped. In a last-ditch, "who cares" sort of effort, they programmed the robot to simply drive forward until getting stuck, and then back up and go in the opposite direction. So the robot would drive forward and backward, crashing into walls or other objects as it did so.

The students had built the arms onto the robot, but the arms were quite feeble: they were short, often got stuck when being deployed, and would invariably fall off of the robot as it drove backward and forward crashing into things. So this was the robot the team fielded the eve of the contest, naming it *Stupid Scorpion* in disgust.

To the genuine surprise of all concerned, the robot did extremely well in the contest performance, placing third best overall among all entrants. It achieved this result through its simplicity and dogged perseverance (the latter a quality that it also shared with its makers)—it just wouldn't give up. Many other robots would get stuck and fail for the rest

of the round, but *Stupid Scorpion* just kept driving back and forth. In theory, the robot was just as likely to hit a ball onto its own side of the table rather than onto the opponent's, but somehow it just kept winning.

In a sense, *Stupid Scorpion* was a deserving winner because it was persistent and reliable. But there were about ten to fifteen other robots that really didn't work but were able to qualify for the contest by winning a round "by accident"—in the manner that *Stupid Scorpion* did "by design." We felt it would be better for the students if they were challenged a little harder to build something that really worked. So this became an important constraint in designing the next year's contest: robots shouldn't be able to win by accident.

We generated the idea of using specific goal areas rather than goal troughs, as we had in the *Robo-Pong* contest, as a way to make it unlikely that robots could score by error. Additionally, we established a disqualification rule for non-performing robots. The changes can be summarized as follows:

- **Higher Minimum Performance Threshold.** In *Robo-Pong*, a robot could win a round (and thereby qualify for competition in the preliminary round) simply by driving uphill and knocking a ball off of the center plateau. In *Robo-Cup*, a robot had to successfully perform a number of competences, including finding the ball feeder, dispensing a ball, locating the goal, and delivering the ball to the goal. Each of these activities was individually as complex as the hill-climbing task that minimally satisfied the contest in *Robo-Pong*.
- **Disqualification for Non-Performing Robots.** Teams whose robots were unable to score a single ball in the preliminary round were given until midnight of the evening preceding the main contest to get their robots working well enough to score at least one ball in a round without interference from an opponent. Otherwise, they would not be allowed to compete in the main contest.

*Robo-Cup* succeeded in steering students toward more functional designs, but at the expense of a diversity of solutions, as noted earlier.

### 3.1.3 The Social Message

As a community event, the contest exhibition should encourage positive goals like fair play and respect for one another's work. As an educational event, the contest should encourage students to interact with and learn from one another during the robot development process. The contest game should be inviting to a wide range of students, including those who are not interested in aggression as a means of personal expression.

Using a competition as a pedagogical tool can present a conflict: if students are highly competitive, they may not wish to share their ideas with others, based on the perception that this would be revealing valuable information that would comprise their robot's chances in the contest. We realized this and made every effort to discourage this sort of behavior, encouraging students to share ideas with one another and consider the final contest a friendly affair rather than a dead-serious one. Many students made an effort to share ideas publicly, and reported that their learning experience benefitted as a result of it.

In addition, we made attempts to move away from the early destructive images of robot-play that we inherited from the contest's origins. The organizer of the *King of the Hill* contest promoted it with an image similar to those promoting the popular American monster truck rallies and demolition shows (see Figure 3-11). The later contests we developed were based on sporting events, like hockey, tennis, or soccer, and were promoted with images like the one shown in Figure 3-12, which advertised the *Robo-Pong* contest. We believed that it was important to de-emphasize the destructive potential of technology as both a positive example and as a way of encouraging participation from those with less hyper-competitive personalities.

This effort extended in detail into the design of the contests themselves. For example, midway through the progress of the *Robo-Pong* class, we realized that there was a flaw in the contest design in which just about any strategy would be vulnerable to a dedicated aggressor robot. The strategy of such a robot would be to head straight for its opponent at the start of the round; it would win by (1) bringing one or perhaps two of the balls from the center plateau over to the opponent's side, and (2) trapping the opponent before it could do anything. We considered this a problem because we didn't want create a contest that rewarded this kind of behavior, both because it would be a bad lesson and because it would

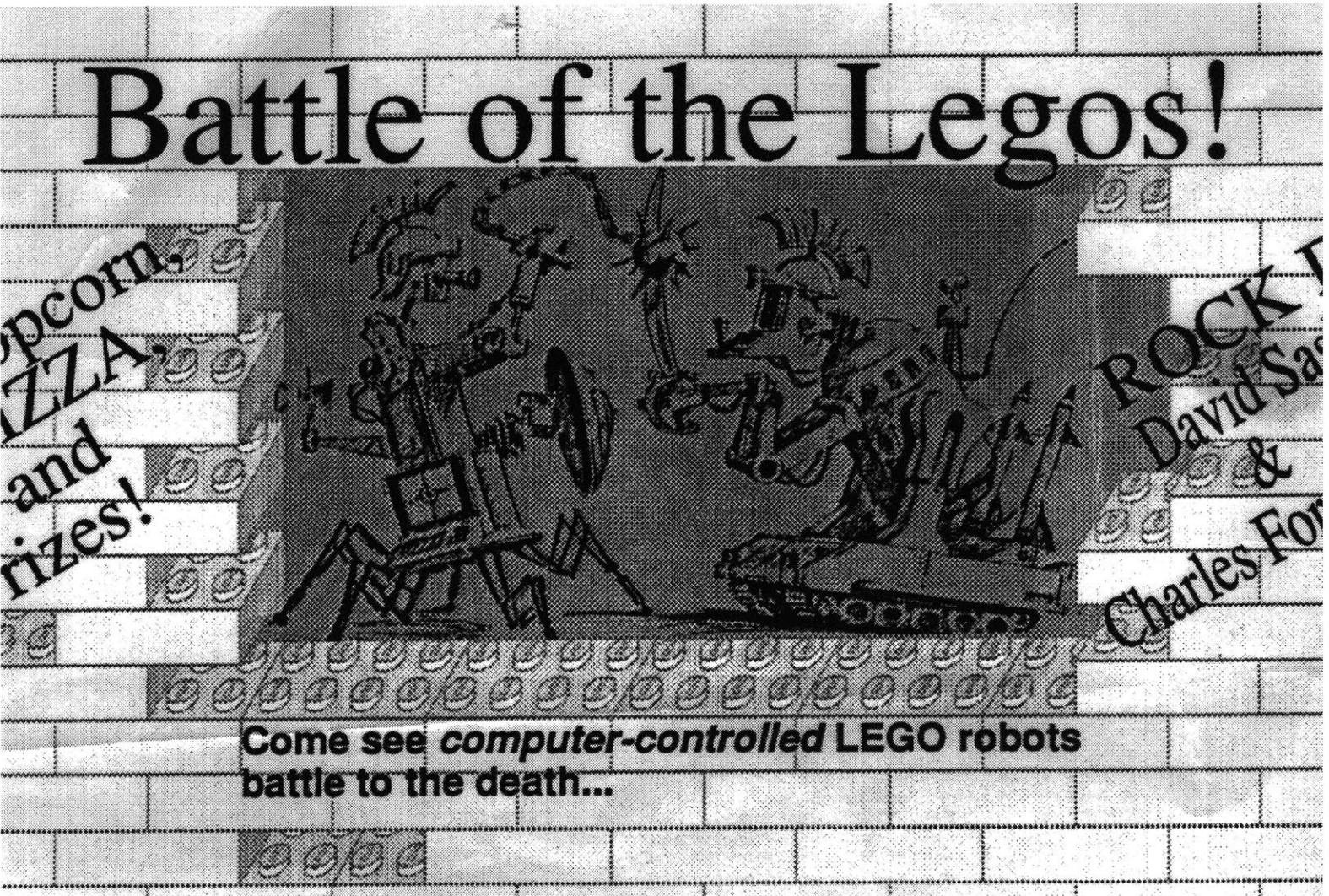
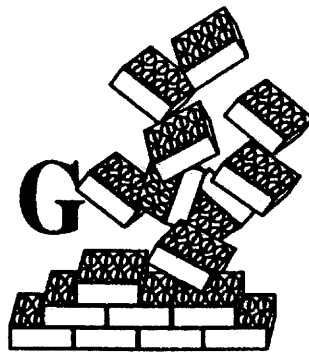


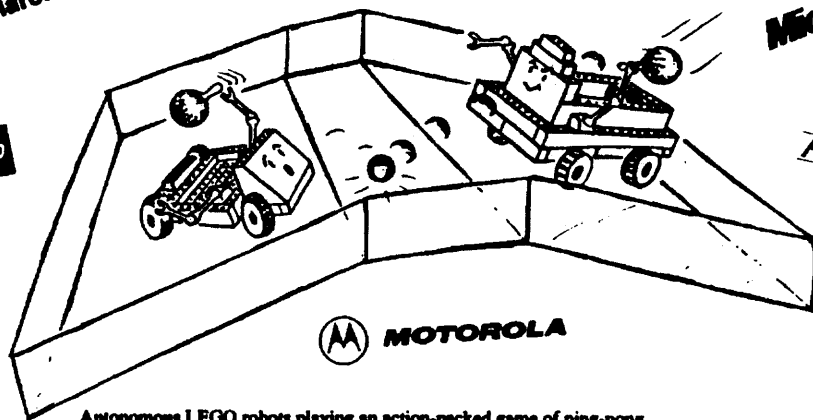
Figure 3-11: Graphic used to promote the first LEGO Robot Design Contest

The Fifth Annual  
"6.270 Contest"

R O b O n G  
O P O n G



Polaroid Corporation



Microsoft

AGE



Autonomous LEGO robots playing an action-packed game of ping-pong

Figure 3-12: Graphic used to promote 1991 *Robo-Pong* contest

discourage the creation of more interesting, complex strategies.

Surprisingly, none of the teams built such a ruthless, “assassin” robot. Three teams did field designs based on aggression, but both were more sophisticated than the minimalist aggressor just described. The extra baggage on the two aggressors ultimately caused their failure, as any excess mechanism is wont to do.

In discussions with participants during the month, several mentioned the viability of the assassin strategy, but decided that they were far enough along in their collector designs that they chose not to implement the assassin. One student, whose team dropped out of the project, claimed that they had lost interest in the project once they discovered that a ruthless aggressor would win the contest. I tried to argue that building such a robot that performed reliably would still be a challenge, but the students were not interested.

These observations fed into the design of *Robo-Cup*, the subsequent contest. In all previous contests, robots were more or less encouraged to collide with one another. In *King of the Mountain*, robots congregated at the top of the mountain. In *Robo-Puck*, robots fought for possession of the puck. In *Robo-Pong*, robots were likely to collide as they drove over the top of the playing field.

To reduce the likelihood of unintentional collisions, we designed a default path pattern for *Robo-Cup* robots that would keep them away from each other. Robots were allowed to draw balls from either ball feeder, but the surface pattern on the table connected each goal to one feeder—it was far simpler to build a robot that shuttled balls back and forth from the goal to the favored feeder than the other feeder. Two robots each following the path to the easier feeder would not collide in normal game action. This implementation, combined with the fact that it was difficult to score a goal in *Robo-Cup*, effectively discouraged the creation of simple brute force attack robots.

### **3.1.4 Summary**

Contest design is challenging because there are many constraints which must be satisfied to create a successful contest. The contest frames the overall problem architecture for the students’ work, and should be open-ended enough to allow for creativity and individual expression, but must be specific enough to focus students on difficult problems. Contests

need to create an atmosphere of friendly sport, and must be carefully screened for quick or obvious solutions that would discourage full participation by the students.

## 3.2 Hardware and Software Design

We created specific hardware and software technology for the students' use in designing their robots. These materials had a tremendous impact on the nature and style of ideas explored by the students and embodied in their robots.

As discussed in the introductory chapter, the early work on the Logo programming language and later work on the LEGO/Logo materials and the Programmable Brick established the intellectual heritage of the work done for the Robot Design project. Specifically, the concerns that guided our designs were:

- **Level of Abstraction.** Any educational technology hides or isolates the user from certain phenomena while revealing or highlighting others. In developing tools to facilitate the design of robots, we paid special attention to the sort of technological ideas we were exposing. Since a robot is a system comprised of a variety of media—electronics, programming, and mechanics—it was necessary to be clear on which concepts we expected students to master and which others they could simply use.
- **Transparency.** Even if a certain idea is encapsulated by the layer of abstraction, it should be easily accessible to students who are interested. For example, we determined that students should *not* need to have a deep understanding of digital electronics in order to build their robots. But we did not want to prevent or discourage students from exploring this topic as part of their robot-building. Quite the contrary, we hoped to invite them to do so through the design of our materials, while simultaneously taking pains not to intimidate students who might not be interested in this topic.
- **Interactivity.** Central to our project pedagogy was the belief that people learn best by *exploring ideas in a playful manner*. This was the *modus operandi* of our technology development, and a key concern was creating materials that would encourage this behavior in our students.



The technology developed for the Robot Design work consisted of three stages of increasing sophistication and educational value. These stages reflect both our technical learning process—our increasing ability to fluently express our model of an effective educational technology for robotic design—and our understanding of what this technology should be.

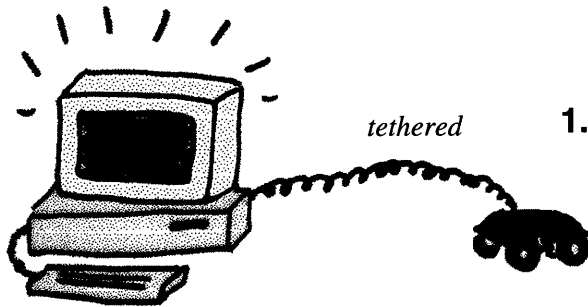
To facilitate the discussion of the core ideas of this section, a brief introduction to the technology is necessary. I attempt to make this discussion as free of technical buzzwords as possible that it may be of value to the readers who are not roboticists by background. I hope that portions of this section will then be a pleasant learning experience rather than an intimidating one for such readers. On the other hand, the reader who is specifically interested in the technical details of our implementations is referred to Appendix D.

The first stage of technology was a hand-wired *Remote Controller*. Students built a controller board into their robots which was tethered to a desktop computer. The desktop computer acted as the “brain” of the robot: the remote board was simply used to interface the motors and sensors to the desktop machine. This was the technology used by students of the *King of the Mountain* contest.

The second stage enabled students to create truly autonomous robots that did not require a clumsy tether. Used in the *Robo-Puck* contest, the *Assembly Language Controller* allowed students to download a program from a host computer, at which point the host could be disconnected and the robot would run on its own. The Assembly Language Controller so named because it was programmed in assembly language, a primitive and low-level computer language.

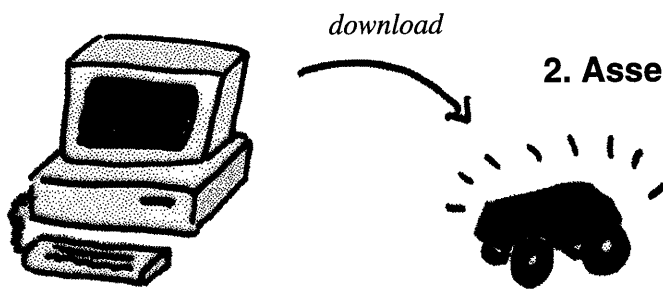
The third stage also allowed robots to roam free from the host computer, but allowed students to develop their programs interactively and with the use of a high-level programming language. The *C Language Controller* board had a number of other features that made it a much more versatile platform for the students’ work than the two previous stages, as will be discussed.

Figure 3-13 illustrates these three stages of technology design.



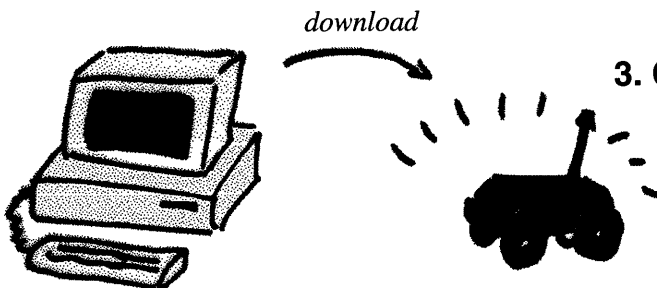
### 1. Remote Controller

- \* tethered to desktop computer
- \* hand-wired
- \* basic functionality



### 2. Assembly Language Controller

- \* program downloaded to robot
- \* machine programming
- \* batch-mode interface
- \* difficult to debug



### 3. C Language Controller

- \* program downloaded to robot
- \* procedural programming
- \* interactive interface
- \* helpful status information

(drawn by Wanda M. Gleason)

Figure 3-13: Three stages of robot-building technologies

### 3.2.1 Levels of Abstraction

The short course description advertising the Robot Design project to MIT students read like this:

*You are given a kit containing a microprocessor, LEGO blocks, batteries, motors, sensors, and wire. Your task: design and build a robot to play in a robot sporting event (details to be provided). Lectures, recitations, and lots of laboratory hours will help you in your task. You have one month.*

Many students took this description literally and expected us to hand them an unsorted potluck of electronic components, with the implicit message “here’s a bunch of junk; figure out how to do something interesting with this stuff.” But we never would have attempted to run a course like that, because students would spend all of their time reinventing the basics of robotics. Instead, we provided materials that gave them a basis of capability upon which they could build.

In fact, an important design criterion of our materials and the course as a whole was that no specific technological knowledge should be a prerequisite (with the exception of some programming background). We did not require students to know soldering skills, circuit design, mechanical design, or a particular programming language in order to be participants. These skills would be learned as needed during the progress of their robotic design work.

The structural and mechanical aspect of robot design was done using the *Technic* system of beams, gears, axles, and connectors developed by the LEGO Group. LEGO *Technic* parts are compatible with the familiar system of LEGO building blocks—the commonly available children’s toy—but the *Technic* series adds an extremely flexible and versatile kit of parts for designing moving mechanical structures.

With raw materials like wood, plastic, and metal and the appropriate machine tools, anything can be built. The process of creating an artifact from these raw materials, however, can be slow and painstaking, since one must hand-craft each and every component of the finished product. With the LEGO *Technic* system, certain types of objects, like frames, linkages, and geartrains, can be constructed quite rapidly. Thus the *Technic* parts become the modules that the builder uses to create designs; the builder is not concerned with how

the *Technic* parts were originally created, but rather, how to use them at face value.

Borrowing the terminology of Abelson and Sussman (Abelson & Sussman, 1985), I call the new set of tools provided by the *Technic* system, for example, a *layer of abstraction*. By this I mean that the *Technic* parts insulate the user from lower-level issues of implementation—how the parts are fabricated or how to use machine tools to create structural members, for example—rather than suggesting that there is something “abstract” about LEGO parts. Quite the contrary; nothing could be more tangible or immediate than the act of holding a LEGO brick and using it to build something. The notion of abstract in this sense is to suggest the underlying technologies that have been insulated from the user’s attention.

In a similar way, the robotic technology we created for our robot design course provided a layer of abstraction for the building of robotic systems. Beginning with our earlier systems, students did not have to concern themselves with issues of microprocessor circuit design in order to use these tools. In our later systems, students were “abstracted away” from problems of machine language programming (which they were exposed to in earlier systems). This section explores the abstraction issue from a pedagogical point of view.

## **Bits and Bytes**

In the first year of hardware robotics, cost, time, and our own experience were limiting constraints on what we could provide. We first agreed on what was a necessary core of a hands-on robotics experience. We determined that it would have to include sensors, motors, and programming. So our minimal system would have to allow project participants to build a machine that incorporated motor control, sensor input, and programming to tie it together.

The students who used the Remote Controller didn’t have to learn the details of the electronic circuit on which the controller was based, but they did have to work at a fairly low level in order to control motors and receive data from sensors. In order to power a motor, a byte with the correct combination of bits set to zero and one would be sent from the host computer to the controller. In order to interpret sensor data, a byte received by the host computer would be checked for a one or a zero in the proper bit position. Thus the remote controller allowed students with little circuit design experience to build robots,

but forced them to deal with bit-level manipulations in order to interface with their robot's motors and sensors. We saw this as beneficial; the concepts were well within the grasp of the students, and provided a valuable lesson in interfacing hardware and software.

## **Machine Registers**

For the next year's class, the *Robo-Puck* contest, we created the Assembly Language Controller. This allowed students to build robots that "carried their own brain" and operated autonomously from the desktop computer. The desktop computer was still required to compose and download programs to the robot, but it was no longer used when the robot was running.

The use of the Assembly Language Controller was similar to that of the Remote Controller: students worked at the level of bits and bytes to control motors and process sensor data. Additionally, however, there was the complexity of the microprocessor upon which the Assembly Language Controller was based. Students were required to program directly in the machine language (or assembly language) of the microprocessor, and forced to understand issues like which machine register to use to control the motors and what sequence of operations was necessary to retrieve information from the sensors.

Though it was valuable for some, the use of assembly language was problematic for the majority of the students. Most students had never programmed in assembly language before, and were left with little time to learn to do so in the tight schedule of the class. The conceptual overhead involved in doing the most minimal programming task (e.g., proof-of-concept code that turned on a motor based on a sensor reading) was significant: the creation of even a trivial robotic task required the understanding of a variety of mundane programming details.

Many of students required substantial help in order to achieve working programs, to the point that they themselves no longer understood the program running on their robot. There was one team that worked all night on their program but failed to get anything running in time for the competition, despite having completed and tested the mechanics and sensors on their robot, due to difficulties in debugging their program.

Other students who had experience in assembly language were successful in imple-

menting their ideas, but still felt hampered and frustrated by the assembly language, mostly for the difficulty in implementing their ideas and the painstaking debugging techniques required.

There was a small group of students who found the assembly language experience to be rewarding. One of these students already had a conceptual grasp of assembly language programming, but lacked practical experience in doing it. He developed a robot design that required a highly specific task to be performed by the software. The job of writing the well-defined program for his robot was valuable in that it served to make concrete to him the ideas of assembly language, and his program was simple enough that he did not get bogged down in obscure programming detail.

Another student also had a simple design that did not require a complex program, though it still required more “tweaking” and incremental design than the aforementioned student. This student had taken a course in assembly language programming at the university level, so she was familiar with the core concepts involved. The task of developing a program for her robot, again, served to make concrete the fashion in which data is moved around within the microprocessor model. The fact that data came from physical sensors and was used to control physical motors made the programming experience more concrete and visceral for her.

Our conclusions from the experience of using the Assembly Language controller were mixed. While the majority found it difficult, it was clear that some students were empowered by the low-level, “nuts and bolts” nature of the microprocessor-level programming task. Some of the problems we experienced could have been mitigated by better presentation of the conceptual material and the provision of better software tools. On the other hand, it was apparent that the degree of complexity of the robots that students would build was fundamentally limited by the low-level programming environment.

The question we faced was whether to upgrade the technology to support high level programming and control, allowing students to build more complicated systems, or to retain the more primitive tools, allowing students to experience the satisfaction of moving bits and bytes around to get work done. With some reservations, we chose to move forward with a higher level environment.

## Procedural Programming

We decided that it would be advantageous to give students a higher level interface to their robot design work, shielding them from details of microprocessor programming but offering them the possibility to express more complex ideas. This was a difficult decision, as we felt that the assembly language programming experience could be quite valuable for students, but it was in keeping with the overall philosophy of the project. In giving students a predesigned controller board and predesigned sensors, we were abstracting them away from low-level details of digital and analog electronics; if we provided them with a higher level software interface, it would be continuing an established trend. The value of the providing students with a higher level software environment would be evaluated by seeing the sorts of problems in which they became engaged, in comparison to the sorts of problems they encountered in the earlier assembly language environment.

We created a hardware and software system to allow students to program their robots using the C programming language. We called the language *Interactive C* to highlight its interactivity, which was an important aspect of its usability; this feature will be discussed later in this section.

With Interactive C, students used procedure calls to interface with the motors and sensors of their robots. For example, the statement “`motor(0, 100)`” would be used to turn Motor 0 on at speed 100 (full speed). The statement “`if (analog(0) > 100) { ... }`” would cause the expression in braces to be executed if sensor 0 was greater than 100. Thus students had a high-level interface to the hardware of their robots and for expressing their control ideas.

The results of evaluating students’ use of the Interactive C system convinced us of the value of this approach. Students created significantly more sophisticated robotic systems than they had in either of the previous two years; this work is analyzed in the following two chapters of this dissertation. A few students did express disappointment that they were shielded from the lower level of hardware and software operation, but by the end of the course, they agreed that it was better to have the expressive power that the higher level system offered.

By providing the higher level tools, students were able to work with a different category

of conceptual material. Rather than being concerned with what combination of bits was required to enable a motor, they could focus issues like algorithms for processing sensor data and strategic methods for organizing their robots' behavior.

### 3.2.2 Observability

Another issue affecting the pedagogical value of educational technology is related to the *observability* of systems that are constructed with it.

The LEGO *Technic* system will again serve as an example. When a student constructs a machine using LEGO *Technic* parts, the functioning (or lack of) of the object is, generally speaking, in plain view. This is to say that with straightforward observation and interaction with the artifact, the builder can readily determine what its modes of operation and modes of failure are. For example, if a LEGO structure often breaks apart at a particular joint, is it more or less apparent which joint is faulty. Solutions to repairing such problems may or may not be immediately evident, but problems are easily diagnosed.

Thus we can say that the LEGO *Technic* system provides a high degree of observability: through natural interactions with the material, the user can readily determine the properties of artifacts that he or she has constructed with it.<sup>3</sup>

The issue of observability become an important concern after evaluating students' work with the Assembly Language Controller and its associated development software. The system suffered from poor observability, which affected students' ability to learn with it in negative ways.

The observability problem was exacerbated by the technical nature of programming in assembly language. Programming errors tended to be of the "crash-and-burn" variety: programs fail without warning, without providing notice as to where they crashed, how or why. In the case of the robotic hardware being programmed, this problem was made even worse because of a variety and intermingling of possible failure modes. Not only could different types of programming errors cause a robot to fail, but so could various other sorts

---

<sup>3</sup>There are some aspects of the LEGO system which I would not consider highly observable. For example, if a rectangular frame is not rigidly braced with square corner joints, then axles supported by it will lose large amounts of energy in their bearing supports. This problem is by no means obvious to the builder. Still, this example and others like it are minor criticisms of a wonderfully designed mechanical building kit.



of hardware errors:

**Hardware Failure.** An electrical problem with the hardware of the computer could cause a crash. In this case, there may still be a software error that lurks behind the hardware problems.

It is important to note that this failure mode is typically not a part of computer science curricula. In most academic courses, students do not worry about the reliability of the *computer hardware itself*. In the robotic projects, however, the computer boards as well as other robotic hardware (sensors, motors, and mechanics) were susceptible to failures. Often, these problems were intermittent—the worst kind because the problems became so difficult to trace.

**Software Coding Error.** A software “typo” (error caused by mis-typing) or “thinko” (error caused by sloppy thinking, like substituting a less-than sign when a greater-than sign is intended) can cause a crash.

In most programming environments, these errors are hard to track down because the computer won’t find the error for you (i.e., the programming mistake does not generate an error message). When programming in assembly language, the matter is made worse because there is also a greater likelihood that the program will crash, rather than just plugging along generating improper results.

**Algorithmic Error.** The algorithmic error is an incorrect design of an algorithm to accomplish a certain task given certain inputs. This mistake does not necessarily cause a complete failure, but more an unexpected performance. The inputs may be correct, but the intended output does not occur because the algorithm for obtaining it is wrong.

**Sensor-related Error.** Often a sensor does not perform as a student expects. The student might create an algorithm that would perform properly if given the sensor inputs the designer anticipated; the problem arises when the sensor does not perform as expected. This failure mode is particularly tricky for a variety of reasons: (1) students don’t like to think that their algorithm is inadequate because the sensor produces anomalous

values; (2) sensor data tends to be noisy in a fashion that is difficult to model and for which to compensate; (3) students don't realize that they don't understand the sensor. To further complicate matters, it is often the case that a given sensor-algorithm *mostly* works—that is, most of the time it gives acceptable performance. This error was commonplace when students used the Assembly Language controller, because it was difficult to get the system to display sensor results in a fashion that would be meaningful to the student.

Because of this variety and intermingling of failure modes, the experience of working in assembly language was frustrating and unrewarding for most students. The most difficult part of the debugging scenario was that students often did not know which *type* of bug they were dealing with, no less how to fix it. We realized that we would have to provide a system that not only better supported students' debugging efforts, but actively encouraged their curiosity to understand the robotic phenomena they were exploring. For example, rather than just hoping that a sensor would work the way they expected, students should easily be able to construct a simple experiment to ascertain the behavior of the sensor.

In response to this set of problems, we built the following features into the hardware and software of the C Language Controller system:

**LCD Character Display Panel.** With the assembly language board, there was no manner for the hardware to provide feedback to the students about its internal state. That is to say, the students could program the board to control their robot's motors, but there was no easy way to provide status information about the state of program execution (other than by observing the state of the motors, which was typically the thing being debugged).

The new board supported a 15-character LCD display panel. The software we developed included a programming statement to write data to the display (both text strings and numeric output). This vastly changed the “observability” of program execution: it became easy to add a print statement to a program and thereby monitor its internal status.

Also, it became much easier to experiment with the operation of sensors. Students could write a one-line program to repeatedly print the value of a given sensor to the display. The robot no longer needed to be connected to the desktop development computer in order to display results—students could disconnect the robot from the computer, bring it to the contest playing table or other location, and manipulate the robot or conditions in its environment and directly observe changes to sensors' values.

**“Heartbeat” System Activity Monitor.** On the LCD screen, a small icon continuously flashed to indicate that the computer board was operating properly. If there was fatal hardware or software failure, this “heartbeat” would stop, and the user could tell at a glance that such a crash had occurred.

**Piezo Beeper.** An electronic beeper was provided with simple routines for making beeps of varying pitch and duration. The assembly language board had included a beeper, but it was difficult to operate from software and was not used by students generally. With the new system, we saw an explosion of “musical robots” that used sound output for both entertainment and informational purposes.

**Command Line Interface.** The assembly language system used a batch mode metaphor of programming: first the program was written, then it was downloaded and run on the robot. There was no opportunity for the user to interact with the program when it was running on the robot. With the C Language Controller, students could interactively control program execution or display the value of program values while their program was running.

While they may seem small, these technology changes drastically improved the observability of the students' robotic systems. Students were able to write a little snippet of code to display the value of a sensor, beep when a particular area of program code was executed, and tell at a glance that their microprocessor hardware was operating properly. While their robots still became complicated, difficult-to-debug systems, with the technology improvements introduced with the C Language controller, students had the tools at their disposal to debug their robots.

### 3.2.3 Interactivity

Another important criterion of an educational technology is the degree to which it encourages *interaction* between the learner and the ideas embodied by the technology. The LEGO building system represents perhaps the pinnacle of interactivity—a pile of LEGO bricks practically begs to be played with and put together in various ways.

It is through intelligent interaction with a material that learning occurs. The LEGO system is successful as a pedagogical tool because novice builders can express their ideas directly with the material, evaluating their design concepts without the need for intermediary representations.

For example, contrast a structural idea expressed in a LEGO model versus one expressed in a traditional pencil and paper drawing. The novice student can more readily evaluate the effectiveness of the idea in the form of a LEGO model: it can be viewed from any angle, prodded, twisted, and dropped to test its ruggedness. The drawing, however, must be carefully analyzed in a cerebral manner, inviting errors from the novice designer.

It may be argued that for an expert designer, simple drawings are as powerful or more powerful than physical models as a design tool, particularly in the more conceptual stages of a design. It is often simpler to sketch an idea than to give it physical form. Further, the expert designer often has the ability to hold in his or her own mind the myriad of implications that each component of a design has on the others. Therefore the expert does not always need the physicality of a model to explore options and alternatives. For the novice, however, the immediacy of a model, particularly one that is created as part of the ideation process itself, can serve to provide critical feedback about the effectiveness of the design ideas.

This criterion was applied to the design of the other components of the robot-building kit. The biggest problem with the software environment of the assembly language system was its batch-mode metaphor: first the user would write a program, then assemble it, then download it, and then see if it worked. In contrast, the Logo Programmable Brick robot-building system (discussed in the Background chapter) provided a Command Center, in which users could type commands which would be executed immediately after being typed.

We incorporated a feature like the Programmable Brick's Command Center into our C

1. User sits down at computer console, plugs his or her robot into the computer via a cable connected to the computer's serial port, and then invokes the programming environment by typing the command "ic" (for Interactive C) to the system prompt.

2. Computer displays:

```
Interactive C version 2.63  
Connecting to board... ready.
```

```
C>
```

The "C>" is Interactive C's prompt, indicating it is ready to accept a command from the user.

3. The user wishes to test the system by performing a simple addition, and types:

```
C> 3 + 2;
```

(The user's input, "3 + 2;", is underlined for clarity.)

The computer downloads this statement to the controller board, which performs the addition and returns the result. The computer displays:

```
Returns 5.
```

4. The user wants to turn on one of the robot's motors:

```
C> motor(0, 100);
```

As soon as the user hits return, the command is sent to the robot; the robot turns on its Motor 0 at speed 100 (full speed).

5. The robot is now spinning in circles. The user types:

```
C> off(0);
```

to turn off Motor 0.

Figure 3-14: Session with Interactive C (page one)

6. The user wishes to determine the value of an analog sensor, and types:

```
C> analog(3);
```

The computer displays:

```
Returns 140.
```

Sensor values range from 0 to 255.

7. The user wishes to download some procedures that he or she has written and stored in the file `myprocs.c`:

```
C> load myprocs.c
```

Computer displays:

```
Loading myprocs.c.  
Downloaded 1034 bytes.  
C>
```

8. The computer has compiled and downloaded the user's procedure file and is now awaiting further input. The user would like to run a procedure from that file, `testrobot()`.

```
C> testrobot();
```

Robot begins executing `testrobot()` procedure.

9. The user would like to repeatedly display the value of analog sensor 3 on the robot's LCD panel, and types in a one-line statement to accomplish this:

```
C> while (1) printf("Sensor 3 is %d", analog(3));
```

Robot begins displaying the value of sensor 3 on its LCD display.

Figure 3-15: Session with Interactive C (page two)

Language controller system—an interactive command line interface. By typing function calls and compound statements at the command line, users were in interactive control with their robot, and could directly command motors settings and examine sensor values, as well as being able to download and execute procedures.

To highlight this aspect of the system, we named the language *Interactive C*, or IC for short. Figures 3-14 and 3-15 show what a session with the new software was like. The sample session illustrates how easy it was for the user to interact with the hardware of the robot by showing how a user might turn on and off the motors, display a sensor value, and run a procedure.

Most recently, the command line interface has come to be perceived as anachronistic, with many operating systems and applications now providing iconic and menu-based objects for interaction with the computer. Yet the command line is far from having outlived its usefulness, as the sample session illustrates—particularly as a programming tool.

The Interactive C system was a huge improvement over the batch mode methods of the assembly language programming software. Not only did the Interactive C system make it easier for students to write programs and understand them, but it specifically encouraged a more playful, experimental way of working with the components of the robot-building kit. It became possible, for example, to write a one-line program to display the value of a sensor on the LCD screen. This encouraged students who were unfamiliar with the operation of a given sensor to write that one-line program, carry their robot to the contest playing tables, and experiment with the robot to see how the sensor responded.

Similarly, writing a short program to test a control idea (like following the edge of a line on the playing table) became a one-hour proposition rather than an all-day challenge. The result was that a number of students, particularly those who were organized enough to give themselves time to play, built sample robot behaviors (like a robot that would follow a flashlight) in a manner analogous to the majority of students would built sample LEGO models to explore structural and mechanical ideas. Because of the system's interactivity, students were implicitly encouraged to play with sensing, control, and programming ideas.

### 3.2.4 Transparency

Related to the issue of levels of abstraction is matter of transparency of those levels. Given that a system insulates the user from lower levels of detail, the question remains of how easily users may explore those other levels if they desire.

This issue became important to us when we made the shift from the more primitive, assembly language system, which insulated the user from little, to the C-language system, which abstracted various hardware and software details from the students. We did not want to discourage interested students from learning about the lower levels. Quite to the contrary, we felt it was important that at least their curiosity would be piqued and that they would feel an implicit invitation to “peel away” our layers of abstraction.

There were two reasons for making our materials open in this manner. The first was to stimulate and support students with different backgrounds, styles of inquiry, and interests. If our system was “closed,” then we would shut out students not interested in the particular abstractions we had selected. The other reason that we did not want to build a system that seemed complicated, magical, or otherwise intimidating; we wanted to encourage learning, not stifle it.

These intuitions were borne out by experience. As mentioned earlier, a few students were initially dismayed when they realized that the Interactive C system shielded them from the lower hardware and software operation. With our consent, one student rejected Interactive C outright, and attempted to program his robot in assembly language. (This example is of interest for a variety of reasons, and is discussed in detail in Chapter 4.) Most encouraging, however, was the multitude of ways in which students followed up on the paths that we deliberately provided into the lower levels. These “access roads” included:

**Prototyping Areas.** Included with the controller board used with Interactive C was an additional *Expansion Board*. This Expansion Board contained some circuit features that we didn’t consider essential to be placed on the main board; also, however, it contained a general purpose prototyping area. We gave the students instructional material that showed them how to connect their own circuits to the main microprocessor controller using the interface bus and prototyping area provided on the Expansion



Board.

A number of students attempted and successfully completed original circuit designs based on our interface suggestions. This activity was facilitated by the existence of the prototyping area on the Expansion Board and the documentation which accompanied it. These students were able to use the higher level of abstraction provided by the Interactive C controller while also delving into the underlying hardware and software details, knowledge they learned in order to get their original circuits to function.

**Course Notes.** As was just suggested, appropriate documentation played an important role in giving students access to lower levels of the robotic system, such as being able to attach their own custom circuitry to the stock hardware. This avenue was followed up in a number of ways.

Most importantly, we made separate presentation of “how to” and underlying-theoretical information in the course notes. This was to let students quickly have the information available needed to *use* the technology, so that they could learn by actually experimenting with materials and ideas rather than reading about them. But, detailed discussions of theory of circuit operation, motor and battery characteristics, and control concepts were made available for students to peruse at their discretion.

**IC Binary Feature.** In the second year of the Interactive C system, we introduced a feature that allowed students to write low-level assembly language code that could be transparently linked into their main high-level C programs. This allowed interested students to easily get underneath the level of abstraction provided by the C language into low-level microprocessor programming.

There was a dual motivation behind this feature. First, it simplified our job of writing necessary low-level drivers; also, it made these levels more accessible to the students. While only a few students made use of this capability, many more read the section in the course notes which described it and were stimulated to think about its possibilities.

To summarize, a critical concern in the development of our technology was to keep the lower levels of our system open to those students who were interested in them. This trans-

parency made our materials more versatile as a designer's medium, and more pedagogically rich to a student base with varied backgrounds and interests.

### **3.3 Sensor Design**

The development of sensors for use in the Robot Design project was perhaps the most guided by serendipity and opportunism, because sensors were the least well-understood aspect of the technology to both we, the project organizers, and our students. This led to valuable collaborative learning experiences in which the traditional roles of teacher and student were rendered moot; this became a powerful ingredient in the learning environment of our project.

When developing specifications for the robotic contest, we needed to ensure their solvability. We did this with the knowledge of particular sensor devices, their capabilities, and how they could be used to construct viable robot behaviors. Our challenge as course designers was then to provide authentic motivations for students to explore the possibilities of the sensor devices and how they could be successfully applied in a contest situation.

#### **3.3.1 Collaborative Learning**

As was mentioned, contests were designed to be solved with the use of specific sensor devices. But often we miscalculated in the details of how a sensor would work under actual practice conditions. This led to valuable scenarios in which we, the project organizers, joined with the project participants in learning how to apply a sensor for its intended—and often essential—purpose. Insofar as this occurred, we were no longer instructors but co-researchers working with the students toward the mutual goal of figuring out how to solve a piece of the contest puzzle.

#### **Noise Problems, Case One**

Problems with sensor noise and unreliability provided valuable learning opportunities from the beginning of our project, in the *King of the Mountain* contest. In general, sensor unreliabilities figured heavily into students' learning; the higher-level implications of this

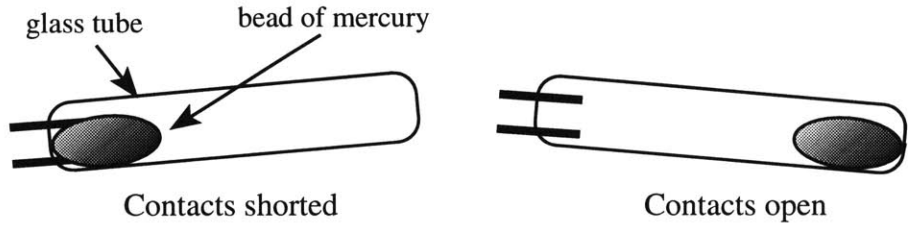


Figure 3-16: Mercury switch

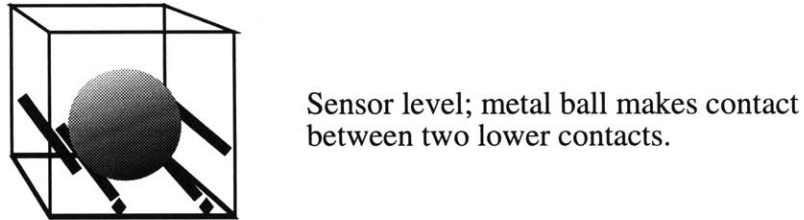
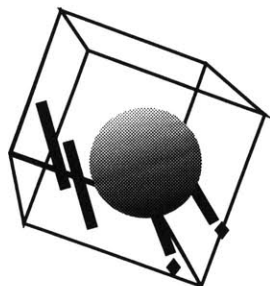


Figure 3-17: Rolling ball sensor in level position

in the design of their robotic systems are discussed in Chapter 4. Here, I will focus on the lower-level issues, in which we and the students learned how to get raw data from the sensors.

In the *King of the Mountain* contest, the central task was to climb the mountain. Our first choice of sensor for this task was the mercury switch—a small glass tube with a bead of mercury inside and two electrical contacts at one end (Figure 3-16). The mercury switch functions in a similar way as the carpenter’s level: when the glass tube is tilted, the bead of mercury slides to one end or the other. If the mercury slides onto the electrical contacts, the two are shorted together, since mercury is an electrically conductive material. The circuitry needed to determine whether the contacts are shorted together is trivial; electrically, the mercury switch is indistinguishable from any other sort of simple switch.

The mercury switches proved difficult to acquire. We were unable to find them in the quantity required to provide at least two in each kit (a minimum of two sensors would be necessary for a robot to be able to sense directionality on a two-dimensional surface), and they were expensive given our extremely constrained budget. We then discovered a sensor in a surplus catalog that seemed to fit the bill. The part consisted of a metal ball trapped inside a plastic box. Four metal contact rods surrounded the ball; depending upon the inclination of the plastic box, the metal ball would make contact between different pairs



Sensor tilted; metal ball makes contact between right-hand lower contact and right-hand side contact.

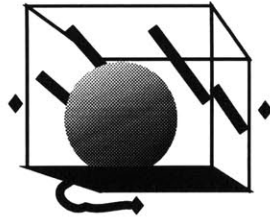
Figure 3-18: Rolling ball sensor in tilted position

of the metal rods, as shown in Figures 3-17 and 3-18. We decided to use the tilt sensors, and ordered enough to give four sensors per team, discontinuing our search for inexpensive mercury switches.

When the sensors arrived and we were able to examine them in detail, it became apparent that the design was not ideally suited to our purpose. Unlike the mercury sensor, the rolling ball sensor measured inclination in discrete segments of about 20 degrees. That is to say, it would take an inclination of at least 20 degrees from the horizontal for the metal ball to shift from the position shown in Figure 3-17, “level,” to that of Figure 3-18, “tilted.” This was a problem because robots needed to sense differences in inclination of far less than the amount of tilt needed to shift the ball between positions.

We suggested that students mount the sensors angled in a manner such that the ball would be on the cusp between the level position and the tilted position. If mounted in this way, a slight additional tilt in one direction would cause the ball to roll forward, and a tilt in the opposite direction would cause the ball to roll backward. This solution seemed adequate if not ideal. At least, the sensors would be viable and the discrete-sensing problem would be circumvented.

When students went to actually use the sensors, a new and more difficult problem was discovered. The problem had to do with the rate at which data was registered from the sensor and uploaded to the controlling computer (as discussed earlier in this chapter, off-board computation was used in the *King* year). When a robot was in motion across the playing surface, vibrations from the mechanical noise of the motors and geartrain, combined with the robots’ lack of suspension and a rough playing surface, caused the metal balls to vibrate



Sensor level; metal ball makes contact with bottom metal plate and left or right side contact.

Figure 3-19: Modified rolling ball sensor

inside the plastic can, to the extent that a given data reading had perhaps a ten to twenty-five percent chance of occurring while the ball was actually in contact with the pairs of metal contacts.

It was thus quite difficult to get data from the sensor while the robot was in motion. One undesirable solution was to stop the robot, take the inclination readings, and then proceed with a navigation correction based on the sensor reading. But students were reluctant to do this for the obvious reason that it would significantly slow down the progress of their robots.

As climbing the hill was the primary task of the contest, the problem was a serious one. Students worked on the problem, and came up with two different solution strategies, which were shared with the whole class:

**Physical modifications to the sensor.** This approach tackled both the discrete-sensing problem and the “bouncing ball” problem at once. The method was to remove the top cover of the sensor, replace it with a flat conducting contact, and then use the sensor in an inverted position (Figure 3-19).

This solution was popular because it was easy to do and gave the sensor much better overall performance. The discretization problem was solved by replacing the center two wire contacts with a flat plate. The bounce problem was not eliminated, but was significantly reduced by the new design (the ball seemed to bounce less on the flat plate than it had on the wire contacts).

**Addition of signal processing circuitry.** Several students worked out a solution using general-purpose “NAND” chips to latch the last data value reported by the sensor. The computer would then read the value of the latch rather than the sensor directly.

This solution worked because the latch could record the sensor state much faster than the computer could possibly sample it.

Both of these approaches produced solutions that provided working sensor technology for the purpose of hill-climbing. What is interesting about this story is that unanticipated constraints of the materials that were available forced the development of creative solutions. These solutions were developed by the students with our assistance. Of the two approaches which were satisfactory there was not an obvious better choice—for some students, one, the other, or both made the most sense.

### **Noise Problems, Case Two**

The essential properties of the rolling-ball sensor episode were repeated with a different sensor in the subsequent year, when we chose a sensor to allow robots to find the hockey puck in the *Robo-Puck* contest. Again, the importance of the following example is the illustration of how real problems were faced jointly by the project organizers and the project participants, dissolving the typical boundaries between instructors and students.

When developing contest ideas for what became the *Robo-Puck* contest, we learned about a newly available infrared sensor device that was inexpensive, easy to use, and because of its special properties, was far superior to simple light sensors for detecting light. The sensor was tuned to detect infrared light that was modulated (i.e., turned on and off very rapidly) at a particular frequency. Hence the sensor would detect this particular frequency of light and reject all other types of visible and infrared light interference. This aspect of the sensor's operation would make it ideal for our contests, with widely fluctuating ambient lighting conditions.

The infrared sensor device, Sharp Electronics part number GP1U52X, is shown in Figure 3-20, surrounded by a ring of infrared transmitter lights (a U.S. penny is located in the center of the circle as a size reference). For the *Robo-Puck* contest, we used the infrared LEDs to build a transmitter beacon that was mounted atop a rod implanted in a genuine hockey puck. Each robot-building team was given four of the Sharp infrared sensors that could be used to locate the puck.

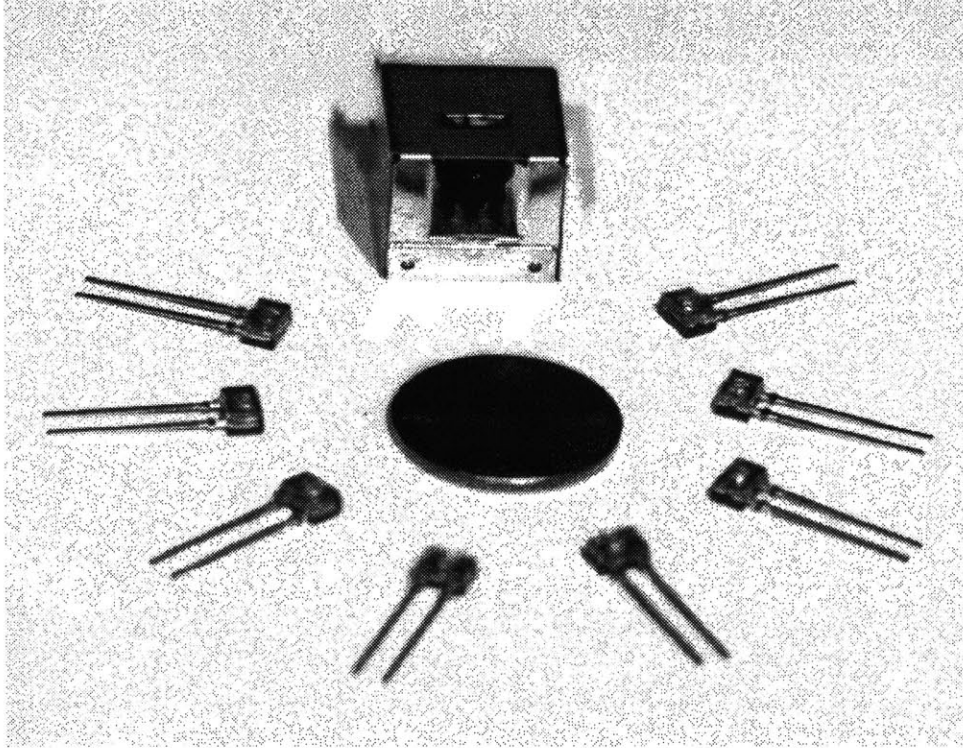


Figure 3-20: One Sharp Electronics no. GP1U52X infrared sensor and eight infrared transmitter LEDs, with a U.S. penny for size reference

The infrared sensors were designed to act as the unit that is built into a television or VCR to receive the infrared signals given out by the remote control. As such they had a very wide field of view—typically 180 degrees. In order to be useful for locating an object, the students put cylindrical shields on the sensors to limit the angle that light could enter.

In general, effective use of the sensor required creativity and experimentation. It was unknown what sort of optical shield would yield the best results, and in general it depended on the nature of the robot's design. Most students developed shields that restricted the sensors' field of view to between fifteen and thirty degrees, but one team in particular had an unusual application. While most robots were simply puck-fetchers, their robot used a rubber-band slingshot to fire a claw toward the puck at high speed. The robot, named *Shotgun*, employed an infrared sensor with a long narrow tube that restricted its field of view to perhaps two or three degrees. The robot needed that sort of accuracy because it only got to fire the claw one time—if it missed, the round was lost. The design used one infrared sensor with this extremely restricted vision and two others with more typical views; the

wider-angle sensors were used to initially locate the puck and the narrow one was used to lock on before firing the claw. (*Shotgun* enjoyed several competitive successes but was not reliable enough to ultimately win the contest.)

In principle, the infrared sensors were digital devices. When the light emitting from the puck was impinging upon the sensor, it should output an electrical value signifying “true”; otherwise, it should output a value of “false.” We did not have a chance to extensively test the sensors before building them into the contest specification, and it turned out that the sensors were not the ideal binary devices they purported to be. Rather than outputting a stable “true” or “false” signal, they were susceptible various interference which neither we nor the students understood very well. This interference caused them to produce a noisy signal that oscillated between the true and false states. Only by averaging this signal over time could one ascertain if the sensor was actually detecting the infrared emissions (if the signal was mostly true, then it was a good bet that the sensor was receiving the infrared signal.)

We and the students made various attempts were made to understand the nature of the noise problem and find solutions, including building light shields that prevented all stray light from entering the sensor, but these were largely unsuccessful. Maddeningly enough, one out of every eight or ten sensors worked ideally while the rest displayed this noise problem. The noise problem itself was not actually discovered until late in the month, causing a scramble among the students to trade their noisy sensors in for ones that “worked.” Students either were fortunate enough to obtain sensors that were free of the noise problem, or were forced to write software to filter out the noise.

Later, after the contest was over, a trivial hardware solution was found that completely eliminated the noise problem. It would have been wonderful had it been known earlier, but despite the frustration that the situation caused to the students and ourselves, the process of working with the noisy sensors was a rich learning experience, ripe with experimentation in hardware and software. Together, we were forced to be creative in their approach to dealing with a serious unanticipated problem in using a key component needed to solve the overall design problem. The answers were not known by anyone involved at the the time when a solution was needed. Certainly, now knowing how to fix the sensors so they



worked properly I would not give students sensors without the fix, but it was worthwhile to go through the experimentation process together. (We still do not understand why about 10% of the sensors worked properly without the fix.)

### 3.3.2 Motivating Understanding

In the previous two examples, it was clear to all involved that the sensor devices in question (inclination sensors and puck sensors, respectively) had to work properly in order for a robot to perform the contest task. But proper use of sensor devices was not always so evident; the following example shows how we successively improved upon a sensor technology, integrating its use in the contest to help students encounter the issue of *sensor calibration* in a explicit, contextualized way, thereby making the learning experience more relevant and meaningful.

Beginning in the *Robo-Puck* year, we employed light sensing in a way that made its use mandatory. At the start of game play, robots were placed above incandescent lamps embedded in the playing table surface. When the lamps were turned on, robots were allowed to begin playing the contest round. We decided that this method of triggering game play would be the simplest and most reliable method for ensuring that robots would be correctly synchronized.

Students were provided with two kinds of light-sensing devices: cadmium sulfide photocells (depicted in Figure 3-21) and phototransistors. The devices were satisfactory for the intended application of detecting the starting lamps, with the exception that students did not understand that the values reported by the sensors would be strongly affected by ambient lighting conditions. Quite a number of students did not make an attempt to either shield the sensors from ambient light or correct their readings by taking ambient lighting into account. This caused problems when lighting levels in the contest situation were different from those in the laboratory, resulting in some robots requiring special handling in the contest situation (they needed to be placed under a shadow so that the sensors would respond properly to the starting lamp illumination).

In the subsequent contest, *Robo-Pong*, the problem of sensor calibration became more significant. In *Robo-Pong*, we used shaded surface differences on the playing table as an

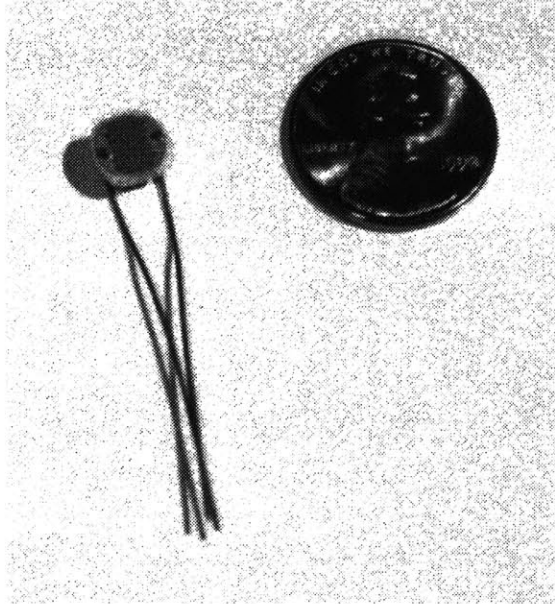


Figure 3-21: Cadmium sulfide photocell

important way for robots to extract information about their location on the table. One half of the table was painted with dark paint while the other half was painted with light paint. Robots could use light sensors to detect which half of the table they were on, and, with appropriate software, drive along the light-dark edge dividing the two halves (the *Robo-Pong* playing table is depicted in Figure 3-8 on page 76).

For detecting the surface reflectivity, we provided a discrete phototransistor to be wired as a light level sensor. We encouraged students to build reflectivity sensors by using this phototransistor in conjunction with an LED lamp oriented in such a matter that the light from the LED was aimed toward the ground surface and reflected into the phototransistor. (The instructions that we gave to the students for building this device are reproduced in Figure 3-22.)

Most students, however, chose to implement reflectivity sensors without the local LED illumination source. Sensors in this configuration relied on ambient room lighting to illuminate the table surface to provide the reflectivity measurement. These sensors did function—given constant room lighting, the measurements obtained over the light and dark portions of the table would correlate to the reflectivity of the table surface. By design, however, they were extremely sensitive to the particular amount of room lighting. If the

## Reflectance Sensor

Measures amount of light reflected from the red LED. Works most reliably when ambient light is shielded from LED/phototransistor pair. Same circuitry can be used to build break-beam sensor by angling components so that LED shines directly into the phototransistor.

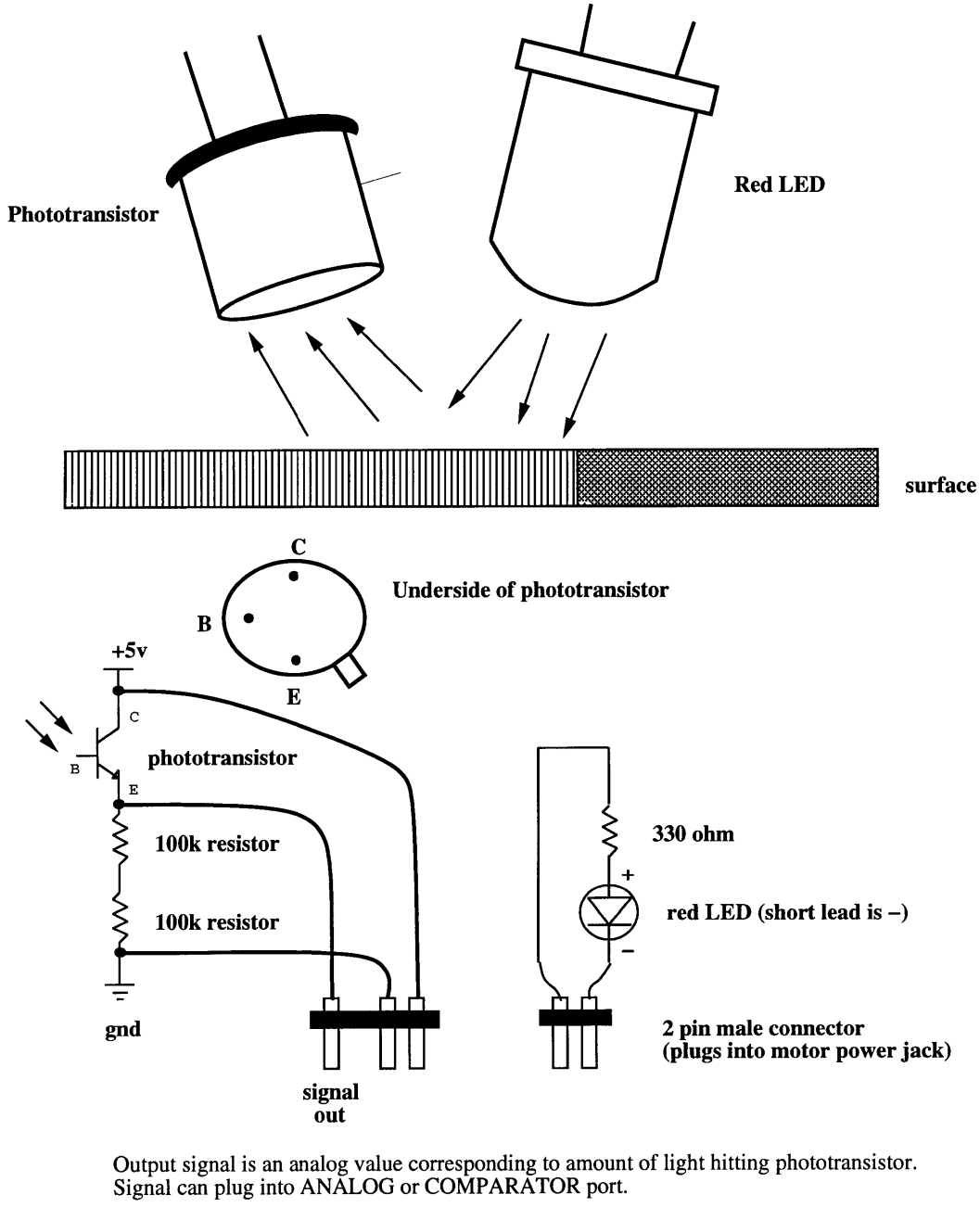


Figure 3-22: Assembly instructions for reflectance sensor from *Robo-Pong* contest handouts

room light level became brighter, then all values received by the sensor would shift in that direction.

Students didn't realize they were building devices with this property, and they did not heed verbal warnings informing them of this situation. The problem was abstract to them because light levels on the contest playing table remained fairly constant during the robot development process. On the day of the contest, however, the table was heavily illuminated by camera lights, causing many robots to malfunction as their software had no way of compensating for the change. The contest lighting was particularly harsh and caused many otherwise functional robots to fail. We felt that we had given the students careless treatment in this regard: while they were *told* that ambient light changes would be a problem, they were not given the chance to *experience* the situation until it was too late for them to do anything about it.

For the subsequent contest, *Robo-Cup*, we made several changes to ameliorate the problem. First, we decided that drastic changes in lighting from development to contest conditions were an unnecessary imposition, so we designed a permanent lighting fixture into the robot playing table. Second, we provided a better sensor for reflectance purposes—one that incorporated illumination and sensing into a single package. The wiring diagram provided to students based on this sensor is shown in Figure 3-23.

These actions did significantly lessen the ambient lighting problem experienced by the *Robo-Pong* students, but the new device was very sensitive to the distance between the sensor and the surface being measured. This caused students to have trouble using the device, since they did not anticipate this problem. Analytically, the sensor's performance can be explained by the fact that since the sensor was supplying the light it was measuring (assuming the sensor was well-shielded), the inverse-square law of light dispersion is itself squared: there is an inverse-square relationship in light levels between the sensor's transmitter and the surface, and again between the surface and the sensor's receiver. While we had made progress in creating a valuable sensing technology, the reflectance sensing concept continued to provide instructive difficulties to our students.

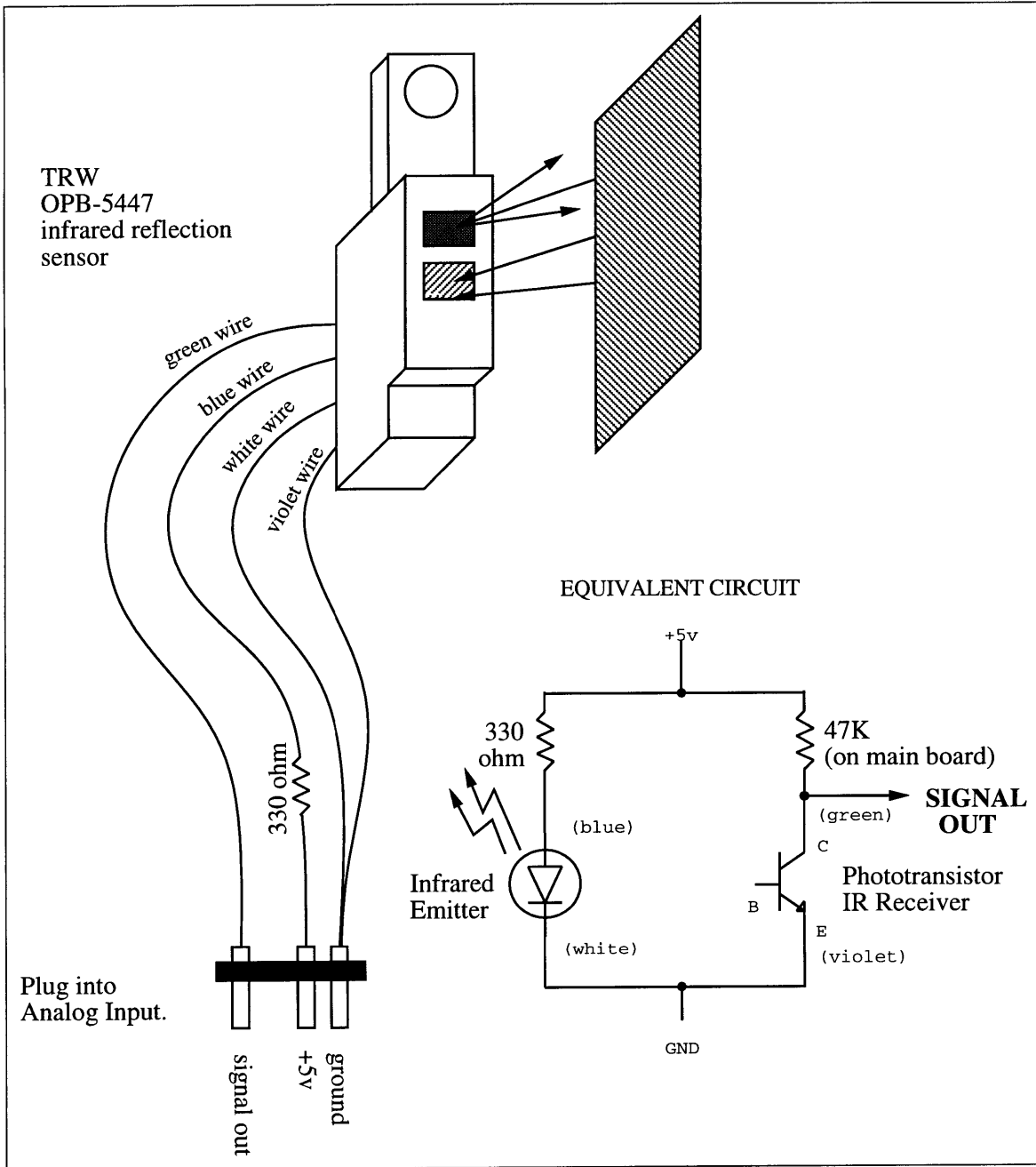


Figure 3-23: Wiring diagram for upgraded reflectance sensor provided to students in *Robo-Cup* contest

## 3.4 Summary

This chapter has examined three facets of the educational technology developed for use in the Robot Design project: the contests themselves, the hardware and software robot-building materials, and the sensors used by the robots to solve the contest tasks.

The contests set the stage for all of the subsequent work to be performed by the project participants. It was therefore critical that the contest puzzle be structured to bring about an appropriate and challenging intellectual inquiry. Several factors must be considered. The puzzle should encourage a diversity of solutions; ideally, it should not be evident even at the concluding competition which one is best. This is to teach a lesson of non-conformity and creativity that challenges the conventional assumption that problems have a single correct answer.

The contest should be at the appropriate level of difficulty for the students who are to solve it. If it is too difficult, students will be frustrated; if it is too easy, students may not give it their best efforts. As was illustrated, this goal can be difficult to reconcile with the goal of encouraging a diversity of solutions. Finally, a contest should send out a positive social message about appropriate uses of technology and appropriate collaborative behavior in the academic context. My personal belief is that the destructive potential of technology should be de-emphasized, and that students should be encouraged to see each other as academic resources rather than competitors.

The tangible hardware and software platforms should encourage students to learn by inquiry. To this end, they should be highly interactive and invite exploratory play, so that students learn by building actual models to test out their ideas.

Any educational technology will provide students with a particular level of abstraction, highlighting or making accessible certain ideas while hiding others. This is a desirable characteristic; no one can tackle all levels of a complex system at once. The designer of such a technology must think carefully about what issues he or she is exposing to the students, and ideally should provide insights and invitations to the ones being hidden. This model supports students with different backgrounds and interests, encouraging further learning rather than suppressing it.

The lesson of the sensor investigations is that a learning environment is strengthened rather than weakened by surprises. Many of the sensors we used in our project were “insufficiently” tested before we gave them to the students; because of this, the design projects became joint inquiries in which students and organizers worked side by side to solve critical problems in a short time frame. This type of learning environment is typical of research situations but not academic ones; in our case, it had a powerful role in stimulating creativity and providing a motivational relevance for the students’ work.

# Chapter 4

## Ideal and Real Systems

The product of each robot design project is a particular robot that interacts with its environment—the contest playing field, the game objects, and the opponent robot or robots—in particular ways. The robot thus embodies and becomes part of a complex system of interactions that defines its existence.

In this chapter I will explore the biases that students bring to the task of designing their robot, a member of such a system. Of particular interest is students' recurring inclination to build robots that will perform properly only under ideal conditions. Students repeatedly build robots that are not well-equipped to deal with the exigencies of the real world, but rather with the specifications of an idealized, abstractified world—a world that the robot designers would like to believe is a close representation of reality, but is not. This result points to limitations in the set of ideas about technological systems and methods that comprise the core of the engineering curriculum. What surprises many participants is that these ideas do not map well to the challenge of designing a robot to play in one of our contests.

This chapter has four sections. The first section presents an overview of the task of designing a control system for a contest robot, and the sort of ideas that students generate in coming up with their own solutions. This section presents a composite set of results from a number of different students' work; this composite portrait is representative of the sorts of issues encountered by most of the students in the classes.



The second section presents a case study of a pair of students who formed a team to build robots for two consecutive years of the Robot Design class. These students believed that control could best be achieved by having the robot keep track of its global position on the playing field at all times. In the face of failures, these two students' views held with a greater degree of clarity and perseverance than most, yet in important ways they are “cut from the same cloth” as others.

The third section presents the approach taken by one student who developed the premise of simulation as a robot development tool. While this student's methodology was somewhat unusual, his approach reflects biases showed by many of the students.

The methodologies chosen by the student robot designers in this study are in many ways a product of the curriculum of the modern engineering university. The final section analyzes the content of this curriculum, pointing to evidence that explains students' choices, and why their intuitions about what sort of control would be effective were largely wrong. In doing so, I point toward a direction for revising the curriculum to encompass a broader range of ideas about systems and control—one that would be more effective in preparing students for demands of modern technological systems.

## **4.1 Introduction to Robotic Control**

The data for the issues discussed in this section is taken primarily from the students' work in the *Robo-Pong* and *Robo-Cup* design contests. In these two versions of the project, both the design task and the materials used by the students were sufficiently rich to allow a variety of approaches to become manifest. In contrast, the *King of the Mountain* and *Robo-Puck* contests were too simple for these issues to arise.

### **4.1.1 The Omniscient Robot Fallacy**

Students' beliefs in how a robotic system should be understood and controlled is revealed in how they approach the central task of developing their robot's strategy.

Some students approach the design of the robot task from what might be called an egocentric perspective, as if they were thinking, “what would I do if I were the robot.”

These students imagine themselves at the helm of their robot, driving it around the table and performing tasks. This approach leads to ideas like the ones in the following narrative:<sup>1</sup>

Our strategy at present seems fairly well developed. First, [our robot] will immediately locate the other robot, roll over to it while lowering the forklift, and try to flip the other robot over. If it can knock it over, or even over the playing field wall, the robot has essentially won, as it can now freely place balls in the goal without interference. If this attempt fails after ten seconds or so, our robot will disengage, roll over to the ball dispenser, park, and get balls and shoot them at our goal with a cannon mechanism. Should the cannon prove unreliable, we may have to collect balls and deliver them manually, but having a cannon would be much more nifty, and would avoid a lot of line- or wall-following difficulty.

It is apparent in this discussion that the student has not thought through, at a mechanistic level, the questions of how the robot might actually perform these tasks; he or she is imagining that it is simply a question of navigating as one might drive a car.

Many students initially generated such fantastic ideas about what sorts of activities their robot might be capable of performing. Here is another student's ideas about how to solve a contest:

I played around with a structure that was to be the "Goal Emulation Unit," which was an idea of mine. We had decided it would be a neat strategy to have something on our robot that looked like a goal, so that it could stand in front of the opponent's goal, and hopefully the opponent would place its balls into our robot's unit . . . We had decided in the beginning that we wanted a fast robot. I thought it might be neat to chase after our opponent as soon as the round started and either steal his ball (assuming that it goes to the dispenser first) by pushing him out of the way, or having some sort of long arm that reached above the other robot to get the ball.

Written at the end of the second week of the project, these two scenarios are representative of students who have difficulty imagining what sort of information with which the robot will be working, and hence what sort of strategies or algorithms will yield realistic performance results.

Quite a large number of students initially generate ideas like these. If an idea requires a complex mechanical structure, such as the "Goal Emulation Unit" described in the previous

---

<sup>1</sup>The narratives in this and the following chapter are taken from the students' written design reports.

passage, students will discard their ideas as soon as they're unable to build the mechanisms that are required to accomplish it (which is typically early in the project). If the complexity resides in the software design, however, many students continue to imagine their robot to be capable, in principle, of very sophisticated behaviors—until they attempt to implement those behaviors in actual code. For example, in the following comment, a student believes his robot to be all but completed, even though he has not begun programming it:

As my team's robot stands now, it is 99% near completion. Only a few structural changes (might) have to be made and one or two sensors have to be attached; otherwise all that is left to do is program the baby.

Evidently, this student is not including the programming task in his estimate of the robot's percentage of completion. This attitude is typical; not only do students consistently underestimate the time-consuming nature of the programming task, but until they attempt the programming task, they don't realize what sort of robot behaviors are feasible to implement.

Other students more readily acknowledge the unknown nature of the programming task. Most fall back on previously learned approaches for dealing with complexity, as this student explains:

In addition to assembling part of the robot base, I have been working with Interactive C, writing some code but mostly pseudocode, and relying heavily on abstraction and wishful thinking.

This student has not gotten into the hard work of getting his robot to perform real behaviors with but one week before the contest.

A second group of students is more cognizant of the practicalities of the issue, realizing from the start that they need to get a firm grasp on their robot's potential capabilities. After describing a number of different strategic possibilities, one student writes this statement after the first week of work:

However, [my teammates] and I realize that the only way to find out what strategy works best is to build hardware that is robust and is powerful enough to implement a lot of different strategies and then implement the actual strategies in software. Our immediate goal is to construct powerful hardware both electrical and mechanical and then write the software to control it and implement the different strategies.

This hands-on approach to developing a robot's program is favored by fewer of the students.

### 4.1.2 Understanding Sensing

Few participants in the class have had prior experience working with electronic sensors. When they are first introduced to sensors, either through the lecture-style presentation we hold, or in the course notes, many students expect that sensors will provide clear and unambiguous information about the world. As one student explains:

We plan to build the sensors and motors first, so that I can start playing with the control software and writing subroutines so that we can abstract away the working of the sensors as much as possible.

This student speaks for many when he proposes the use of modular abstraction to deliver clean information about the robot's state to some higher level control program. However, there are many reasons why sensors don't work as straightforwardly as the students would like; some of these have been discussed in Section 3.3. Students are then surprised when they attempt to program a functional behavior into their robots. As one explains:

I decided to write the wall-following function which was the example in the text. It turned out to be much more difficult than the text had led me to believe. First of all, I needed to figure out how much power went to the inside motor in a turn. This took more than a few hours to finally debug and figure out what kind of radius each power differential would give. Another thing I had to figure out was the thresholding and logic statements needed for two sensors, one at the front and one at the back. This thresholding problem took me more than a while to figure out, and in the process, I found errors in my logic statements. . . This method has not proved anything, because in its first incarnation, the robot was low on batteries. . . [the] present program hasn't been tested yet, because. . . we took the robot apart to strengthen each subsection, as in the wheel assemblies, and the subframe to connect them.

In the end, this student has concluded that his tests are moot, since the robot had a low battery level when the tests were being done. Even if he hasn't solved the particular problem at hand, though, it is clear from his narrative that he learned a great deal about the nature of the problem situation.

Here is another account of sensor troubles written by a participant who began systematic sensor experiments before the “crunch time” of the end of the project (and hence had time to write about it):

The sensors have plagued me with problems from the start. The photoresistor sensors work great and I think that we can rely on them for the polarized light sensors. However, I think that the polarized light filters are not very reliable in tests that we have been using. It seems that the photoresistor in the polarized light sensor can't tell the difference between facing the opposite goal and being far away from its own goal. I think the problem lies in ambient light reaching the photoresistor. I think the only way to solve this problem is better shielding, but I think that the ambient light problem is unsolvable unless the contest is [held] in a pitch black room.

The reflector sensors seem to work all right, but when we try to shield them from ambient light, we get random reflectance and therefore detection when we aren't supposed to. The problem seems to be solved when we place the sensor inside my sweater which is very “bumpy” and keeps light from randomly reflecting off of things. We are going to try to fix this problem this week because our strategy really depends on them.

These two cases are unusual only in that the respective work took place early in the course of the design project. They are quite typical, however, of what nearly all of the students encounter when they engage in the actual programming task. Students then discover all sorts of interrelations and complexities that arise, aside from implementation issues, that make it difficult to abstract sensor data into clean information upon which control strategies can be devised. They realize that sensor data are erratic; that there are dependencies on hard-to-quantify properties of the mechanical system; and that how one solves the control problem is a factor in the ultimate reliability of the sensor data. For example, here a student finds the solution to achieving a good edge-following behavior lies in accomplishing several changes:

When we first tested the robot's ability to detect a green/white boundary, the robot had difficulties. After some software changes (by [my partner]) and mechanical changes by me of placing the sensors even closer to the ground and also placing shields around the sensors, we have no problems detecting the boundary.

Students had a tendency to establish hard-coded thresholds for interpreting the meaning of sensor data. In *Robo-Pong*, two sensor varieties were particularly problematic in this

regard: motor current sensing, which determined if a robot's movement was impeded (causing its motors to stall and hence draw more current) and light sensing (for determining on which side of the playing table the robot was located).

The trouble with the motor current sensing was that the numerical reading that indicated a stalled motor would change as a function of the voltage of the motor's battery. When the battery voltage fell, as the battery gradually discharged with use, the current sensor would return higher values—indicating that a motor was stalled when it in fact was not.

Several teams of students failed to adequately deal with this problem and created robots that acted as if they were stuck when, in fact, they were just driving uphill. This occurred because uphill driving would necessarily cause additional load on the motors, which the robots' programs would interpret as the case where the robot was indeed stuck. In the main contest performance, one robot in particular suffered an amusing yet incapacitating failure mode of this type. It would repeatedly drive forward, stop, and back up as its program interpreted the motor current sensors as indicating that the robot was stuck. The robot seemed to be battling an invisible enemy as it tried again and again to collect the balls in its trough, each time backing up after advancing a few steps.

Students in *Robo-Cup* were warned of this problem and some concluded that the motor current sensing was simply too unreliable to be a part of their design. As one student explained:

I wrote servo-like routines to monitor the clamp's position, and attach or release it as required. The original plan was to use the motor force sensing to determine if the motor was stalled, and control its motion like that. The force sensing, it turns out, is very unreliable, so we chose to install a potentiometer [a rotational position sensor] on the axle instead, and use that to keep track of position.

This student has performed an analysis that this sensor is too erratic to be trusted, but most did not discover the sensor's failure mode. Students simply did not anticipate the sensor's fundamental unreliability.

The light sensors used in *Robo-Pong* were problematic in that they relied on room lighting to illuminate the table playing surface. Hence the values registered by sensors—of the amount of light reflecting off of the playing surface—were dependent upon the amount of ambient light in the room.

We as organizers were aware of this situation when we designed the contest and chose sensors for the robot-building kits, but we underestimated the impact on students' robots that resulted from changing the light levels for the contest performance. Anticipating that the lighting conditions would be different on the eve of the contest than they were in the development laboratory (since we intended to use camera lights), we had informed students that they should either shield their sensors from ambient light and provide a local source of illumination (e.g., a light emitting diode or flashlight bulb), or write calibration software that would take into account the level of ambient lighting.

However, we failed to provide a readily accessible way for students to try different lighting conditions during robot development, and the camera lights we used were particularly harsh, causing wide fluctuations in light levels even from one side of the table to the other. It was an unfortunate event as a number of otherwise competent robots became unreliable in the face of the extreme lighting conditions.

This situation led us to propose to a number of changes in *Robo-Cup*. First, we provided a light sensor that incorporated its own light transmitter, so that it would be substantially easier for students to deploy light sensors that were well-shielded from ambient light. Second, we incorporated a diffuse lighting fixture into the contest playing field itself, so that lighting conditions would not change drastically from development to performance situations. Also, we modified the programming environment to make it easier for students to construct sensor calibration routines: we made it possible for calibration settings to be “remembered” throughout multiple performance runs.

Nevertheless, students continued to have similar difficulties dealing with the sensor calibration issue. Its implications were underestimated and the concept seemed unfamiliar to them.

### **4.1.3 Models of Control**

The overall strategies that students used to control their robots lie in a spectrum between two categories, which I shall call *reactive* strategies and *algorithmic* strategies. In addition, each of these strategies is built from two lower-level methods, *negative feedback* and *open-loop* approaches. Analysis of the various approaches that students take toward implementing

an overall strategy to control their robots reveals interesting biases, as was the case in analyzing students' approaches to understanding and deploying sensors.

These descriptions that categorize the students' robots were created in a retrospective analysis of the students' work on them; students were not aware of these categories during the course of their projects. I believe that students would recognize these distinctions if asked, however.

## The Higher Level: Reactive and Algorithmic Control

The issue of control manifests itself in at least two aspects of the Robot Design project: in the development of the robot's higher-level strategy from a conceptual standpoint, and in the actual programming of that strategy into the machine through the process of writing computer code. In the conceptual area, the *algorithmic* control method is by far dominant over the *reactive*.

The algorithmic method is characterized by a program that dictates a series of actions to be taken as its central feature. For example, here is an algorithmic main program loop from a student's solution to *Robo-Pong*. The code waits for the starting lamp to turn on (`start_machine`), then executes a routine to turn the robot downward (`rotate_down`), executes a routine to sweep across the ball trough (`grab_balls`), waits for two seconds, and then drives to the other robot's side of the table (`other_side`):

```
main()
{
    start_machine(); /*starts the machine up when*/
                    /*the light is on*/
    rotate_down();
    grab_balls();   /*go into our grab balls routine*/
    sleep(2.0);    /*wait two seconds*/
    other_side();
    finish();      /*block other machine or */
                  /*knock more balls over?*/
}
```

In this program, the contest is solved by executing a specific series of actions. Here is another example of the same approach, albeit slightly more complex, for solving *Robo-Pong*. The names that the student has given to the subroutines of his program are fairly self-explanatory as what action each subroutine performs:



```

game()
{
    start();
    Turn_to_Uphill();
    Go_Up_and_Orient_Plateau();
    DoorOpen();
    WalkPlateau();
    DoorClose();
    JiggleLeft();
    DoorOpen();
    FollowRightWall();
    DoorClose();
    JiggleLeft();
    DoorOpen();
    FollowRightWall();
    DoorClose();
    JiggleLeft();
    FollowRightWall();
    Alloff();
}

```

Code written using the same algorithmic method can be found for *Robo-Cup* robots.

Here is an example of such a main routine:

```

starting_routine()
{
    /* drive forward full speed */
    drive(7);

    /* wait until sense edge */
    while(abs(ana(reflectance0)-white0)<40) {}

    /* drive a little further */
    sleep(distance(2.0));
    alloff();

    /* turn a little */
    pivot_on(7);
    msleep(80L);
    pivot_off();

    /* drive until front bumper triggers */
    while(!(digital(front_bumper)))
    drive(7);msleep(20L);
    drive(0);

    collect_balls(3);
    goto_goal();
    goto_button();

    collect_balls(3);
    goto_goal();
    goto_button();

    catch_ball();
    goto_goal();
}

```

```
    goto_button();  
    return;  
}
```

As illustrated, the algorithmic control methodology consists of a list of instructions to be followed in order to accomplish the intended task. Each instruction may consist of a set of sub-actions, but the overall principle is to decompose the task into a sequence of activities to be followed, much like one might construct an algorithm to sort items in an array or compute a statistical quantity.

The trouble with the algorithmic approach is that there is no recovery path if things do not proceed according to the plan. Not only is there no way to recover from unexpected circumstances, but there simply is no provision for detecting that something out of the ordinary has occurred. These solutions have a problem that is reminiscent of comical factory scenes in which the machinery continues to plug away at an assembly process that has gone completely awry.

Students often deceive themselves as to the reliability of their algorithmic solutions. Suppose each component of an algorithmic solution has a 90% likelihood of working properly on any given occasion. One might think that the overall solution would have a similarly high probability, but this is not the case, since the probabilities multiply together with the overall result decreasing in likelihood with each step. If the algorithm has six independent steps, then the overall probability of proper functioning is only 53%—not nearly as good as the 90% chance that each individual step has of working properly. During testing, an algorithmic solution might require a gentle prod or correction here and there, and students don't realize that their machine actually needs this sort of help a significant portion of the time.

The reactive methodology, on the other hand, is characterized by actions that are triggered through constant re-evaluations of external conditions (i.e., sensor readings) or internal conditions (program execution status). While the algorithmic method does make use of external stimuli, the reactive method is more *driven* by this data, rather than by using it as part of a predetermined activity sequence.

There were no purely reactive robots in any of the contests; instead, some students combined elements of a reactive strategy into an overall algorithmic framework. An

example is the robot *Crazy Train*, which was the champion of the *Robo-Pong* contest. The robot was a collector-style machine that scooped balls into its body.

*Crazy Train* worked as follows. As the round began, it took a reading from its inclination sensors to determine its initial bearing. If it determined that it was pointed toward the top of the hill, it would drive forward until it crossed the center plateau; in the process, it was likely to knock a center ball onto the opponent's territory (thus gaining an early advantage—insurance in case something went wrong later in the round). If it was not aimed uphill, or, after having executed the initial upward movement, it would drive to its own ball trough and collect balls into its body. After making one sweep of the trough, it would drive onto its opponent's side of the table, delivering the balls it was carrying onto the opponent's side. In addition, when ten seconds remained in the round, *Crazy Train* checked to see which side of the table it was presently on. If for some reason it was still on its own side of the table, it would execute a “panic” routine which would attempt to drive onto the opponent's side before the round ended.

There are two aspects of *Crazy Train*'s program that characterize the reactive method. Firstly, *Crazy Train* had an unusual response to sensing its orientation at the start of each round. Depending on the orientation, the robot chose whether to go for the potentially risky maneuver of immediately knocking a ball over the top, or the safer maneuver of traveling to its trough to collect balls. It chose to go for the early lead by knocking over a center ball only if it is was efficient maneuver—when the robot was already aimed toward the center plateau. In this fashion *Crazy Train* reacted to an external condition and choose an appropriate response.

The other feature of *Crazy Train*'s program is its panic behavior when the contest round was about to end. Here the main program, albeit an algorithmic one, is interrupted by another behavior which is triggered by a change in internal condition—a timer keeping track of remaining contest time. If necessary, the panic behavior attempts to drive onto the opponent's side (if the sensor registers that the robot is already on the opponent's side, which should be the case if the algorithmic task has been completed successfully, then no additional action is needed). The students who created *Crazy Train* implemented the panic behavior by exploiting Interactive C's multitasking capability. Their program had

both the algorithmic behavior and the panic behavior as independent program tasks; at the appropriate time, the algorithmic task was terminated and the panic task was initiated (a third program task was used to coordinate this activity).

*Crazy Train* is perhaps the most dramatic example showing the usefulness of reactive program strategy, and, not too surprisingly, it was an overall contest winner. Its success can be attributed to a combination of its reliable and effective ball-gathering algorithmic strategy bolstered by useful reactive behaviors: there were a few game rounds in which the algorithmic strategy failed but the reactive behaviors delivered a winning performance. It also is representative of the extent to which reactive behaviors were used at all; that is, where reactive methods were used, they were simplistic, though sometimes effective, responses to a limited number of potential situations.

### **The Lower Level: Negative Feedback and Open-Loop Methods**

The issue of reactive versus algorithmic control as it applies to the macro level of overall robot strategy has its analog at the micro level, the mechanism by which the strategy is actually carried out. The micro level consists of the low-level methods for driving the robot's motors to accomplish an intended action.

For example, the macro tasks of collecting and delivering balls in both *Robo-Pong* and *Robo-Cup* can be decomposed into micro- or sub-tasks such as climbing uphill or downhill, driving along the retaining wall of the playing field, or driving along a light-dark boundary painted on the playing field floor. The contests were specifically designed to be decomposable into subtasks like these—ones that are feasible to be solved with the technology provided in the robot kit.

I distinguish two low-level control methods that were implemented in the robots. Most robots used a mixture of these two methods rather than one or the other exclusively.

The two methods are *negative feedback* and *open-loop control*. In the negative feedback method, a robot responds to a situation with a small corrective action. Through repetitive application of the small maneuver, the robot achieves the overall action desired by its designer. For example, a robot might follow along the edge of a wall with the use of a sensor that measures the distance to the wall, repeatedly turning to move toward the wall if

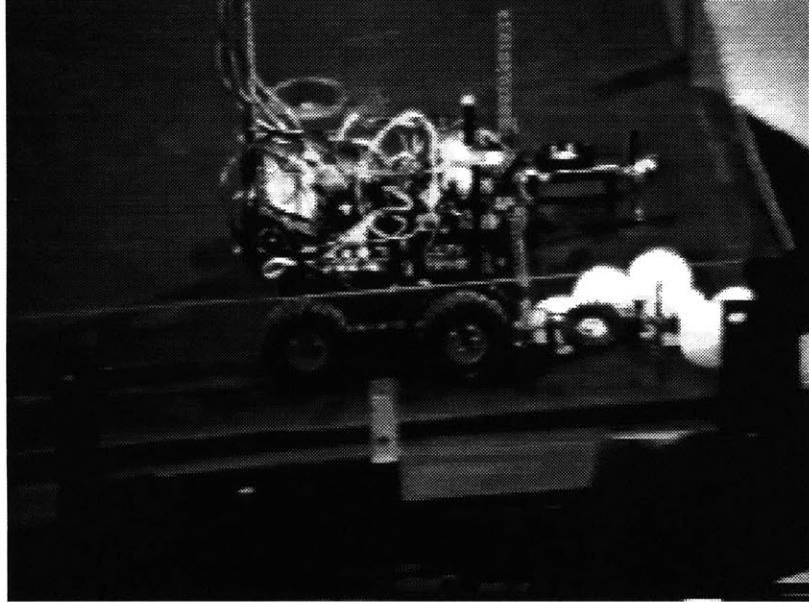


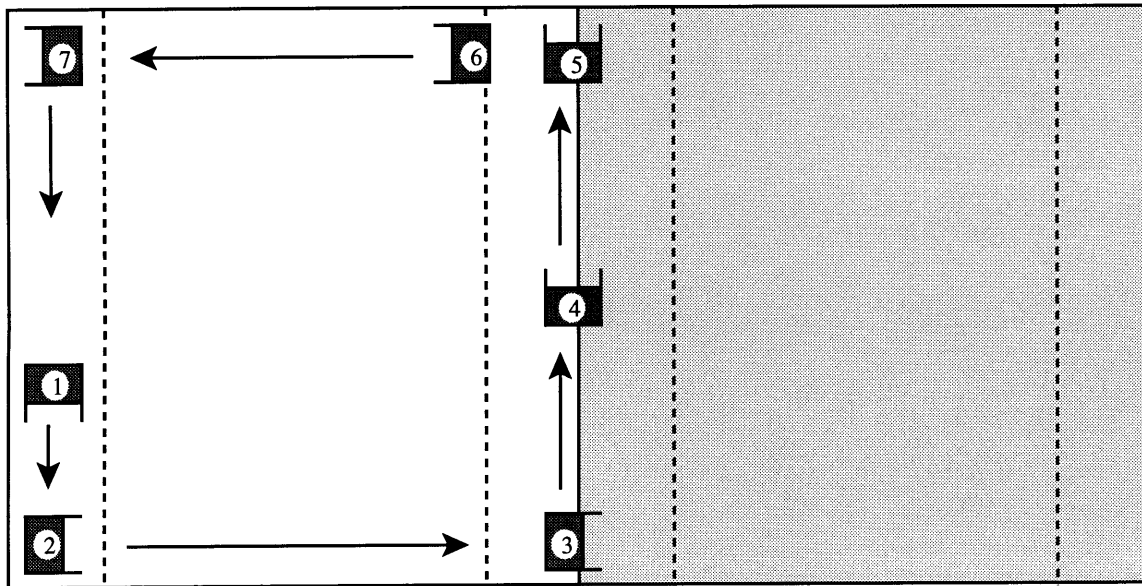
Figure 4-1: Photograph of *Groucho*

it senses that it's too far away, or turning to move away from the wall if it's too close.

In the open-loop method, the robot responds to a situation with a single action which the designers presume will bring it to a new state with respect to its surroundings. A typical action would be a predetermined, timed motor movement, usually lasting one or more seconds. For example, the robot might turn its motors on for a period of two seconds (this value would be experimentally determined) in order to pivot ninety degrees and thereby negotiate out of a corner.

Few of the robots' activities can be considered open loop in the formal sense because the open loop-like maneuvers are part of a larger robot strategy that generally involves sensor input and hence is not truly open loop. However, if a robot takes an action that does not accomplish the desired result, and the control algorithm cannot recover gracefully from this circumstance (nor makes an effort to detect it), the action is likely to be representative of open-loop thinking.

An example will help make these ideas more clear. *Groucho*, a *Robo-Pong* robot (pictured in Figure 4-1), employed the following strategy. It would drive into its trough at the beginning of the round and then execute a series of motions that would make it drive in a rectangular pattern, scooping balls from its trough and depositing them onto the



1. Robot drives along trough, scooping balls into its grasp.
2. Robot negotiates corner and turns uphill.
3. Robot stops at dividing edge, allowing balls to fall onto opponent's side.
4. Robot drives along edge to opposite side of the table.
5. Robot hits opposite side; turns to drive downhill.
6. Robot drives downhill.
7. Robot hits bottom wall and negotiates corner; continues pattern in step 1.

Figure 4-2: *Groucho's* strategy as played in *Robo-Pong* contest

opponent's territory. *Groucho* used an overall algorithmic strategy, repetitively performing the sequence of these steps, as illustrated in Figure 4-2.

In implementing this strategy, however, the students who built *Groucho* used mixed techniques of feedback loops and open-loop control. At points 2 and 7, when negotiating corners, they used feedback from the playing field walls to do so: it would hit the wall, back up, make a small rotation, drive forward, hit the wall, and try again. Typically the robot would strike the wall three to five times until it rotated sufficiently.

At points 3 and 5, however, the robot executed timed movements to accomplish the rotations. The designers had experimentally determined how much of a rotational movement, measured in fractions of a second, would be required to perform the desired turn, and then hard-coded these timing constants into their program structure.

The question arises as to why the designers of this particular robot chose to implement a feedback-controlled turn during some points of their robot's path and open-loop-controlled turns at others. There is a clue to the explanation contained in the source code written for the robot. As mentioned, in the trough area, the students used feedback from touch contact to negotiate turns, but in the plateau area, they used timed turns rather than feedback from the center dividing line. It turns out, however, that the students attempted to base the plateau turns on feedback from the dividing line, but at some point commented out the code to do this, replacing it with a simple timed turn. Here is the relevant code excerpt from their program. The line bracketed with `/*` and `*/` is the one that the students commented out; the subsequent line, `left(100,1.);`, implements the timed turn that replaced it.

```
printf("Crossed the border.\n");
beep();beep();
time=seconds()+1.1;
/* while ((left_reflectance!=side)&&(seconds()<time))
   {bk(0);bk(1);fd(2);fd(3);} */
left(100,1.);
ao();
printf("On the Line...\n");
```

It can be presumed that some kind of difficulty or unreliability was encountered when using the reflectance sensor to determine when the robot had turned enough, so that the students substituted the open-loop turn movement instead. This robot's mechanics were sufficiently well-designed and reliable that the robot's performance through the turn remained

consistent through the contest performance, but this is not typically the case.

The example of *Groucho* brings out a tension between the negative feedback technique, which ultimately can be more reliable, and the open loop technique, which is often quicker to implement and can result in faster performances, but is more subject to failures. In these excerpts, two students comment explicitly on this matter:

We decided to simplify our strategy, making our robot more reliable and easier to build. We now plan to have our robot aimed at the dispenser at the outset of the contest. The robot will then move forward until it senses that it has crossed the white/green boundary, using an infrared reflectance sensor. The robot will then turn and press the button repeatedly, catching and shooting each ball into the net. *This strategy involves much open loop control, but greatly simplifies our control system, making the robot more reliable.* (Emphasis added.)

I discussed with my teammates that if we're going to be a fast machine, then we'll have to implement a lot of open-loop [controls] interleaved with closed-loop feedback. Otherwise if we used mostly closed-loop [methods] to "feel" our way around the playing field, we'd be as slow as our opponents.

These students were conscious of the trade-offs involved, but others were not. Many students used open-loop movements based on timed actions; robots built from these constructs faced problems from a number of sources. The most significant variable was the level of charge in the battery powering the motors. As the movements' durations were experimentally determined, they were highly dependent upon battery performance.

Robots were powered with lead acid rechargeable batteries, which have a characteristic discharge curve as shown in Figure 4-3. As the battery is used, its voltage (which determines the amount of power delivered to the motors) gradually decreases. Often this deterioration was not immediately noticeable, so students were not necessarily aware of the situation as they were developing their programs. Finally the battery would reach a point, the end of its usable life, when the voltage would drop off rapidly and the battery would be obviously "dead." However, during the usable period there would be a significant difference between the initial voltage and final voltage levels. (Some other types of batteries, such as nickel cadmium, have discharge curves in which the voltage level remains nearly constant for the bulk of the batteries' usage, until the level drops off sharply, rendering the battery suddenly useless. Use of these batteries would have made the discharge effect less pronounced.)



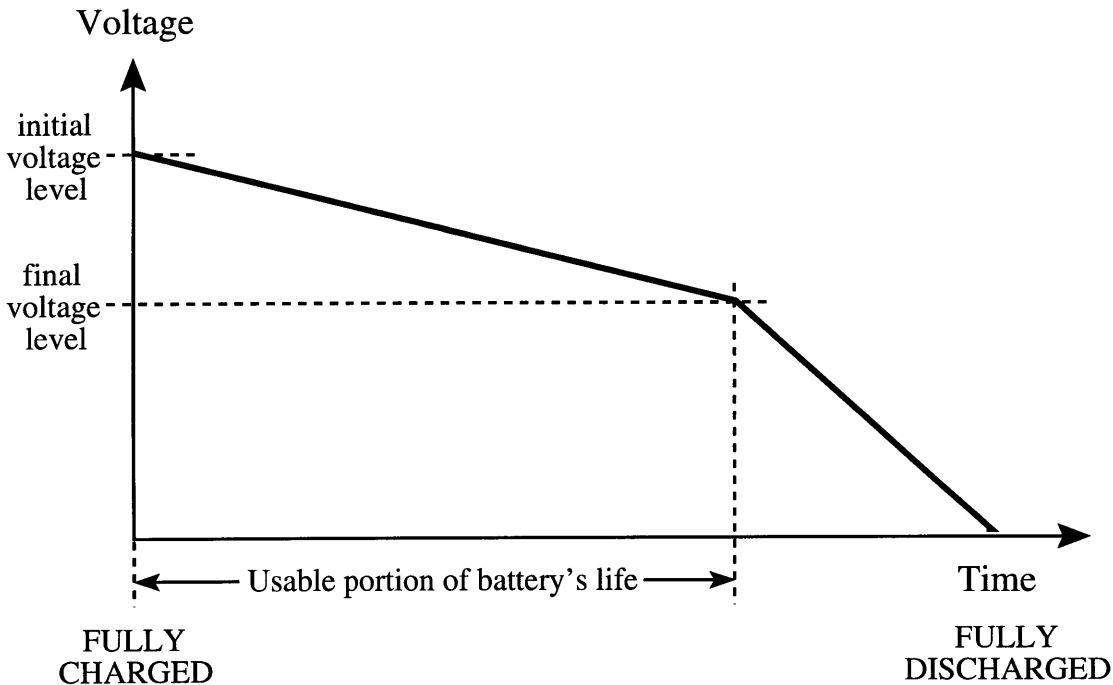


Figure 4-3: Idealized lead-acid cell discharge curve

Most students only realized that the algorithms they had created to solve the contest were dependent upon battery level when they would swap a partially discharged battery for a fully charged one, and discover that their program no longer worked correctly. As mentioned earlier, this realization did not occur until quite late in the robot's development—when there were only a few days or even hours left before the contest—and hence it was too late for them to make a substantive change in the robot's strategy (that is, substituting feedback-based movements for open loop ones). Students would deal with the situation by doing final program development—i.e., tweaking of timing constants—with batteries as fully charged as possible.

The degree to which this problem would manifest itself in the students' designs was largely a function of the gear ratio (that is, ratio between motor rotational speed and final drive wheel rotational speed) used in the robots' mechanics. If the gear ratio was such that the motor was driven in a "comfortable" portion of its power range, a slight change in battery level would not have a drastic effect on the performance of the motor. On the other hand, if the gear ratio caused a high amount of drag on the motor, slight changes in battery level would have a large effect on motor performance.

We as organizers only recognized this situation in retrospect, and hence were as confused as the students as to why some robots seemed to have little difficulty in performing reliably (at least in this regard) while others were quite troublesome. We were more savvy of these control issues when running *Robo-Cup* than *Robo-Pong*. In an attempt to encourage students to use feedback-based control rather than open-loop control, I wrote an additional chapter of the course notes for the *Robo-Cup* students that explained the differences and trade-offs between open-loop and feedback-based control. I also discussed the need for calibration of sensor values to local conditions.

A number of students became fixated on using feedback control in surprising ways, however, spending a considerable portion of development time creating a feedback controller to make their robot able to drive in a straight line. Here is how one student explains his system:

The robot is now able to drive in a straight line thanks to its optical shaft encoders. The software continually samples both left and right sensors and adjusts the exact power level to the motors to compensate for any fluctuations in the motion of the robot. The algorithm itself is a combination of differential analysis and Newton's method, along with an "adjustable window" of correction. Simply stated, analyzing differentials gives a reasonable algorithm for adjusting the power levels of the motors, and the adjustable window minimizes excessive wobbling. This algorithm gives us as much accuracy as the resolution of the shaft encoders permits, while keeping wobbling reasonably low.

The ability to drive perfectly straight, however, often does little to help a robot in its overall ability to solve the contest task. While the activity of driving straight is based on negative feedback control, a strategy that employed this ability in a central way should be considered open-loop at the next higher level, as the robot would be performing a task with little feedback from the crucial environmental features! So these students' belief that they were using the preferable feedback control was misguided.

Taken as a whole, the collection of biases revealed in students' thinking suggests important misconceptions and misunderstandings about what are effective ways to build reliable real-world systems. The next section builds on this hypothesis, presenting a case study of a student team that tried unsuccessfully to exploit feedback by embedding it in an

open-loop strategy, in a fashion suggested by the students fixated on their robots' ability to drive in a straight line.

## 4.2 Case Study One: Monitoring Robot Position

In the prior section, I examined ways that students negotiated issues of control in a broad sense, including the higher levels of reactive and algorithmic control and the lower levels of negative feedback and open-loop control. In this section, I present an in-depth case study of one particular team of students. These students brought strong personal convictions about what type of control system would be best for operating their robot. Even after their approach proved unsuccessful, they remained quite attached to it—an indication of the strength of their beliefs.

The students, who I will call “Stan” and “Dave,” participated in robot-building teams for two consecutive years of the contest. While all members of the team collaborated on the implementation of each robot, Stan and Dave’s strategic ideas dominated the designs. (Their team had other members as well during each of the two years.)

Stan was a master LEGO Technic builder, having been a avid fan and collector of the materials since his childhood and through his teenage years. His ability to design with the materials was quite impressive in both artistic and functional senses. He also was driven to create spectacular and elaborate designs that would have great audience appeal.

Dave, on the other hand, was an accomplished computer programmer, who reveled in building sophisticated and elegant computer programs. Both Stan and Dave were pleased to defer to each other’s strengths during implementation of their ideas, but collaborated during the conceptual design stages.

In *Robo-Pong*, Stan and Dave’s team chose to build a shooter robot. The task of building an effective shooting mechanism was a genuine challenge even for the experienced LEGO designer; there were a number of teams who gave up on building shooting robots after being unsuccessful in developing working firing mechanisms. Stan was undaunted and built a firing mechanism that was far better than any of the others, shooting the balls further and recocking quicker.

But the ball firing mechanism is only half of a shooter robot; the other half is a mechanism to feed the balls into the barrel of the shooter. For this Stan ended up adopting a mechanism that was suggested to him by another team developing a shooter robot (he obtained their permission before developing his own version of their idea). The result was a robot that was quite competent at scooping up and firing balls—or so it seemed, when they operated the robot under direct human control.

When Stan and Dave approached the control problem, they made the assessment that the feedback control methods we were proposing for overall robot guidance were too error-prone and ambiguous. According to them, robots that used local feedback from the environment were inclined to wander and make missteps on their way to the feedback goal. Stan and Dave didn't trust local feedback as the most effective or reliable way to accomplish the task.

Instead, their approach was based on two unusual sensor applications. The first was a method to ascertain the robot's initial orientation on the playing table (*Robo-Pong* used a randomized angle of initial robot orientation). The second was a mechanism that was intended to keep track of the robot's position on the playing table in terms of a displacement from its initial position. Thus, by knowing the initial orientation and by recording all changes from it, they expected their robot to be able to navigate about the table at will. Surely, they believed, this would be a more effective approach than relying on erratic data sampled as the robot wandered about the playing surface.

It is worthwhile to examine these two sensor application strategies in detail, as both required innovation on their part and reveal the depth of their commitment to this approach.

The primary sensor provided in the *Robo-Pong* robot-building kit for detecting inclination was the mercury switch. However, the kit included an alternate inclination sensor, which we had not tested but could potentially operate more effectively than the mercury switches. The device was a small metal can, larger than a pencil eraser but smaller than a thimble, which contained a tiny metal ball. The can's base was flat and four wires protruded down from it. Inside the can, the four wires terminated in distinct contact areas. In operation, the ball rolled around inside the can and created an electrical contact between the inner wall of the can and one (or two) of the four contact areas. In a sense, the device

was a two-dimensional version of the rolling ball sensor (see Section 3.3), but without the discrete sensing problem.

Our reason for not relying on the metal can sensor as the primary inclination-sensing device was that we were unsure of how pronounced the contact bounce problem due to vibration would be—the difficulty that confounded participants in *King of the Mountain*. Because the sensors were quite cheap, we decided to provide them anyway, buying enough to give two to each *Robo-Pong* team in addition to four mercury switches. We left the testing and evaluation of the device to the participants.

It turned out that the bounce problem with the metal can was indeed troublesome compared to the mercury sensors', so most participants used the mercury switches. Stan and Dave, however, thought of an unusual use for the metal can sensors. Because the robots were to start the contest on the inclined surface, robots could use inclination sensors to determine their initial angle of approach up the slant of the incline. The bounce problem would be eliminated since the robot would be at rest at the start of the round.

Because the initial orientation was specified in thirty degree increments, there were in principle twelve discrete initial orientations. Stan and Dave realized that by using three metal can sensors (each of which gave an inclination reading accurate to one of four quadrants), rotated by thirty degrees with respect to one another, his robot could *precisely determine* its initial orientation. This precision would be necessary to combine with the other component of their strategy, a movement controller.

The movement controller was a method to enable the robot to keep precise track of its position as it navigated across the playing surface (finding its way from the starting position to the trough, for example). For this, Stan and Dave employed *shaft encoders*, a sensor technology for measuring the angular rotation of a shaft.

Shaft encoders are typically employed in mechanical devices in which angular travel or displacement is confined within limits and hence can be repeatably measured. For example, a shaft encoder on a joint of a revolute robot arm can measure the angle of the arm. Usually such encoders are used in conjunction with limit switches; to calibrate the joint for sensing, the controller will drive the joint until it triggers the limit switch, thereby determining the zero position of the joint. All subsequent readings are then made with respect to the zero

position, which can be reliably re-established at a later time.

For mobile applications, shaft encoders have questionable value in determining position, because of the problem of a vehicle's wheels slipping with respect to the surface on which it is moving, causing errors in estimations of where the vehicle has moved on the surface. In these applications, shaft encoders are best suited for velocity and approximate distance measures; the speedometer/odometer of an automobile is an example of this usage.

Stan and Dave were cognizant of the slippage problem, but believed they had a workaround solution that would yield acceptable results. Rather than attaching the shaft encoders to the drive wheels, which they did realize would be quite prone to slippage, they added free-spinning *trailer wheels* that would rotate when dragged along the driving surface by the robot's movement from the drive mechanism. They attached the shaft encoders to the trailer wheels, which presumably were much less prone to slippage than the drive wheels.

Stan described the plan to me during early stages of his implementation. I argued with him that the approach was dubious; mobile robotics researchers had built various robots that attempted to perform global positioning based on similar ideas, and they all had shortcomings that forced the robots to frequently recalibrate their estimates of position based on local environmental references. He was undeterred. Since I did not perceive my role as preventing him from exploring an idea that he was interested in, and there was no harm in his trying it, I backed off.

Stan and Dave proceeded with the concept; Dave performed much of the programming. At various points during the development of the robot they were able to make remarkable demonstrations of the robot's ability to respond to control commands and measure and correct for its motion. The robot could drive to specific commanded positions; or, if the robot were dragged along a table to a position eight or ten inches away and then released, the position controller would drive the robot back to within an inch of its original position. Of course this only worked when the robot was dragged with considerable downward pressure so that the trailer wheels tracked the movement (i.e., they did not slip). I tried to argue with Stan that as impressive as this demonstration was, the approach was still futile because displacements in game play would be of the sort that *would* cause slippage. I think at this point Stan was so pleased with his apparent success that he was not listening to me (I don't

believe I would have listened to such advice either, had I implemented a system so effective in laboratory testing).

The robot's overall strategy was to drive down into the bottom of its side of the table and sweep back and forth along the trough, firing balls over to the opponent's side. If the robot's initial orientation were such that it was aimed upward, it would drive forward to knock one of the three top balls over the edge, as an "insurance" measure, before retreating into its main activity in the trough.

During the testing, Stan and Dave maintained confidence in their design. Indeed, the robot seemed to largely work; often it would only require minor hand-administered nudges during practice rounds to be successful. In the actual contest, however, the design failed badly. The slightest deviation from ideal circumstances caused the robot to fail without possibility for recovery. In ignoring feedback from the specific features of the playing field, the robot really had no hope if any number of things were to go wrong: a slight miscalculation as to the amount of rotation needed to orient properly, a bump from the opponent robot, or a even an irregularity in the surface of the playing field. Stan and Dave had fooled themselves during the robot testing—when the safety net of their prodding corrections was gone, the robot could not reliably perform the sequence of steps needed to work, and it was a competitive failure.

Stan and Dave teamed up again to design a robot for the *Robo-Cup* contest. In his first written report for that project, Stan reflected on his experiences in building the *Robo-Pong* robot:

**Lessons Learned.** Last year we spent a lot of time and resources on non-essentials. Our mechanical design was not stabilized until the night before the second contest. The majority of the time spent on software was in building a sophisticated object-oriented system. Our strategy was stable early enough, but it relied too heavily on unproven sensors and the accuracy of shaft encoders. This year we are focusing our energy on creating a robot that works well, even if we don't have time to make it look good.

**Design Goals.** In the past, robots have often failed because one particular sensor is critical and the software doesn't handle exceptions and breakdowns very well. Our goal is to build a robot that can check for errors and compensate before the errors escalate. Reliability and robustness are the targets for our project. Given enough time for testing, most unexpected conditions can be

accounted for.

It seemed as if Stan had learned his lesson. Yet, remarkably, Stan and Dave proceeded to repeat exactly the same mistakes in their *Robo-Cup* robot design.

While Dave did not attempt to build as sophisticated a global positioning system, Stan and Dave did again employ shaft encoders as the primary sensor for their *Robo-Cup* design. In *Robo-Cup*, the contest task involved driving from the starting position to a ball dispenser, actuating the ball dispenser's control button (causing balls to drop from above), and delivering the balls to a soccer-like goal. Not surprisingly, Stan and Dave chose to shoot the balls in, reducing their robot's task to the job of successfully moving from the starting position to a location beneath the ball dispenser at which balls could be shot directly into the goal area.

Stan and Dave's insistence on using the shaft encoders to guide their robot's movement from the starting position to the ball dispenser caused them to ignore an obvious alternative solution: a reflectivity coding on the playing surface which would have given the robot the information it needed (i.e., where the ball dispenser is located).

In this year, one of their troubles was gone: rather than robots being subject to a random orientation at the start of the contest, they could be placed within the designated starting circle at any orientation by the robot's own designers. Knowing that an exact measure of initial position was critical to his robot's success, Stan requested permission to allow the use of a template that he could employ to position his robot repeatably at the same location! We consented.

Their robot's performance in the actual contest was disappointing. Stan and Dave were not able to tune the performance of the shaft encoders to be consistent. The night of the contest, the robot would stop its journey to the ball dispenser just about an inch short of the desired position, lock itself to the wall, and proceed to fire balls that would miss their target for the rest of the round. The location at which it would anchor itself was just at the threshold of being able to trigger the ball dispenser at all; in one round it was not even able to do so, and spent the bulk of the round in futile attempts to strike the panel that would dispense the balls.



There is a short coda to this story. Beginning with the *Robo-Puck* contest, we ran a rematch competition at the Boston Computer Museum each year, which took place several months after the date of the MIT contest. Stan rebuilt his *Robo-Cup* robot from the ground up for the rematch, this time using a strategy based on feedback from the playing field features rather than from shaft encoders. His efforts were rewarded: though Stan's robot didn't win the overall competition, it did win one round—Stan's first competitive victory in two years of robot-building.

### **4.3 Case Study Two: Simulation as a Development Tool**

The first year that the Interactive C programming environment was available (the *Robo-Pong* class), there was a student who didn't want to use it. This student, "Tom," felt strongly that he would have a better learning experience if he programmed his robot in assembly language, thereby learning the "nuts and bolts" of the project hardware and the 6811 microprocessor, rather than being abstracted away from these things by the C language system.

Consistent with our approach of letting students delve into the aspects of the work that they individually found most interesting, we supported Tom in his endeavor. We did warn him that we would not have a great deal of time available to help him resolve various problems that he might encounter; this was agreeable to him as he specifically wanted to work through these sort of things himself. Tom was an accomplished MIT student in his junior year who had excelled in MIT's microprocessor architecture course and we felt comfortable that the arrangement would be productive.

Tom took a surprising approach to the project, however: he began by writing a program to simulate the operation of the 6811 microprocessor. His prior experience in the microprocessor architecture class had provided him with a powerful experience of the role of simulators. In the design optimization contest for the microprocessor course, Tom spent the bulk of his effort developing a compiler optimized for the microprocessor hardware,

rather than upgrading the hardware itself.<sup>2</sup> To assist in the development and testing of the compiler, he had created a simulator for the microprocessor hardware. This allowed him to test his compiler optimizations much more effectively than if he had to run its output on the target hardware directly.

Tom believed the same strategy would pay off in the Robot Design project. By creating a simulator for the robotic hardware, including the microprocessor, the robotic sensors, and the motors, he could develop much better final robot; he would not have to “muck around” with the robot hardware until he had a program that he had already tested to be effective on the simulator.

However, things did not go according to the plan. The job of creating a viable simulator for the robot project was more difficult than Tom expected. Aside from the issue of debugging the simulator itself—a simulator can do more harm than good if its own results are not trustworthy—there were aspects of developing the simulator that introduced unanticipated complexity.

One problem was related to complexity in the 6811 microprocessor itself. While Tom’s simulator succeeded in being able to execute the basic stream of 6811 instructions, there were other aspects of the 6811 hardware that were more difficult to simulate (primarily, the interrupt facility). Unfortunately, these aspects could not be ignored; they were an intrinsic part of the operation of the electronic hardware attached to the 6811.

The other problem was more subtle and yet less tractable: the difficulty of simulating the microprocessor *in the context* of the actual robotic task. Here Tom was caught off-guard: the simulator for his prior microprocessor design project was relatively straightforward, because the entire system to be simulated lay in the realm of the microprocessor itself. The robotic system, however, was based on a microprocessor interfaced to physical sensors, motors, and mechanics. In order for the simulator to be a valuable tool, it would have to do the job of simulating input from the sensors and the effects of output from the motors—which meant that both the environment outside of the robot and the robot itself would have to be built into the simulation!

---

<sup>2</sup>This was an unusual approach; the format of the course encouraged students to improve the hardware to achieve better performance.

It so happens that this very approach is taken by some robotics researchers. Rather than dealing with the vagaries and expenses of developing physical robotic hardware, some researchers build complex robot simulations that they use to then study control algorithms, vision systems, path planners, and other aspects of robotics work.

Other robotic researchers are dubious or even scornful of the validity of this approach. To the “hands-on” roboticists, the value of any theory, model, or architecture of a robot control system is only as good as it can be tested on an actual robot with actual sensors, motors and computational hardware. By using simulators, this group claims, perplexing and deep questions are glossed over and the results obtained do not hold much validity. They point to studies developed on simulated experiments that break down when tested on actual hardware.

This tension between simulation versus reality is an on-going debate in the robotics field. Many in the simulation camp now include models of sensor noise and other physical problems—developed with actual experimentation—in their simulated environments. In other recent work, researchers test their control strategies on both physical robots and simulations, thereby gaining the virtues both: the irrefutability of an actual system and the versatility of a simulated one. An example is the work of Meeden et al. (Meeden, McGraw, & Blank, 1993).

In Tom’s case, the results were disastrous from a performance point of view. By the time he realized the simulator was not going to be useful, much of the month had expired. When he finally delved into the actual programming on his robot, he progressively encountered a series of low-level hardware and software problems—ones which we had successfully shielded other students from in developing the Interactive C environment.

With literally a day to go, Tom gave up on the assembly language approach and attempted to get something working using Interactive C. It was too late, though; while he did field a robot on the eve of the contest, he had insufficient time to work through the higher-level control problems to get his robot to perform the task.

Even though the robot was a failure, this is not to say that Tom did not have a valuable learning experience. Certainly he learned in intimate detail a great deal about the 6811 microprocessor and the hardware we had developed around it, but there was a more profound

engineering lesson as well. I would guess that Tom would think twice about the complexity of a real-world system before pegging his hopes on a simulator again.

## 4.4 Analysis

The examples in this chapter illustrate that, rather than being “blank slates,” students who participate in the Robot Design course have a variety of preconceptions about systems and control. These ideas are formed by experiences in the traditional academic curriculum, and warrant examination specifically because they are not particularly effective when applied to the Robot Design task.

In the first section of this chapter, *Introduction to Robotic Control*, we saw how many students have trouble seeing the contest task from a robot-centric perspective, and instead imagine the task from their own omniscient perspective. This naiveté was tempered when they tried to put their ideas into being and realized that the robot’s point of view was very different from their own.

With few exceptions, all of the robots created for *Robo-Pong* and *Robo-Cup* used fundamentally algorithmic solutions. This fact is not surprising; there are at least two factors which would strongly encourage the formulation of algorithmic solutions.

One of these is the nature of the contests themselves, which require a series of activities to take place in a short period of time. A robot that wandered about as it waited to be guided into action by sensory stimuli would not be efficient; the typical winning robot employs a fast, reliable, and effective algorithmic strategy. If they’re not interfered with, these robots successfully perform their task on nearly every run.

The other factor encouraging algorithmic solutions is the Interactive C programming language, which is fundamentally procedural. While it is a general-purpose programming language, and can in principle support the construction of a variety of control methodologies, as a procedural language it encourages students to create procedural control structures.

One feature of Interactive C, however, encourages reactive control: the multi-tasking capability. Indeed, this feature was used by students to implement reactive controls, as was discussed in the *Crazy Train* robot design. This evidence points to the effect that

the programming language has on students control ideas; extensions or revisions of the programming language could encourage students to think differently about control.

While it is then not unexpected that students create largely algorithmic robots, what *is* surprising is how poorly these systems perform with respect to the students' own expectations. Nearly all of the students who participate in the class are genuinely surprised by the difficulty of getting their robot to work reliably. They blame performance problems on failures of particular components of their systems, rather than re-evaluating overall approach to control.

The students' control ideas come from those presented in their university courses. At MIT, examples are found in the introductory Computer Science course, *Structure and Interpretation of Computer Programs* (course number 6.001), and the *Software Engineering Laboratory* course (course number 6.170). Among the central ideas developed in 6.001 is the concept of *abstraction*: that by encapsulating messy implementation detail into conceptual “black boxes,” complex systems can be built from components with relatively clear functionality. Abstraction is a way of managing complexity—a way to keep minutia in check, and to create large systems with clearly understood parts. As stated in the text of the course (Abelson & Sussman, 1985):

In our study of program design, we have seen that expert programmers control the complexity of their designs by using the same general techniques used by designers of all complex systems. They combine primitive elements to form compound objects, they abstract compound objects to form higher-level building blocks, and they preserve modularity by adopting appropriate large-scale views of system structure.<sup>3</sup>

The Software Engineering course expands and fleshes out this principle, teaching programming techniques like procedural and data abstraction through extended programming projects. The course makes a point of teaching how to write programs in a modular fashion, test code modules independently, and then integrate them into a complete system. Typical final projects are the design and implementation of a text editor or a computerized Othello game.

---

<sup>3</sup>*Structure and Interpretation of Computer Programs*, page 293.

While these projects are ideal for the goals of the course—teaching abstraction and modularity in the architecture of large computer programs—they promulgate the mindset that large systems are made from parts that are completely understandable and formally specifiable. In both of these examples, as in many others, the computer program consists of precisely formalizable data structures and algorithms to operate upon this known data to generate known results. Indeed, a large portion of the Software Engineering course is dedicated to methodology of testing with the intent of proving that a given program module will yield correct results when presented with data that satisfies particular conditions.

Much of the engineering curriculum is based on modern system theory: mathematical methods for understanding algorithms and systems that take particular inputs and yield particular outputs. Dynamical systems analysis, in which system state is represented by a state vector, and signal-processing theory based on transfer functions are leading examples. These domains of theory deal with systems that are perfectly represented in the mathematical models that students learn to manipulate.

Of course the value of such engineering knowledge is precisely that it allows us to construct systems that are indeed controlled to extreme degrees of precision. The great engineering successes are based on our ability to understand, model, analyze, and precisely control the world around us. But not all large systems are controllable by these means. As discussed by Ferguson, recent work on chaotic systems has challenged the assumptions of those working on automated traffic control systems:

For engineers, a central discovery in the formal study of chaos is that a tiny change in the initial conditions of a dynamic system can result in a major unexpected departure from the calculated final conditions. It was long believed that a highly complex system, such as all automobile traffic in the United States, is in principle fully predictable and thus controllable. “Chaos” has proved this belief wrong. The idea that roads will be safe only when all cars are guided automatically by a control system is a typical but dangerous conceit of engineers who believe that full control of the physical world is possible.<sup>4</sup>

The example of the third section, *Simulation as a Development Tool*, raised the issue of the role of simulation in understanding and representing complex systems. For some

---

<sup>4</sup>*Engineering and the Mind's Eye*, page 172.

domains (e.g., VLSI design) simulation is a powerful and accurate technique for developing and testing complex systems. For others (e.g., structural engineering), there are difficult issues of determining the applicability of a simulated model to the actual artifact which affects the reliability of the simulation in profound ways. An additional complicating factor is that many simulation packages are developed by theoretical rather than practicing engineers, raising the question that the practicing engineer must trust the output of the simulation without necessarily knowing the founding assumptions upon which it is based. Henry Petroski quotes a Canadian structural engineer on this point:

Because structural analysis and detailing programs are complex, the profession as a whole will use programs written by a few. These few will come from the ranks of the structural “analysts” . . . and not from the structural “designers.” Generally speaking, it is difficult to envision a mechanism for ensuring that the products of such a person will display the experience and intuition of a competent designer.<sup>5</sup>

The reliance on simulation as an engineering technique is a consequence of assumption that complete control of the physical world can be attained. While simulations can reveal problems before they develop into serious engineering failures, over-reliance on the trustworthiness of simulations can lead to disasters. Contemporary examples of failures due to inadequacies in simulation abound; here are two striking ones from Peter Neumann’s column in the *Communications of the Association for Computing Machinery* periodical (Neumann, 1993):

On April 1, 1991, a Titan 4 upgraded rocket booster (SRB) blew up on the test-stand at Edwards Air Force Base. The program director noted that extensive 3-D computer simulations of the motor’s firing dynamics did not reveal subtle factors that apparently contributed to failure. He added that full-scale testing was essential precisely because computer analyses cannot accurately predict all nuances of the rocket motor dynamics. (See *Aviation Week*, May 27, 1991 and Henry Spencer in *SEN 16*, 4, Oct. 1991.)

The collapse of the Hartford Civic Center Coliseum 2.4-acre roof under heavy ice and snow on January 18, 1978 apparently resulted from the wrong model being selected for beam connection in the simulation program. *After* the collapse, the program was rerun with the correct model—and the results were

---

<sup>5</sup>To *Engineer is Human*, page 201.

precisely what occurred. (Noted by Richard S. D'Ippolito in *SEN 11*, 5, Oct. 1986.)

If engineering students gain early hands-on experience with the complexity of electrical, mechanical, structural, and software systems, they will have more respect for the sorts of pitfalls that can be expected later when working on real projects, like in the case of the Civic Center cited above. With project experience like that in the Robot Design course, students learn some of the limitations of traditional control and analysis, and gain some of the experience that is needed to be a realistically-minded engineer.



# Chapter 5

## Design Styles

This chapter discusses the design activity engaged in by the students of the Robot Design project. These students have a diverse set of backgrounds; each year, there is a distribution of students from the four undergraduate years as well as a few graduate students, with a corresponding difference in academic experience at each level. For some, the Robot Design project is their first experience in creating a complete technological artifact with their own hands and minds, while others can call on past experience gained through previous academic work, industry work, or personal pursuits.

We operated the project based on an “affirmative action” policy that was meant to encourage an alternative, bottom-up style of design work. Our motive was to allow students to learn about unfamiliar ideas, materials, and methods by a playful, exploratory process rather than the traditional methods of teaching through lectures, texts, and problem sets. Some aspects of our method was discussed in Chapter 3, which described how we created the technological materials to support exploratory work. This included contest designs that encouraged creative thinking and a variety of solution strategies, and robot-building hardware that was highly interactive to encourage experimentation as way of learning.

This chapter presents the results of this agenda. In the first section, I discuss students’ actual working styles as they created their robots. Most of the students used an exploratory process to learn how to work with the LEGO materials, but many found this method unfamiliar and uncomfortable. Most students reverted to the traditional top-down style when they reached the programming phase of their project work; some students referred to

this method as being the “computer science mind.” In general, even though the top-down style did not produce effective results, students had trouble abandoning it.

The remaining sections of the chapter focus on the adaptability of the learning environment to students’ own interests and desires. The second section discusses a variety of design excursions taken by students, reflecting the openness of the technology we provided to them as well as our own encouragement for them to make such explorations. The third section discusses ways that students adapted the ostensible goal of the project—that of building a successful robot—to suit their personal interests.

## 5.1 Affirmative Action for Bottom-Up Design

As creators of the course and its materials, we were participants in a bottom-up design process ourselves. We therefore had a deep sense that this sort of design was both effective and valuable. First-hand, we had learned about the properties and capabilities of the robotic systems as we created them, and as users of our own technology, we saw natural ways to extend it—making it more powerful, versatile, and easy to use, and more suitable as a substrate for the design course itself.

Rather than having students design their robots by “thinking hard,” we wanted them to (for example) make a series of prototypes as part of the process. To learn about sensors, we wanted them to experiment with the devices we gave them. To learn about robot control, we wanted them to write programs for their robot and see the results. As was discussed in Chapter 3, we designed our robot-building kit to encourage such a playful exploration of ideas.

Thus, we wished to not only *allow* bottom-up design, but to *encourage* it. I would liken the effort to a sort of affirmative action for intellectual styles: top-down design is so heavily favored in academic circles that one must consider anything other than that style to be a minority deserving of special consideration and opportunity.

### 5.1.1 Experimenting with the LEGO Technic System

When learning to use the LEGO *Technic* system, students typically used a bottom-up style of exploring the materials and prototyping their ideas, as described by this student:

In the “strategy” stage, I have played around with the LEGO (I had never seen Technic before) in the hope of getting a “feel” for LEGO, which I believe I will need before insight into a good structure for the robot is possible.

The most defining characteristic of this phase of the project—the LEGO design—was the aspect of re-design: a repetitive, iterative, trial-and-error process involving conceptualization, implementation, testing, analysis and iteration. Often this process happened so rapidly that the steps flowed together, making it difficult to separate them from one another. Rather than attempting to analyze all of the possibilities and implications of a design idea, students would typically build something to see if the idea was feasible, and then either improve the artifact (through another design iteration) or discard it. As one student explained the development of his robot’s chassis:

My teammates helped a lot by pointing out weaknesses in my design and by forcing me to rebuild, but I had to do the building myself. It was discouraging to dismantle the old robots, but trial and error is certainly necessary. Here’s a quotable quote: *It’s astounding how much going backward is entailed in going forward.* I just hope I’m through going backwards with the chassis! (Emphasis added.)

The LEGO materials are ideally suited for this style of work. Since they are disassembled as easily as they are put together, students know that they aren’t committed to a particular mechanism or concept simply because they have build it. To make sure that this aspect of the LEGO system was preserved, we didn’t allow students to cut or glue the LEGO parts for particular applications; they had to use the LEGO materials as they were designed. (There were a few minor exceptions to this rule, but the exceptions were designed to encourage the creation of modular, reusable parts, like easily mounted sensor assemblies.) This stands in contrast to “raw” building materials, like those used in MIT’s *Introduction to Design* class.

We encouraged students to try things out as early as possible as a way to learn about the capabilities of the materials they had at their disposal, and as a way to try out their

own ideas. This was perhaps the dominant way of learning and way of designing that the students used. In the following quote, a student discusses how both his vehicle and his strategy changed as a result of explorations with the LEGO materials:

Since the first report the vehicle has changed extensively in both structure and purpose. These changes are both the result of hands on LEGO bricks interaction and conceptual changes in strategy itself. As we proceeded with the construction of the robot we realized the difficulties [inherent] in transferring ideas into working LEGO kit contraptions.

Many students developed their mechanical ideas in this fashion. A common experience was that of discovering that a favorite idea was easier to conceive than to implement. The following two quotations reflect a typical process by which ideas were discarded:

Playing and tinkering have had an interesting interplay. I have been the team member who has come up with complex schemes for our robot and has consistently failed to get very far in prototyping such systems with LEGO even after hours of work.

We spent a lot of time mainly revising our strategy and playing with LEGOs. Our initial ideas of arms and shooters that span the entire board were not “LEGOizable.”

It is typical for students to learn about their own capabilities and the properties of the materials by developing a series of prototypes. Here a student explains his team’s process with regard to developing their robot’s chassis, which is characterized by an iterative learning process:

Towards the beginning of the week, we built a small vehicle out of LEGO to get a feel of how gear trains work and how LEGO fits together in a strong configuration. We paid too little attention to perfect vertical spacing, and ended up having diagonal braces holding together pieces awkwardly. We used too much LEGO for just a frame, had no place to put the battery or board, and had an inefficient gearbox. We dismantled this machine and built a second one that was a little more efficient in LEGO use, and turned a little better as well. Finally, we took this apart and thought out another one. When we built this one, we were careful to reinforce all structures with vertical LEGO beams, use perfect spacing, and make it the size we wanted. We also discovered that it is extremely important to use many structural vertical beams in the gearboxes so that the axles don’t bend and create friction.

Here is another student's comments which describe a similar learning process:

So far things have been built rather than discussed and virtues debated. We will agree that we want a shooter and so someone will start it. Then someone else will pick it up, play with it, make a few enhancements, and pass it on to the next experimenter. I built the first release mechanism but it was not very sturdy. John came in and quickly made it into a working system.

The original pull-back mechanism came from an idea that I saw one of the alumni working on, but was largely changed. This is how we have been working. No one was really totally responsible for the shooting system since it came from many seed ideas that were worked out after much experimentation by other members of the team.

While these students felt comfortable with the iterative design process he and his teammates used, others who worked in essentially the same manner were less pleased with the way the time was spent:

We basically decided to build components since we really did not know how to decide between strategies. We knew we needed a drive mechanism, a steering system, some way to score, and a unit for pushing the ball release button. Of course, different strategies would require changing these systems but once built it is easy to modify. *The result is probably not as efficient as a system properly designed in the first place.* (Emphasis added.)

This student's comment points directly at the tension that many students experienced. They adopted bottom-up design strategies since the ones they were used to didn't seem to apply. Even though the method they used was effective, they were left with a nagging sense that they weren't doing design the right way.

Some resolved this inner conflict by shrugging off the exploratory work as insignificant play, a prelude to the "serious" work to follow:

Saturday, 11th January: Our team played with LEGOs all day. I designed free rotating wheel, the chassis and [an] automatic transmission box. Tom was working on a catapult mechanism and Aram was making some strange high friction apparatus. This was not meant to be serious work, just trying to get some feeling and experience working with LEGOs, and see how much material is actually available, and how complex we can get.

What is most interesting about these comments is that while the students are indeed using bottom-up design practices, they themselves do not validate this process as being

instrumental to their own learning or to the successful completion of their design. Most often, students almost apologize for their excursions away from the task at hand.

Still, the Robot Design class did not *impose* a particular design methodology. Students were free to organize their own work as they like. As the following passage from a student's journal illustrates, there were differences of opinion about what methods are best, even within a design team:

Tuesday, 7 January 1992: We attended lecture, received our kits, and went home to start building the 'bot. It was clear from the beginning that there was a distinctive clash in approach between myself and my teammates. I favored a top-down approach, immediately wanting to devise a strategy. My teammates Dave and Tom, who favored a bottom-up approach, were more interested in building pieces of the robot, nicknamed "Robbie" by Dave.

... I also came up with two rudimentary ideas for robot strategies, believing that we couldn't begin the structural design of the robot until a strategy had been decided ...

Thursday, 9 January 1992: After the lab closed, we went to my room and played with LEGOs. I immediately wanted to set a strategy in order to create a proper design, but Dave and Tom just wanted to play with the pieces.

Saturday, 11 January 1992: Dave had tried to design an automatic transmission, and on this day he tried to design a "chassis" to connect to it ... Tom worked on a ball projectile device. I also continued clamoring that we should decide on a strategy.

Sunday, 12 January 1992: Today, we finally agreed to discuss basic strategy. Things probably worked out for the best this way because we have learned much about using LEGOs in the past few days. We decided that it would be best to implement a rather simple "get the ball, put it in the goal, and go to the dispenser where the opponent isn't" strategy.

This student concluded that the "play first, decide later" strategy was effective, but not all students drew this conclusion from the same experience.

Another student explains how he had to succumb to his "juvenile instincts" in order to suppress his "computer science mind":

My teammate and I were clueless when it came to LEGO building; we spent several afternoons and nights just staring and letting our juvenile instincts take over. . . we tried creating gear trains and looking for ways to attach LEGO bricks securely. If nothing else, one important quality we will gain is the ability to think mechanically instead of tackling problems with a computer-science oriented mind.

To rationalize the process he used to learn to build with LEGO parts, this student calls it an “ability to think mechanically”; he does not want to confuse that kind of thinking with the orderly, divide-and-conquer strategies he is used to—the type of thinking he associates with computer science.

Other students were more accepting of the bottom-up approach they used, though it was still unfamiliar to them:

Lots of time was spent on playing with LEGO building blocks. I found out that LEGO’s building blocks were more than a toy. I built gear blocks with serious gear reduction and found lots of interesting LEGO techniques. As I spent more time on playing with LEGO’s, interesting ideas popped into my head and I was amazed to find that I could put those ideas into reality.

While this student expressed surprise in her hands-on design process, others took it in stride. In the following excerpt, a student describes how a key design feature was happened upon by accident:

I left off in the last report where Willy and I had decided to mount the servo motor on the front wheel of our tricycle to provide a steering mechanism. Using a chain to connect our servo on the main chassis to the front wheel, I managed to get it working while Willy was building gear reductions for the two rear wheels. We weren’t crazy about the way it worked but it was a success, at least until the [wire to the servo motor] broke. I didn’t feel like resoldering it that late at night, so I just set the front wheel straight forward and helped Willy finish the drive part of our chassis.

We tried out the two motors and we liked the way they worked. It appears that we have decent torque as well as considerable speed. More importantly, when I went to turn off the motors, I accidentally threw one into reverse. The result was more efficient turning than we ever saw with the servo. Therefore, the servo motor is now history and we’re steering just by spinning the rear wheels in opposite directions, like a wheelchair.

This student summarized his team’s process succinctly:

Some of our mechanisms were the result of careful planning, but most have just come about from playing with the pieces and seeing what we can do.

This synopsis could be applied to many of the students’ LEGO-building work.

## 5.1.2 Programming Strategies

When the Interactive C programming environment (discussed in Section 3.2) became available, students had two different sorts of reactions to the user-friendly, rapid-prototyping development environment. Some, as we anticipated, used the system to create playful robot demonstrations that had little to do with the contest problem per se. For these students, making a simple demonstration was the best way to learn about the potential that a robot might have—knowledge they would use to guide their design process when they went about tackling the contest task.

Many students, however, explored the programming environment just enough to know how it worked, and then assumed that they already knew enough about programming that they didn't need to explore *robotic* programming. These students expressed simple surprise that the top-down methods didn't work when they actually tried to get their contest robot design to perform its task. It seemed that with regard to the mechanical aspect of design, students realized they needed to play with the materials (i.e., LEGO parts) to express their ideas, but that with regard to the programming and control side, they expected that they could construct solutions just by thinking about what should work:

I spent most of the latter part of the week working on my wall-following procedures with limited success. *I only recently realized the need for using the robot in conjunction with programming, for often what you think will happen does not.* My first wall-following attempt looked good on the screen, but when I ran it, it looked [terrible]. (Emphasis added.)

This student's experience was typical. Many students spent time constructing elaborate control programs before they had spent time trying to get simple behaviors to run on their robots. They then attempted to "fill in" the sub-procedures of their master control program, as this student describes:

The coding is also progressing, though at a slower rate. Our approach to programming has been top down. We have a shell of our strategy in place. We need to implement many of the subroutines that the main control program will call. We are debating various turning strategies ranging from shaft encoders, optical sensors to determine our orientation [with respect to] the playing field, and using the back bumpers to align the machine perpendicular to the side wall.



This design style is the traditional top-down method favored by most computer scientists. In the task of robot programming, it failed badly, because, as was discussed in Chapter 4, most students' robot strategies were based on assumptions of robot control that were faulty. Students were particularly susceptible to this type of design problem because of their confidence in their understanding of the problems they were facing, their confidence in their programming ability, and their confidence in the problem-solving strategies they had used in the past.

### 5.1.3 Time Constraints

While the existence of a particular mechanism or piece of code did not per se imply a commitment to its use, many students were confronted with the situation of time running out with less-than-ideal system components in place. As described by the student in the following quote, often a "prototype" became the final design:

I don't know how many prototypes (for our LEGO design) we've made, but it's been a bundle for sure. We can't seem to make a robot that corresponds well with our strategy (which primarily deals with blocking the goal with our robot). We're looking for a VERY strong robot that won't fall apart if run into by the opponent's robot, hence eliminating chance of defeat. We also need speed (hence, a low gear ratio) to get to the ball dispenser and shoot quickly. Thus far, we've developed a few pretty decent models, however, we have been ambitious (desiring the near perfect model). This could hurt us now, as we are running out of time. We have a working model right now, and we'd like to improve it, but might have to stick with it due to time constraint.

The same phenomenon happened with code, often resulting in major simplifications of strategy and programming.

Each year a few students decided to perform major reconstructions of their design with just one or two days before the contest. Usually this forced students to realize how interdependent the different levels of their design had become. For example, one team redesigned the primary drivetrain for their robot the night before the contest. While the new geartrain was significantly better than the old one, their program no longer worked because it was peppered with hard-coded timing constants that turned out to have dependencies on the original geartrain. The team never recovered from their mechanical "upgrade."

This real-world engineering lesson of systems and deadlines was a surprise to many of the students.

## 5.2 Design Excursions

This section illustrates various ways in which students formulated their own investigations and little research projects based on ideas that came from working with our materials. These projects were facilitated by the technical openness of our system, as was described Chapter 3, and our own policy of encouraging such endeavors.

As part of tendency and freedom to “play around” with various ideas during the course of their projects, a number of students spent a fair bit of time on sub-projects that only sometimes had practical relevance to their main project. Some of these design excursions were a natural result of the bottom-up process that the students were using. A mechanism, chunk of code, or other artifact would get discarded when it no longer remained a necessary part of a revised strategy or functional plan. Sometimes a sub-project would be started to play with an idea or on a whim, without the expectation that it would necessarily become part of the final design. Through the normal course of this working style, artifacts that were once thought valuable would no longer be considered so, and vice-versa.

Other design excursions came from a more deliberate attempt to create something unusual. Some of these efforts were driven by functional considerations (e.g., a new type of sensor device that would play a key role in an overall strategic plan); others were simply aesthetic (e.g., additional hardware or software for creating musical sound effects).

During the *Robo-Pong* year, several teams embarked on a sub-project to develop motor driver electronics that would be more effective (i.e., would deliver more power) than the stock motor driver circuit. This phenomenon is interesting on a number of levels, because it illustrates not only the students’ desire and willingness to invest time in what might be considered a frivolous sub-project, but how their efforts fed back into the design of technology and policy for the class.

The remainder of this section discusses these design excursions to illustrate the variety of learning styles that were supported by these projects. We will begin with the motor

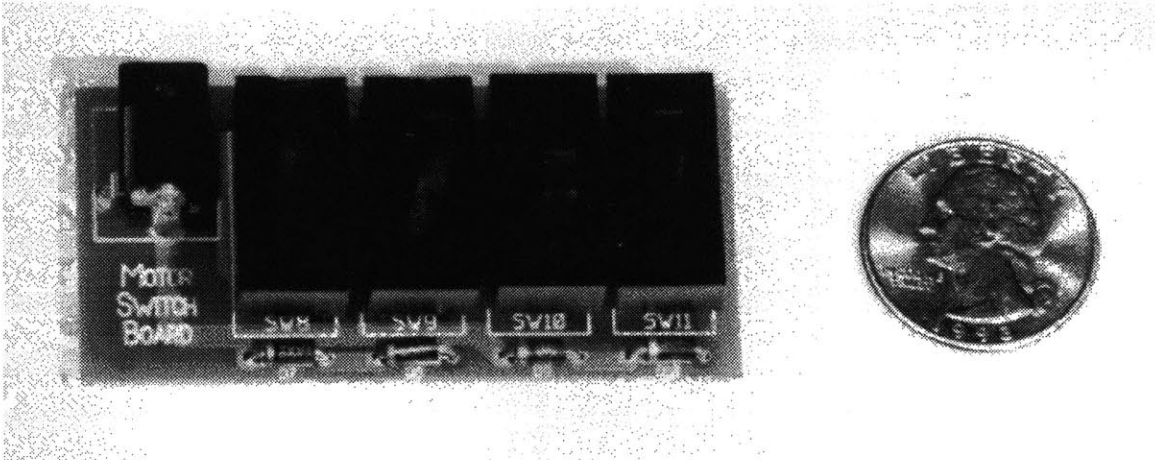


Figure 5-1: Motor Switch Board used for manual control of kit motors

driver circuits just mentioned.

### 5.2.1 Improving the Motor Drivers

In the first two years of the project, there was no straightforward way for students to test motors other than by driving them through the controller hardware. It became apparent that this significantly hampered students' ability to play in early stages of LEGO designs and thus develop a sense for the capability of their LEGO constructions. In order to test their vehicles, students would push their motor contacts up against the battery terminals to see how their devices would run, a clumsy and inelegant solution (though it worked and was helpful nonetheless).

We developed the *Motor Switch Board* as a remedy to this situation. Introduced in the *Robo-Pong* year, the switch board was a panel into which student would plug their motors; the panel was then plugged into a battery. Students could easily switch on and off up to four motors (see Figure 5-1).

There was one complication with the use of the switch board. The amount of power that the controller electronics delivered to the motors was significantly less than the amount of power delivered from the switch board. Motors operated by control electronics ran at about half the power than those operated by the switch board. Students were informed of this effect, and cautioned that they should not expect the same level of power when they

installed their controller boards. Nevertheless, a number of teams built mechanisms that performed adequately when powered from the switch board, but not from the electronics.

Upset when they discovered just how much less power was available from electronic control, approximately five student teams then decided to take advantage of the “\$10 Electronics Rule,” a rule we had established to encourage creative sub-projects. According to the \$10 rule, students could spend up to \$10 of their own money to purchase electronic parts for their own custom modifications.<sup>1</sup> In addition to restricting the monetary expenditures to \$10, we set a limit of ten additional components.

Rather than redesign their mechanics, these teams set out to build their own motor control electronics to replace the “poorly performing” stock electronics. Perhaps motivated by the motto, “Stronger is faster, and faster is better,” some of these students soon discovered that the latter half of this premise turned out to be an erroneous assumption.

In building motor driver circuits, two types of components are typically used: electro-mechanical switches (relays) and electronic switches (transistors). Some circuits use one device family, the other, or a combination of both. We allowed students to use either approach, so long as they adhered to the terms of the rule. We did not give them particular advice as to which approach was better.

As it happens, designing motor control circuitry is a non-obvious problem. There are a number of tricks that must be employed to ensure that the control circuitry is not damaged by current spikes that occur during normal motor operation. Most of the students who went about building their own circuits did not do the research required to uncover these tricks, and constructed hardware that caused significant trouble—which often did not show up until after they thought their circuit additions to be finished.

One student, who received help from an expert friend, constructed an exemplary circuit. He used more than ten components, however, having overlooked this constraint. It was upsetting to have to tell him that he could not use his circuit, particularly since it was an excellent solution to the perceived problem, but we had established the rule and had to stand by it in the interest of fairness.

---

<sup>1</sup>We had agreed upon the \$10 figure as an amount large enough to allow the purchase of interesting components but small enough to not favor teams with strong financial resources.

Other students encountered a variety of other difficulties. Most common was increased unreliability in the basic computer hardware as a result of poorly designed motor circuits. Unfortunately these could not be easily fixed without breaking the ten-component limitation.

One team who ended up using their motor circuit to the dire end encountered a different sort of complication. Their circuit delivered a great deal more power to the motors, as they desired, but it also made the actual power output of the motors much more dependent upon the battery voltage level (see discussion in Section 4.1.3). It seemed that the standard circuit tended to reduce differences in actual power as a function of battery level.

In order to gain reliable results, these students developed their control software with heavy dependencies on the assumption that their battery was fully charged. After just a few minutes of usage, the battery voltage would drop and their robot would no longer function properly. They didn't have time to properly generalize the program parameters as a function of battery level, so they were forced to constantly have batteries on charge and to charge batteries between rounds during the contest itself. In the end, it made for quite an unreliable design.

The battery discharge problem, and the change in motor performance that resulted from it, was a problem that was endemic to the hardware system we had created; all students had to deal with it. Depending on the design of the basic strategies for negotiating around the playing field, the problem would manifest itself with more or less conviction. The change in motor driver design, however, made the problem nearly intractable, as this group of students discovered.

After considering the experiences of the students who had attempted to construct their own motor circuitry, we were faced with a difficult decision. The attempts had caused a lot of frustration both on the part of the students who were trying to get buggy circuits to work, and on our part, those who were helping them. There were already enough hardware problems in getting all of the students up and running using the standard hardware, with too few people available to help. While these students had taken the admirable initiative to do something clever, and had a valuable learning experience, they were the cause of an untoward drain on the class resources.

The modifications were clearly unsuccessful from an overall performance standpoint.

A couple of teams had succeeded in fielding robots that were obviously more powerful than their competition, but the students were not capable of controlling that power and turning it into an advantage. We knew that we didn't want to be designing contests based on the premise that bigger is better, and we didn't see the need to encourage this attitude in students either.

In subsequent years, we decided to disallow modifications to the motor driver circuitry. To address the students' concerns about the shortcomings of our standard circuit—both perceived and real—we took a dual approach. First, we implemented a somewhat better standard circuit in subsequent years to alleviate the actual problem. Second, we installed current limiters in the hand-controlled motor switch board—deliberately “crippling” its performance so that students would not be deceived by the amount of power they would see in their early manual testing!

While these actions mitigated the electronic power problem as well as students' perception of it, I have mixed feelings about the decision. Had the human resources for running the class not been so constrained, it would have been better to not cut off that avenue of exploration for the set of interested students.

### **5.2.2 Developing New Sensors**

Other design excursion projects were less problematic from a practical or policy level, but still indicative of students' style of participation in the project. In the *Robo-Pong* year we distributed a number of Hall effect (magnetic) sensors in the students' kits. These sensors were unusual in that we did not have a specific purpose in mind for their use, nor did we provide a debugged circuit for employing them. One student, “Frank,” became quite intrigued with the Hall effect sensor technology, and set out to design an ultrasensitive Hall effect sensor—one that would be good enough to detect the Earth's magnetic field. With the use of this device, he hypothesized, his team's robot would be able to know its orientation with respect to the playing field at all times, which surely he could turn into an advantage in game play.

Frank experimented extensively with the Hall effect devices provided in the kit, determined that the provided ones were inadequate for his purpose, and then went on to research

better devices. Aware of the \$10 limit on additional electronic purchases, he requested that he be allowed to have prospective sensor devices donated to his project as engineering samples. We agreed to this suggestion; there was nothing preventing other teams from doing the same if they so chose.

Frank's work did not result in a deployable sensor device; when the engineering samples did arrive it was too late to develop circuits around them and use them in the robot design. Nevertheless, the sub-project was a valuable engineering experience for him.

### 5.2.3 Musical Robotics

In both the *Robo-Pong* and *Robo-Cup* years, musical hardware and software additions were a popular diversion. Students initially became excited about the possibilities of musical robots via experimentation with stock hardware and software provided—a small electronic beeper and software routines for playing simple tones of varying pitch and duration.

While some students focused their musical efforts on transcribing their favorite songs to play on the beeper (e.g., The Beatles' *Hey Jude*, Wagner's *Ride of the Valkries*, and MIT's *The Engineer's Drinking Song*), others modified the sound playing hardware to achieve greater volume or variety. The most ambitious project involved grafting an electronic keychain “zapper” (a pocket device that creates sound effects of laser blasts and other explosions) to the robot's microprocessor controller. The project involved electronically mixing the sound output from the zapper device and the microprocessor beeper circuit into an amplified speaker driver. Under software control, the circuit was able to simultaneously play tones while triggering zapper sounds. The *pièce de résistance* of the project was the musical selection that the students chose to play on their circuit: William Tell's *Overture of 1812*, complete with cannon sounds, courtesy of the keychain zapper.

For several students, the work on the musical projects, which were obviously of no utility in getting their robots to perform the contest task, was one of the few relaxing aspects of the whole project. Here is how one student describes working on the musical program:

I wrote a short program to make the beeper play a little tune (*Flight of the Valkyries*), and Eric wrote two programs to play *Let it Be* and *Hey Jude*. I found writing this program to be the most relaxing and not nerve-wracking part

of the contest.

These excursions were precipitated by the simple provision for sound output we had built into the stock kit hardware. If for no other reason than for providing this kind of emotional outlet, I consider the audio projects to have been extremely valuable.

## 5.2.4 Alternate Control Systems

In the first year of the hardware contest (*King of the Mountain*), one student, “Peter,” undertook a development effort that foreshadowed our work in subsequent years: that of creating on-board computational control (in the *King* contest, students’ robots were controlled by off-board, desktop computer computation).

Peter was motivated by an aesthetic sense that objected to the standardized hardware we had provided that year, which required a tethered cable to join the robot to its “brain.” Providing local control on the robot itself was something we would have liked to have given the students, and would provide in all following years. But in the *King* year, it was not available, and with good reason: we did not know of an inexpensive and sufficiently simple way to implement on-board computing at the time.

Peter knew of a way to implement a compromise solution. Rather than use a fully programmable microprocessor to control his robot, he proposed the use of a dedicated circuit with limited programmability, known to computer scientists as a *finite state machine*, or FSM. The hardware of an FSM consists of just a few simple integrated circuits—in fact, single-chip programmable devices to implement FSMs exist, and this is just what Peter proposed to use.

Peter succeeded in implementing a hill-climber using the FSM method, though his robot did not win the contest overall. His robot still needed the cable to provide power to operate the motors, so the aesthetics of the solution were not completely ideal, but he had the satisfaction of knowing that his was the only robot that year with true on-board control. He also did avoid any number of annoying interfacing problems that other students had to deal with.

In the contests since, a few students poured their efforts into developing alternative



approaches for controlling their robots; the case of the student determined to work in assembly language, described in Section 4.3, is one example. This student was motivated by a desire to be “close to the hardware”; others wanted to move in the opposite direction, toward more abstract models of control. In the *Robo-Cup* year, several students expressed interest in experimenting with robot control ideas developed by MIT’s Professor Rodney Brooks, who developed the *subsumption architecture*, a novel philosophy and practical approach for developing robust, adaptive control systems for mobile robots (Brooks, 1986; Connell, 1988). One student in particular spend most of his month’s time (to the chagrin of his teammates) in an attempt to get Brooks’ software to run on the course hardware. He did not get the system functional in time to use it for the contest, but ultimately finished the project during the following academic term.

### **5.3 Redefining the Goal of Participation**

Another way that our project encouraged involvement was by allowing students to adapt the goal of participation to their own desires. For most students, the purpose or goal of their participation in the project is to create a workable solution to compete in the final contest. Certainly, many students are interested in the project for the learning experience they expect to have in the process, but the goal of creating a working robot at the end is a big part of the motivation. Some students, however, choose goals other than this norm as what was most important to them. The protracted design excursions discussed in the previous section are examples of this phenomenon—students deciding that what was most important was spending time on issues that may or may not be most effective in producing a functional robot.

From the earliest contest, there were examples of students who redefined the contest challenge in a way to make it more interesting for their own participation. During the *King of the Mountain* contest, one team chose to take a particular rule item, which was intended to eliminate the possibility of a purely mechanical solution to the contest, as a direct challenge.

### 5.3.1 Mechanical Intelligence

The *King of the Mountain* contest had a clause that decreed that robots would be placed at a randomized orientation during actual contest play. This rule was established with the intent of making it unlikely that a purely mechanical solution could be successfully built. Our supposition was that robots would have to do at least a little sensing and computation to be able to climb the hill, given that they could not count on the direction they would face at the start of the round.

One team took this as a direct challenge. This team set out to build a robot, which they later named *Stupid*, that would solve the contest without any sensing and computer control at all. The design solved the problem of climbing the hill through a sort of “mechanical intelligence” based on gravitational feedback. The robot they built was a simple walking device. It had a platform base and a pair of supporting legs. In operation, the legs would swing forward, lift the base off of the ground, advance the base forward in the air, and then deposit the base back down again.

But there was a trick. Every time the legs were raised in the air, they were free to pivot about their center. Due to the way the legs were balanced, they would rotate to seek a subsequent step that was uphill from the last position! So the robot solved the problem of climbing uphill without any need for “sensing and control” in the traditional sense. As soon as the robot was given power to walk, it would start climbing uphill, regardless of initial orientation.

The design needed no additional mechanism or control to stop at the top of the hill. When it reached the top, it would indeed take a step downhill (it had nowhere else to go) and was “too stupid” to stop, but then on the following step, it would swing around 180 degrees and climb back uphill again. The design was a simple but elegant solution to the contest problem.<sup>2</sup> (It should be noted that none of the hill-climbers *needed* a separate stopping mechanism: after reaching the peak of the hill, they would go beyond the top, but would then just turn around and head back for the top.)

---

<sup>2</sup>The mechanics required to implement *Stupid* were not trivial. In fact, a last-minute modification to the pivoting mechanism resulted in a jam during the contest performance, causing the robot to lose a round and eventually cede the first place contest position.

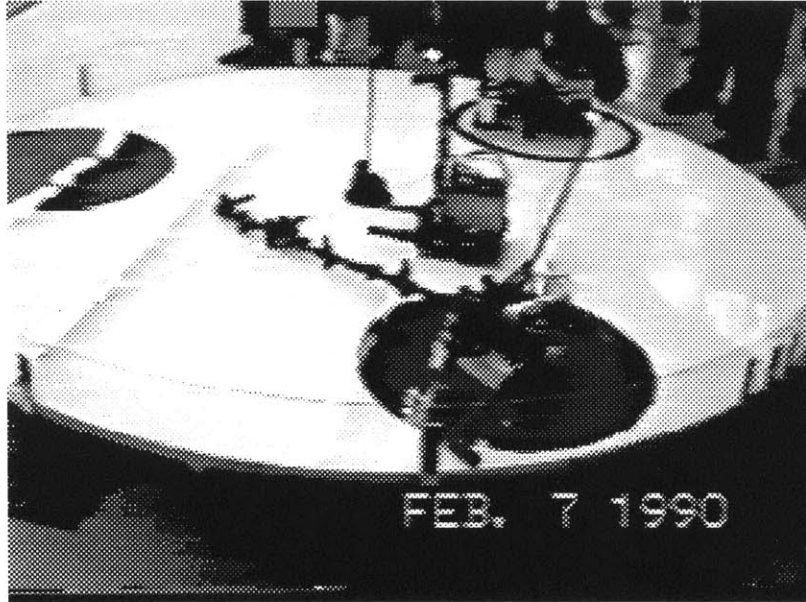


Figure 5-2: *Blind* unfurls its arm toward the puck

The same team of two students returned to collaborate on a design in the *Robo-Puck*, the subsequent contest.<sup>3</sup> Again the students attempted to foil the contest designers and develop a machine that did not rely on computer control.

The challenge of *Robo-Puck* was to locate and gain possession of an infrared light-emitting hockey puck. Again we employed the randomized orientation rule to discourage mechanical solutions. This time, the team's solution was based on the fact that while the *orientation* between one's robot and the puck was thus randomized, the *distance* between them was always the same. (This can be seen in the diagram of the *Robo-Puck* playing table, shown on page 74.)

The students created a massive arm that swooped out and scooped the puck from its initial position. In the spirit of *Stupid*, they named this robot *Blind*, since it did not use light sensors to locate the puck, but rather attempted to grab the puck from its starting position in the center of the rink, before either opposing robot would have a chance to displace it from this location.

Figures 5-2, 5-3, and 5-4 show the arm in successive stages of deployment, sweeping the

---

<sup>3</sup>Beginning in the 1993 contest year, it was decided that students would be allowed to participate as contestants in the project only one time. This was due to the large oversubscription of the class and the need for admission lotteries in the past several years.

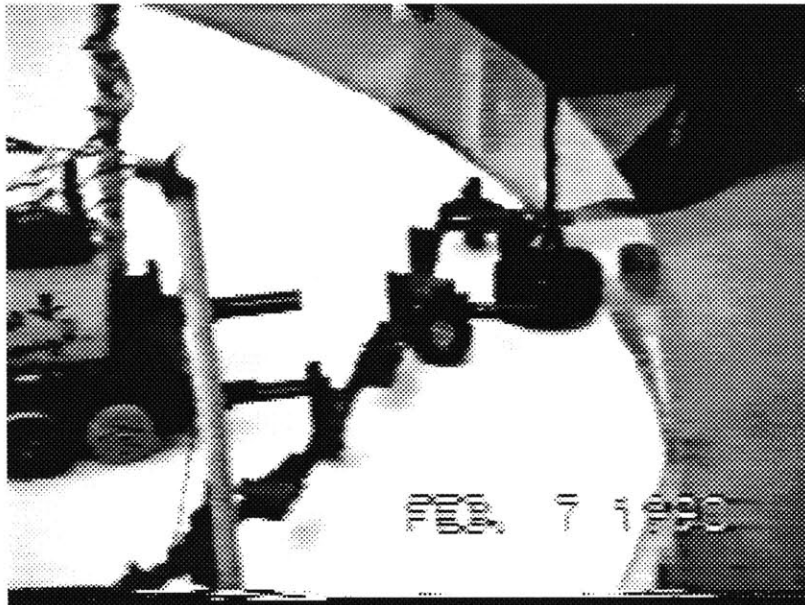
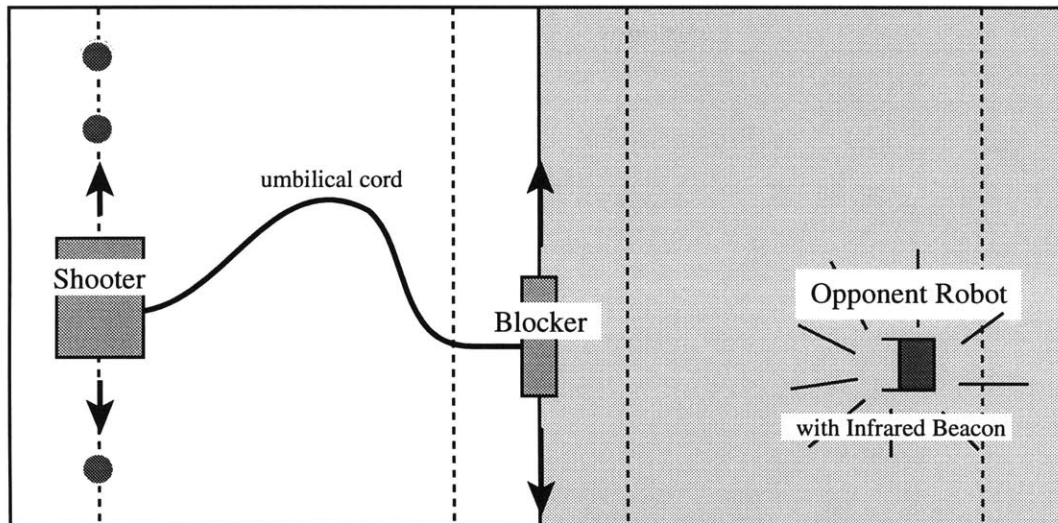


Figure 5-3: *Blind* brings puck to the rim



Figure 5-4: *Blind* loses control of the puck to *Bertha*



Shooter Robot moves back and forth in ball trough, picking up balls and shooting them over the hill

Blocker Robot tracks infrared beacon of opponent, moving along light-dark boundary to block its path

Figure 5-5: Strategy diagram for *Mutton Jeff* in *Robo-Pong* contest

puck from the center of the rink and toward the rim. In these figures, *Bertha*, a competent puck-fetching design, is hot in pursuit of the puck, and ultimately won the particular contest round. *Blind* did require a little bit of programming, to make sure that the arm triggered at the appropriate time, but the robot remained true to its designers' goals of being essentially a mechanical solution.

### 5.3.2 Deliberate Complexity

Other students took a more accommodating approach toward the objective of the contest, but added their own constraints to define what sort of robot they'd like to build. For example, a number of students perceived the flaw in the *Robo-Pong* contest design that would have allowed a simple but dedicated "assassin" robot to win, but rejected it for their own robot, deciding it would be too trivial a project. Students that did design attack robots ended up burdening them with enough unnecessary complexity that they failed to have the reliability needed to win the contest overall.

Some students deliberately chose designs that would be complex. One of the most

striking of these was a dual robot design created by a brother-and-brother team during the *Robo-Pong* contest. Each brother was responsible for one of the two essentially independent robots that made up the design. One robot drove into the ball trough and traveled back and forth, picking up balls and shooting them over to the other side. The other robot drove to the center plateau dividing edge and moved back and forth along the edge, tracking the opponent's infrared beacon (see Figure 5-5).

When the system worked, it made for an astounding performance. The blocker robot was effective in keeping the opponent at bay while the shooter robot delivered balls to the opponent's side. The robots sometimes had trouble disengaging from their starting configuration, but if they deployed themselves successfully they were effective and exciting to watch. The two students who designed the robotic duo were less concerned with it winning the contest, which it did not, than with it having at least one successful contest round, which it did.

Other students focused more on the *process* of their work rather than the final outcome. Here a student describes how he and his partner's proclivity for complexity got in the way of producing results:

Our design process is pretty haphazard. Darrin is a true hacker at heart, and we both just play around with ideas and try to get them to work. Since our big idea didn't materialize (probably more for lack of time and frustration rather than poor design), this random playing around may not have been as good as a consistent plan. Also, I think our team succumbed to a little to what Fred Brooks calls the "Second System Effect" in his book on software engineering, *The Mythical Man-Month*. Brooks' thesis is that when a designer builds his second system, he is tempted to include all of the widgets and neat ideas he thought of during the construction of his first system, but wasn't able to include because of time constraints, etc. This is dangerous because it can put the second system design seriously on the wrong track as the implementors try to get all of these cute hacks to work, and the system gets bogged down, "overdesigned," and becomes unwieldy because of its complexity. In the same way, our robot is a victim of this effect as Darrin and I tried to work in the neat ideas we had last year (rotating wheel assemblies, a differential, centrally located drives, etc.) As we found out, the design got out of hand and we didn't have the time to get it right. Perhaps we should have followed the KISS rule: "Keep It Simple, Stupid!"

### 5.3.3 The Talent Show

Another example of students' redefining the goal of their participation was a compromise to rescue satisfaction from a failed situation. Beginning with *Robo-Puck* in 1990, we had established two nights of contest game play. In the first night, one round of matches would be run (each robot would play one time), and results would count. The second night was the public event, drawing a large crowd to see the remainder of the competition, as many rounds as were necessary. We ran a double-elimination contest, meaning that a robot was allowed to lose once and continue playing until lost again. Hence, a loss in the preliminary night did not preclude a robot from playing in the public contest, though it would be eliminated after sustaining a second loss.

Beginning in the 1991 *Robo-Pong* year, we put a clause in the rules that stated that robots had to demonstrate “minimal proficiency” in the preliminary contest round in order to qualify for participation in the main round. We created this rule to help ensure a reasonably entertaining contest night—one without too many disappointing robots.

Minimal proficiency was defined as being able to win against a non-moving, “placebo” opponent. If a robot lost in the preliminary round, it would have to be tested against such an opponent before qualifying for the main round. Students were given until midnight of the eve before the main contest to make their robots capable of defeating “the placebo.”

In the *Robo-Pong* year, this qualification rule was essentially moot, as any robot that could move around the table at all was likely to “accidentally” hit one of the center plateau balls over the top of the hill and onto the opponent's side, allowing the robot to claim victory. Indeed, the *Stupid Scorpion* robot (described in Section 3.1.2) created in a last-ditch effort to get a working machine, used a rudimentary back-and-forth movement strategy and placed third overall in the *Robo-Pong* contest, much to the surprise of its creators. In 1992's *Robo-Cup*, however, a robot had to demonstrate a series of competences in order to score a single point: it had to navigate to the ball dispenser, trigger the panel to feed a ball, and then direct the ball to reach its goal. The qualifying rule clause thus became a serious affair, determining participation in the main contest.

As it happened, one team in *Robo-Cup* either decided or realized, before the night of the preliminary round, that they would be unsuccessful in qualifying for the performance in the

main contest. They then arranged to have their “performance” take place in the preliminary round: they invited a number of their friends to attend (though the preliminary round was not publicly advertised, it was open to an audience, provided they didn’t complain about the pace of the contest rounds!). They decorated their robot with various accessories, and then programmed it to play a song and dance (“La Cucaracha”) *rather than try to solve the contest*. Their friends gave a big cheer when their turn came, and the robot functioned as it had been programmed:

José Luis had proposed adding music to the robot, so I got a small speaker; we think it will play “La Cucaracha,” which prompted the name “BAYGON”—a household insecticide in Latin America.

I was not aware that the students had planned this until witnessing this event as it was occurring. Afterward, I attempted to persuade them to not give up yet (they had until midnight of the following evening to get their robot to qualify), but they had had enough of the project. While I was glad that they had the creativity and conviction to make the most of their preliminary round performance, it upset me that by the rules we had established, they would be kept from competing in the main contest round. Surely our rules weren’t meant to keep dedicated students from having their chance to display an interesting robot!

It was too late to change the rules for these students’ case. In subsequent years, rules were revised to allow students whose robots had been disqualified to create show robots for the main contest performance, or to compete against other disqualified robots in an exhibition round.



# Chapter 6

## Conclusion and Future Directions

This dissertation has been concerned with the relationship between two central questions:

- How does one design an evocative learning environment (based on robotic design) for undergraduate-level students?
- What do students do when confronted with this learning environment?

As suggested by Figure 6-1, the meanings of these two questions is defined by exploring the relationship between them. The learning environment presented to our students, as described in Chapter 3, consists jointly of the physical hardware and software platform, the structure of the contest challenge, and the pedagogical organization of the Robot Design course. Through four successive iterations of our project, this dissertation has presented the data of our work in creating environments for robotic design, and the students' work of using these materials to build robot projects. This dissertation has presented a set of interrelated issues that have been studied: our learning how to build robots, our learning how to design materials for robot-building, and our learning what students are learning when they build robots. Thus the thesis has not only documented the students' experiences, but our own learning process in developing the Robot Design environment.

In analyzing students' approaches to control, we saw a strong tendency for building lock-step, algorithmic processes that expected or required precise and accurate sensory readings. Students expected the sensor data to provide them with unambiguous information for interpreting their robot's relationship to its environment. The approach toward control

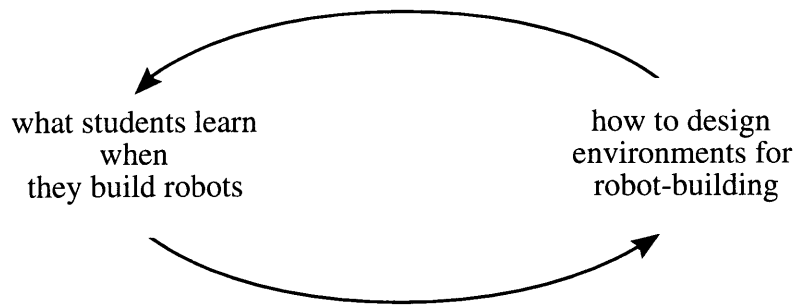


Figure 6-1: Designing robot-building environments and studying learners' experiences with them

required for a robust robot was unfamiliar and uncomfortable to the students. Many students interpreted their robots' failures by pointing to particular components of the robot which had "failed" rather than the architecture of their overall solutions, which were fundamentally vulnerable to such failures. I conclude that the robot design experience presented students with a problem that strongly challenged their notions of systems and control. This sort of experience is necessary in order for students to reconsider their deeply held views.

This study has also revealed a disparateness between phenomena that students observe during their robot-design processes and the formal models of engineering science that may be used to explain such phenomena. When students are confronted with puzzling situations during their design activities, they cannot readily produce the engineering science that may be used to explain the phenomena they observe. This happens even though it may be demonstrated that students "possess" the requisite knowledge in other contexts. Teaching students theory in traditional academic contexts does not mean they will have access to this knowledge in design-oriented settings; yet, as fledgling engineers, design situations are precisely what they will face. Therefore I conclude that design experiences like those encountered by the students of the Robot Design should be a fundamental part of the undergraduate curriculum.

In this conclusion, I revisit the central themes from the three main chapters of this dissertation. Interwoven with these summaries are suggestions for future directions of work, and preliminary results based on the influence of this work on the larger community of robotics researchers and university educators. First, within the context of the design of our educational technology, I discuss the level of abstraction of the materials provided

to students, including the approach we settled on, choices made by other educators for their own courses, and the potential for developing robot-design materials for secondary education. Next, I return to the issue of students' approaches to the problem of control, and suggest a way of modifying the robot contest specification to further the valuable challenges provided to the students. Finally, I revisit the discussion on design styles, and argue that irrespective of students' approach to the design problem, they encounter the difficulty of reconciling the phenomena they encounter with their formal engineering design knowledge.

## 6.1 Educational Technology

Chapter 3, "Technology for Learning," explored a set of issues related to the design and assessment of an educational technology—specifically, a technology optimized for the construction of small mobile robots and the exploration of issues related to the design of the same. We examined this technology from several perspectives, including the design of the robotic environments (the contest specifications), the design of a hardware and software platform to support students' robotic projects, and the creation of sensor devices to enrich the robots' capabilities and the students' potential for learning.

The problem of designing effective contests involved a number of issues. The contests had to focus students on the task of creating a viable robotic device, but at the same time, provide freedom within the constraints, and encourage a multiplicity of solution strategies. One of the primary goals was to impart the attitude that there are lots of ways to solve a problem, not one necessarily right one.

Through the iterative process of designing and evaluating contests and course versions, we learned the structure of the "problem of the problem,"<sup>1</sup> that is, the challenge of designing and operating the project. The contests, hardware and software, and sensor devices were all part of the overall problem of developing, structuring and administrating the class; of creating an environment that is rich in learning opportunities, provided enough support that students feel able to take charge of their own experience, but did not limit them or constrain them from diving into pockets of interest that arise. In a large part, the chapter was a thick

---

<sup>1</sup>This phrase was suggested by Donald Schön.

description of the issues that we faced as course designers, how we dealt with them, and my own interpretations of the resulting effects on students' learning.

### 6.1.1 Levels of Abstraction

A turning point in the evolution of the project occurred when we decided to create the robot programming environment based on the Interactive C language, which replaced the earlier assembly language methods. By providing students with a higher level of abstraction for interacting with the robot hardware, we allowed them to focus on issues of strategy and algorithm rather than details of registers and bitmasks, and to create much more sophisticated robot systems. The alternate choice of providing the lower level of microprocessor programming would be equally valid given different pedagogical goals.

The development of the Interactive C system was partly motivated by problems we observed students experiencing with the assembly language system. The hardware and software of the assembly language materials suffered from quite poor *observability*: when something went wrong, it was very difficult to tell at which level of the system the problem lay. In other words, it was difficult to ascertain whether a perceived difficulty was due to a catastrophic hardware failure, a hardware problem in a sensor, an algorithmic problem in interpreting a sensor's data, or a software problem in the algorithm for interpreting a sensor's data. As described in Chapter 3, this commingling of possible failure modes caused a great deal of frustration among the students.

By providing students with tools to discriminate among these failure modes, the Interactive C system was a great step forward. But it did much more than this. It allowed students to express algorithmic ideas in a familiar and more readable form than assembly language; it allowed them to create more complex programs (which facilitated the students explorations into the control issues discussed in Chapter 4). This expressive power, however, came at a price: students were insulated from a variety of the details of their robots' operation. Some degree of abstraction is part of any project; one does not discard the infrastructure of technology at every turn. The question we faced was whether the abstraction we had introduced was a worthwhile tradeoff.

When we were developing the Interactive C system, part of our justification was that

it would solve the observability problems we had experienced with the assembly language system. Most of these problems, however, could have been addressed while preserving the assembly language system. For example, the “system heartbeat” indicator, which revealed a catastrophic hardware problem at a glance, could have been installed on the assembly language system. Similarly, code libraries could have provided the capability to display sensor values on the computer console in a similar manner as did the LCD display on the Interactive C system.

Nevertheless, we could not have created the degree of interactivity that we did with Interactive C with the assembly language materials. While the level of abstraction matter can be argued in either direction—depending on one’s educational objectives, sometimes it’s better to give more expressive power; sometimes it’s better to keep students close to the level of the hardware—the interactivity of the C language system gave students the chance to rapidly try out their programming ideas, create more sophisticated programs, and explore issues of sensing and control in greater depth.

### **6.1.2 Robot Design at Other Universities**

After the end of the 1991 course, we realized that the robot-building technology we had created could be of value to others who were interested in robots, both for educational purposes (as in our project) and for research purposes. By 1992, we arranged with the MIT administration for free distribution of the course technology, including the controller boards, the custom Interactive C software, and the 250-page course notes (Martin, 1992). We also released a revised version of the more simple 1990 hardware, which we called the “Mini Board” (Martin, 1993).

In addition, the project received publicity through popular media exposure in monthly magazines (Arneke, 1990; Freedman, 1990) and daily newspapers (Chandler, 1992), research publications (Martin & Sargent, 1992; Martin, 1992, 1993), and via Internet newsgroups (the electronic equivalent of “word of mouth”). An eclectic community comprised of robotic hobbyists, academic educators and researchers, and industrial professionals came into being, brought and held together largely by the existence of electronic communication (i.e., e-mail). Our technology was adopted by these users because of its unique packaging of

features suitable for robot design projects, its ease of use, and, more recently, the existence of a community of supportive users.<sup>2</sup>

The process by which this community arose and now flourishes is a matter of separate interest (it would not have happened without the widespread adoption of electronic mail within the professional and academic world), but here it is valuable to note the ways in which the materials have been employed by others for a variety of purposes. I have observed three broad categories of usage: educational initiatives, research projects, and hobbyist/professional purposes.

The choice we faced in choosing the higher-level Interactive C environment has also presented itself to course developers at a number of universities who have started courses inspired by the MIT Robot Design project. Many of the courses others have created share the central activity in which students design their own little robots, but they have different specific pedagogical goals which depend on the interests of the various course developers. This illustrates both the adaptivity of the technology we designed, and the creativity of university educators in building their own learning environments.

At the time of this writing, I estimate that there are between twenty and thirty such projects. Here I present just a few.

**A microcomputer architecture course.** At the New Mexico State University (NMSU), Professors Patricia Teller and Joseph J. Pfeiffer, Jr. are using a version of the technology developed for the Robot Design project in their sophomore-level class on microcomputer architectures. Their class is built around the activity of designing and programming a small robot built from LEGO parts and operated by a microprocessor controller.

In the NMSU class, students program in assembly language. This was a deliberate decision by the course designers because they wanted students to learn about the inner structure of microprocessors. Professors Teller and Pfeiffer report that because of the hands-on, physical action of the robots' motors and sensors, students have been more

---

<sup>2</sup>This observation is based on an informal survey I recently conducted of members of this electronic community.

successful in learning assembly language than in previous courses.<sup>3</sup>

**A senior design course.** At Bucknell University, Professor Thomas Sloane has been running a senior design course based on a robot design contest since 1991, inspired by our *Robo-Pong* contest. In his version of the design project, however, students design the microprocessor control hardware in addition to building robots to solve the contest problem. As he describes the project, “different systems come and go depending on the participants’ goals.”<sup>4</sup> Thus, in the Bucknell course, students learn digital design as a part of their robotic projects.

**A programming course for mechanical engineering students.** At the University of Wollongong in Australia, Peter Dunster has introduced hands-on robotics projects within the Mechanical Engineering department. He was motivated by a “woeful lack of programming knowledge and almost no practical experience” in his final year students. With the support of a department head also looking to bring more practical subjects into the syllabus, he has established a successful project course.<sup>5</sup>

**An undergraduate communications course.** At the Trondheim College of Engineering in Norway, Professor Fredrik Wilhelmsen is using our technology with undergraduate students to teach data communications and distributed systems. One project involves simple interacting robots, and another employs a controller board operating a video camera through an infrared link. Professor Wilhelmsen cites the need to broaden students’ understanding of the potential of computers: “The nice thing about the Mini Board is that it makes the real world easily accessible from the computer. It is very easy to connect sensors and actuators. . . Most students think that the screen and the printer is the only way to get [information] out of a computer.”<sup>6</sup>

**An introductory graduate course in robotics.** In the Computer Science and Engineering department at the University of Connecticut, Professor Robert McCartney is

---

<sup>3</sup>Personal communication, 1993.

<sup>4</sup>Personal communication, 1993.

<sup>5</sup>Personal communication, 1993.

<sup>6</sup>Personal communication, 1994.

offering a graduate course (spring 1994 semester) in which students build their own robots on various mechanical platforms (modified toy cars and simple three-wheel bases) and use the same control technology developed for the MIT Robot Design project. In Professor McCartney's course, the emphasis is on designing robots that can wander around an office building and construct a map of the floor plan. By providing students with materials to build their own robots, and a problem in contemporary robotics research, he hopes to combine the practical with the theoretical.<sup>7</sup>

In each of these examples, the individuals involved have adapted the technology to their own idea of what is important for students to learn. A common concern is providing practical experience to students, and a belief that learning-by-designing is valuable. Beyond this there is variation with respect to the issues discussed in Chapter 3: the level of abstraction, modularity, and structure of problem posed to the students.

### **6.1.3 Robot Design in Secondary Education**

Many educators agree that children should be "technologically literate," but there is little consensus as to what this means in practice. While there are exceptions, typical attempts at creating technology classes suffer the same problems as much of school: textbook-based and traditional laboratory approaches which do not facilitate an empowering relationship with technology. The Robot Design concept would allow younger students to explore technological ideas in a way that lets them reflect on their own creations and realize the significance of systems thinking in a first-hand way.

The *LEGO tc logo* materials discussed in the Background chapter have been marketed by the LEGO company for use by middle school (aged 11 through 14) children; LEGO Dacta USA has recently announced a substantial revision of this product, called *LEGO Control Lab B*, which is aimed at high school children. A similar repackaging of our technology, originally developed for university use, would allow younger learners to explore ideas of sensing, control, and engineering design.

This work to adapt the materials created for the Robot Design project for use by high

---

<sup>7</sup>Personal communication, 1994.



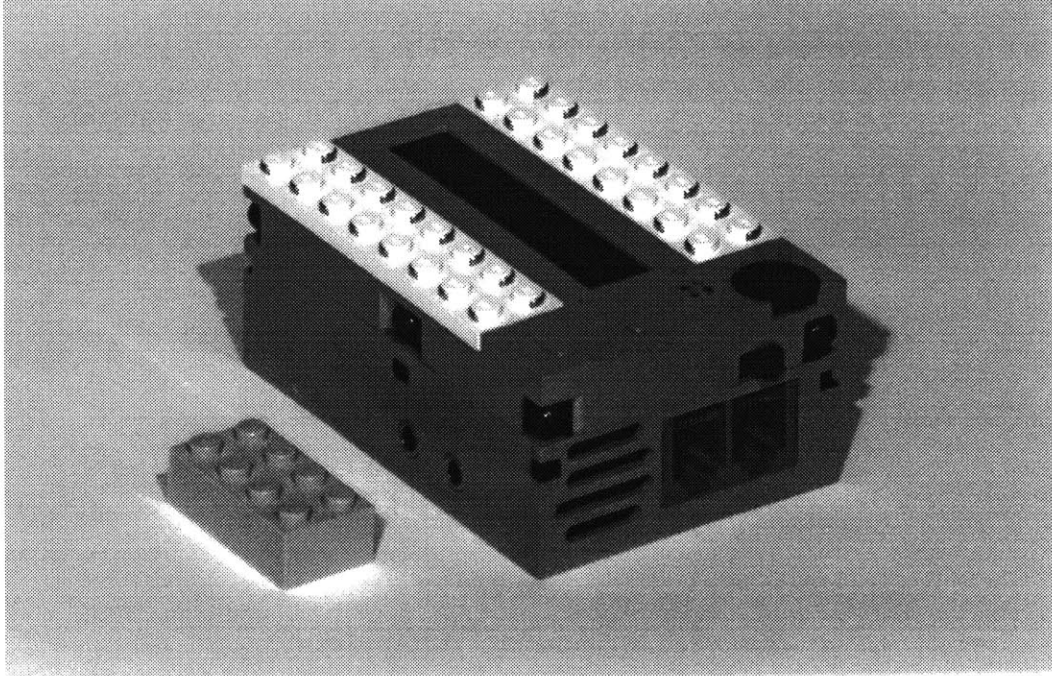


Figure 6-2: The 1994 version of the Programmable Brick

school students has already begun. In the Epistemology and Learning Group at the MIT Media Laboratory, Randy Sargent has led the design of the next generation of the LEGO Programmable Brick (shown in Figure 6-2). Building on experience gained in the first Programmable Brick work (discussed in the Background chapter), the Programmable Brick puts all of the electronics in a small plastic box and provides nice connectors and an easy to use programming environment. As one example application for the Programmable Brick, Sargent plans activities in which children will explore the concept of embedding computationally active elements in the environment (sometimes referred to as *ubiquitous computing*), allowing them to create, for example, a room that will detect when a person enters it, turning on a light or playing a greeting message (Sargent & Resnick, 1993).

With the Programmable Brick or similar technology, it is feasible to develop curriculum plans and project structures to give high school students the chance to build competitive robots like the ones that have been developed by the MIT students in this study. Whether or not they intend to pursue careers in science and technology, children deserve the chance to explore ideas of sensing, control, and engineering early in their academic careers, and in a way that encourages and develops their own creativity and sense of mastery. It is

unfortunate that many school activities relating to mathematics and science have the effect of deterring children from these fields of inquiry. If children are given the chance to engage in personally meaningful design projects with a rich technological content, the result might be very different.

## 6.2 Models of Control

Chapter 4, “Ideal and Real Systems,” explored students’ notions of systems and control as instantiated in their robot designs. The picture that emerged from this study suggests that students’ pre-existing notions of control led them into trouble when they approached the robot design task. Students had a great deal of difficulty understanding the properties—and limitations—of the robotic sensor devices, and more generally, the interface between the computation and the real world. It became apparent that students were accustomed to constructing algorithms that operated on precisely known quantities, as in an algorithm to sort items in a list. Most students constructed robots whose solution algorithm was quite fragile with respect to unanticipated situations, which in practice happened with disappointing frequency. This caused some students to question their models of control, but most simply chalked up the failures as being due to “Murphy’s Law” or limitations in their ability to execute a design which they still believed to be correct. For many, however, a seed of doubt has been planted, and I expect these students to be less surprised in the future as they discover that the real world is much more complicated than the clean models of it that they have been taught.

The problems given to the students in the Robot Design contests brought particular control problems into focus. In general, the contests contained sub-tasks that were amenable to local, negative feedback control (e.g., climbing a hill, following a line, or following a wall) within the context of overall tasks that involved a fair bit of uncertainty (e.g., the actions of the opponent robot(s)). While some students avoided the local, low-level feedback approach in favor of attempts to centralize control through predictability, most realized the utility of the local feedback techniques. Few students, however, created robots that dealt with the uncertainty of the overall robot behavior problem.

An interesting issue for further study is the question of how closely this sort of engineering problem corresponds with the types of scenarios engineers encounter in the real world. Certainly the concept of negative feedback is central to real-world control systems, but it would be fascinating to study the correlation between vicissitudes in the world of the students' robots and the world of installed controls. When engineers build anti-lock braking systems for cars, do the sensors perform as erratically as those on the students' robots? Probably not. On the other hand, critical early warning radar defense systems have fallen prey to various forms of sensor misinterpretation (e.g., unanticipated radar reflections from the moon after more powerful radar was installed were interpreted as a missile attack by automated hardware/software systems) (Roberts & Berlin, 1987)). The lessons about the dangers of overly trusting sensors are quite apropos to the challenges of contemporary engineering.

### **6.2.1 Artificial Life and Behavioral Robotics**

In fairness to the students of the Robot Design project, the contest specifications and the Interactive C language conspired to make it difficult for students to create robust solutions. Because contests were so short, between thirty and sixty seconds, winning robots had to go about their task quickly and without making any missteps. And because Interactive C is a traditional procedural language, it tended to encourage sequentially organized programs.

A yet more challenging robot design contest would require robots to perform a task for a significantly longer period of time (e.g., ten minutes). In this kind of contest, students would be forced to think about the control task in an entirely new way, along the lines of recent work known as *artificial life* and *behavior-based robotics*. In this work, synthetic entities (e.g., mobile robots) have very different control architectures that allow them to “survive” in unstructured environment for extended periods of time. Translated to the sort of robot design tasks that have been discussed in this dissertation, these robots would have to navigate freely around a playing field, recover from collisions with other robots, and perform some kind of task that would require a longer period of time to complete.

This kind of contest might not have the high-action thrills that characterized the previous contest challenges, but it would require students to construct robots that used behavioral

responses to environmental stimuli rather than robots that ran a pre-designed specific action loop. Along with the change in contest task, it would be appropriate to introduce a control language along the lines of the activation networks suggested by Pattie Maes (Maes, 1990) and the layered behavior mechanism suggested by Rodney Brooks (Brooks, 1986). Such a contest would bring these alternate models of control to the forefront of students' attention, and would provide the structural motivation for them to explore these models.

### 6.2.2 The Role of Sensing

Because sensors are a robot's interface to the real world, the types of sensors that are available have a large influence on a robot's potential abilities and the richness of the robot control problem for the student. Many of the issues that students grappled with during their project work was related to a process of coming to understand the actual (as compared to ideal) functioning of a particular sensor device.

For this reason, the area of sensor development is particularly ripe for making improvements to the technology of the Robot Design project. Appendix D.4 contains a discussion of our process of developing the infrared robot-sensing technology; the use of this sensor, as was the case with many others, led to a variety of problems that were quite valuable from a standpoint of students' and our own learning.

It so happens that all of the sensors we have used are fundamentally simple devices that provide data that is mostly local in nature. A more powerful microprocessor unit, however, would provide the foundation for highly data intensive sensors—specifically, a electronic camera to give robots visual capability. The development and inclusion of such technology into the Robot Design concept would allow at least two different directions of project work:

**Sophisticated robot strategies.** If the vision system were supplied with predesigned functions for its use (as was the case with our other sensors), students could make use of the vision system without a detailed understanding of machine vision principles, allowing more complicated game designs and students to create more sophisticated robots.

**Hands-on machine vision projects.** Such a system would also allow students to ex-

plore the problems in machine vision in a hands-on way. The subject of machine vision is often taught via conventional means, since it's not typical to give each student his or her own robot, camera system, and algorithm development platform. But a hardware, software, and workshop approach that did specifically do this is a logical extension of the work developed in this thesis.

As miniature camera technology continues to drop in price and become easier to use, this concept will become eminently practical.

## **6.3 Design Styles and Engineering Knowledge**

The crux of the problem of engineering education is helping students bridge the gap between the formal engineering science knowledge and the observed phenomena of a design situation. In their robot-building work, students were frequently confronted with confusing situations that stemmed from their use of incorrect, or lacking, models of the underlying engineering phenomena. In this type of problem-solving, the challenge is to figure out what the problem is; i.e., to come to understand what type of explanation underlies the phenomena being observed. This is a very different sort of problem-solving—it is more a *problem-finding*—than is fostered by traditional academic contexts. Students need to be given these intellectual opportunities, since in their work as professional engineers and designers, the hard part of solving a problem or doing a design is figuring out the right questions to ask to get a handle on an underconstrained situation.

For the purposes of this conclusion, I would like to reinterpret one of the events discussed in the earlier chapters (the motor driver conundrum discussed in Chapter 5) and offer some additional thoughts on the role of the teacher in facilitating students to reflect on the lessons of their robot-building experiences.

### **6.3.1 Mysterious Phenomena**

The motor driver situation was the result of students' reaction to what they perceived to be a frustration with the stock motor electronics that they were provided in the *Robo-*

*Pong* year. Students built assemblies that operated adequately when controlled from the manually-operated motor switch board, but then performed poorly when driven from the stock electronic circuit. A number of students then chose the “design move” of developing their own control circuits to provide improved performance.

This activity was fraught with problems. The first was in constructing control circuits that did not violate the parts count limitation we had imposed on custom designs. The teams that did satisfy this constraint then discovered the surprising problem that their motor power output became highly dependent on the battery charge level.

In retrospect, an analysis of the situation revealed an electrical problem in which the motor power level was governed by the equation  $P = V^2/R$ , where  $P$  is the power level of the motor,  $V$  is the voltage of the battery, and  $R$  is the effective resistance of the control circuit. In their custom circuits, the students had greatly reduced the value of  $R$  in order to achieve greater power, and had inadvertently made the actual power level much more sensitive to changes in  $V$ .

At the time, neither we nor the students who had build the circuits were able to make this kind of analysis of the situation. It was clear that the performance problem was related to the new circuitry the students had installed, but we had not predicted it nor did we have a formal explanation for it. The situation was particularly confusing because other students who had *not* modified their motor circuits experienced a similar problem (high sensitivity to battery voltage). It turned out that there was a particular configuration of the gear train that would also cause the same effect.

(It is worth noting that there may be other interpretations or explanations for the battery voltage sensitivity problem; here I have not actually justified why I believe a  $P = V^2/R$  relationship to be governing the situation. The point, however, is not which explanation is correct, but rather the process by which we come to believe one over the other. We want students to be making these kinds of arguments as part of their process of learning to become an engineer.)

The overall situation was a significant drain on our resources for assisting students, and the result was that the students who had modified their driver circuits did not achieve the overall performance benefits they had anticipated. While their robots were indeed more

powerful, the students were unable to harness this power in their programming. We had not yet figured out the technical reasons behind the battery sensitivity problem, and our response to the situation was to *steer students clear of the problem*. In the subsequent year, we made three changes in this regard: (1) we deliberately “crippled” the manual motor switch boards so that students wouldn’t be tempted by seeing a power level that they wouldn’t be able to achieve in their electronic control; (2) we incrementally improved the performance of the electronic control; and (3) we disallowed modifications to the stock electronic circuit.

I was not altogether pleased with this solution at the time, and I consider it even less desirable in retrospect. The motor driver experiments were a rich source of learning for those students who participated in them; while we did reduce the need to explore this path, we also unequivocally shut it down. With our current ability to explain the phenomena that were mysterious at the time, perhaps we could re-open this area for interested students to explore.

### **6.3.2 The Role of Reflection**

Perhaps a shortcoming of the Robot Design project as an educational venue is the fact that there is no formal opportunity for students to reflect on their experiences after the contest itself; due to time constraints, the contest is the last gathering for course participants (with the exception of students who choose to prepare their robots for the rematch competition held at the Boston Computer Museum a few months later).

The educational challenge is to find the best forum or circumstances for students to consider the lessons of their design work. In the “exit interviews” we conducted with students a few days before the contest, I was struck by the degree to which students had difficulty talking about their experiences in the course. Students who were quite involved and animated during the design process often had little to say about what had been meaningful to them or what they had “learned.”

Further, students found it difficult to abandon their deeply held views about systems. Even after seeing robot after robot fail during the contest, most students would complain of a particular sensor or mechanism that had failed on their robot rather than consider the possibility that the overall approach toward control they had taken was fundamentally

susceptible to component failure.

Experiences like the Robot Design project should not be isolated events in our educational systems, and there's certainly no reason they should be restricted to a university setting. The best way to make the theoretical knowledge that we consider important valuable to a student is to give him or her a chance to put it to use. When we discover the gap of "messiness" that lies between theory and practical applications, we should not ignore it or toss it away as uninteresting, but encourage ourselves and our students to dive in and explore the complexity of putting ideas into practice.



# Appendix A

## Robot Glossary

This appendix section briefly describes each of the robot projects discussed in this thesis. The glossary is organized in alphabetical order by the robot's name.

**Baygon, the Happy 'Bot** *Baygon* was a robot produced by the team of Jorge A. Calvo, Jose Luis Elizondo, and Michel Montvelisky for the *Robo-Cup* contest. They realized they would not be able to qualify for the main contest, so they invited their friends to see their robot perform a surprise song-and-dance routine during the qualification round. See page 175.

**Bertha** A contestant in *Robo-Puck*, *Bertha* was a successful puck-fetcher that used a single infrared sensor to locate the puck. *Bertha* was designed by Cheryl Aittama and Chris Palmer and is pictured on page 75.

**Blind** The second robot of the David Hogg/Katie Lilienkamp team, *Blind* competed in the *Robo-Puck* contest with a method for capturing the puck that did not require sensing it. *Blind* exploited the fact that the initial position of the puck was a known distance from the initial position of the robot. *Blind* is pictured in a three-sequence animation shown beginning on page 171.

**Crazy Train** The champion of the *Robo-Pong* contest, this robot was a collector-style tank-like machine which scooped balls into its body. *Crazy Train* was built by John Kerwin, Gregory Gancarz, John Lum, and Dave Lum. See page 131.

**50/50** A contestant in the *Robo-Pong* contest, this was an aggressor robot that used a radar-like infrared sensor that could locate and track the opponent robot independently from its own movement. Built by Michael Gull, Henry Chung, Alex Wu, and James Sarvis, *50/50* lost when it apparently detected a reflection of the opponent's infrared beacon rather than the beacon itself. See page 248.

**Groucho** A contestant in the *Robo-Pong* contest, *Groucho* was a capable collector-style robot that was so named because it had a “face” that employed LEGO beams to simulate the eyebrows on the famous comedian. Also, its creators, Ed Tobin, Matthew Lee Domsch, and Adam Skwersky wore “Groucho glasses” on the eve of the contest. *Groucho* is shown in Figure 4-1 and discussed on page 133.

**Juicy Chicken** A contestant in the *Robo-Cup* contest, this robot was a shooter that had a separate baffle mechanism intended to assist the shooter in delivering balls to the goal. The baffle would position itself immediately outside the goal and deflect balls shot toward it into the goal. Additionally, the baffle carried an incandescent lamp which the shooter could track for aiming. Created by Bill Kaliardos, Thomas D. Wu, and Paula Bontá, *Juicy Chicken* was a complex design which was never successfully debugged. See page 81.

**Mutton Jeff** A sophisticated dual-robot design created by the brothers Matt and Tim Wall, *Mutton Jeff* solved the *Robo-Pong* contest with separate offensive and defensive robots joined by an umbilical link. The offensive half, which carried the computer, shot balls over to the other robot’s side while the defensive half prevented the opponent from crossing over the playing field median. See page 173.

**The Shotgun** A contestant in *Robo-Puck*, this robot was notable for its creative use of infrared sensing in a strategy based on shooting a claw at the puck. Because of its speed, the robot was unbeatable when it fired accurately, but it sometimes missed its target. *The Shotgun* was created by Rajeev Surati and Tim Wall and is discussed on page 74.

**Stupid** The first robot by the Hogg/Lilienkamp design team, *Stupid* solved the *King of the Hill* contest without using any form of electronic computation. A clever mechanical design caused *Stupid* to climb uphill when power was applied to its motors. It defied the organizers’ attempt to create a contest that required programming as part of a viable solution. See page 170.

**Stupid Scorpion** A contestant in *Robo-Pong*, this robot was created in a last-ditch effort after multiple attempts at more sophisticated robots had failed. *Stupid Scorpion* simply drove back and forth, reversing direction each time it detected that it had gotten stuck. Remarkably, it consistently hit more balls onto its opponent’s side than it brought back onto its own side, and placed third overall in the contest. None were more surprised by its success than its creators, Richard Davis, Ben Renaud, and Francis Njie.

The viability of robots like *Stupid Scorpion* was interpreted as a flaw in the *Robo-Pong* contest, and subsequent contests were designed with a higher degree of difficulty (see discussion beginning on page 82).

# Appendix B

## Contest Design

This appendix provides the full text for each of the Robot Design competitions from the years 1989 to 1992.

### B.1 King of the Mountain, 1989

The Third Annual 6.270 Contest:

BATTLE OF THE LEGOS

Sponsored by SIX APPEAL, The Course Six Social Group  
with funding from EECS and Microsoft Corp.

Organizers: Mike Parker, Randy Sargent, and Fred Martin

#### DESCRIPTION

-----

Each contestant will be given all parts necessary to build a computer-controlled LEGO robot, including electronics, LEGO, motors, and sensors. We will spend IAP building the robots, and will hold several workshops to discuss different aspects of the design, including the electronic hardware, mechanical hardware, and software & interface. In February, a competition will be held to determine the winning machine! Over \$500 worth of prizes will be awarded, and all

contestants will be able TO KEEP their \$150 Computerized Lego Robot Kits.

THE PARTS & PLAYING FIELD

\*\*\* The Game \*\*\*

The competition will be a King of the Hill battle. The goal of contest will be to reach the highest elevation on a hill-shaped playing field, and to do so in an elegant and entertaining fashion. Limited offensive and defensive weaponry is encouraged; this will be further described.

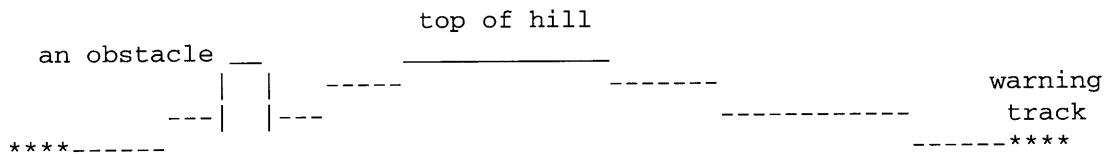
The playing field will be shaped like a slice off of the top of a hemisphere, with a flat patch at the top. Viewed from above, it will be a perfect circle with a diameter measuring approximately eight feet. The hill will have a positive slope in the direction of its apex at all points, except for the flat patch at the top.

There will be obstacles on the hill, in the form of rectangular or cylindrical blocks, mounted firmly into the playing surface. These obstacles will be no larger than one foot on an edge or in diameter, and will be painted bright red.

The playing field will be suspended above the ground, and machines may be pushed off or may fall off of the playing field. Cushioning material to catch falling machines will be installed.

The hill will be painted flat black in color. A highly reflective mylar "warning track" will be mounted in a circular stripe around the edge of the field. With an appropriate light sensor, it will be easy for a vehicle to determine that it is located on this warning track.

==== extremely crude image of cross-section of hill =====



====

\*\*\* The Materials \*\*\*

Each entrant will receive a hardware package valued at approximately \$150, which will become the property of the entrant after the

contest. This package will include:

- \* an ample supply of LEGO, including beams, axles, gears, wheels, and other parts from the LEGO Technics line.
- \* electronic hardware and plans for building a robot serial interface, which is compatible with any standard serial line.
- \* actuators, including motors, solenoids, lamps, and buzzers.
- \* sensors, including photocells, touch and mercury switches, and phototransistors.
- \* miscellaneous supplies, such as perfboard, glue, and wire.

Contestants will also be given documentation that will include schematic diagrams for various sensor designs and other helpful suggestions.

### \*\*\* The Computer \*\*\*

All machines must be solely controlled by computer. The afore-mentioned "robot serial interface" is ideally suited for this purpose. You may use any computer you wish, and write your control programs in any language you wish. Software for accessing the serial port of standard Athena VaxStations will be provided, and VaxStations will be available the eve of the contest.

### \*\*\* The Robot Serial Interface \*\*\*

The standard interface can power up to four motors, and receive information from up to eight sensor. Following is a technical description of the interface.

When the robot controller receives a serial byte from your computer, it writes it out to a register. A pair of "motor driver" chips (supplied) are wired to this register. Each two bits of this eight-bit register control one motor. Writing a "1-0" into a certain two-bit field will turn the corresponding motor on in one direction, while writing a "0-1" into the same two bits will turn the motor on in the other direction.

Each of these four "motor ports" can be used to control TWO non-directional actuators (e.g., lamps) in an on-off fashion.

The robot controller interface continuously broadcasts eight bits of digital sensory information back to the host computer, as standard serial bytes. Assuming a serial rate of 9600 baud, you will receive sensor samples at nearly 900 Hz.

A simple hack to get "analog" sensory information: use your analog

sensor (such as a photocell) to control the timing cycle of a 555 counter chip. The higher the resistance of the sensor, the longer the timing pulse; tune the thing to give you pulses in the range of 1 ms to 100 ms.

Run this timing pulse into one of your eight sensor ports. Now, back at your computer, write software to count the number of sequential samples that are 1's. This will tell you how long the timing pulse is, which corresponds to the amplitude of signal received by your sensor.

This scheme will give you a data rate of up to 10 Hz, which is not bad.

#### THE OBJECTIVE AND JUDGING

-----

The objective of the contest is to be King of the Mountain! to climb the highest up the mountain by the end of a two minute round. And, to do so in an elegant fashion, demonstrating ingenious and efficacious design.

Robots will battle in randomly chosen groups of three. In every battle, the performance of each machine will be scored. For each level of the contest, the robots averaging the highest scores will be selected to compete on the next level. The final score of every robot will be the average of all its battle scores. The robot with the highest average score will win the contest (2nd and 3rd prizes awarded also).

Scoring and selection of robots for battle will be done by a panel of three judges. Half of a robot's score in a battle will be based on its ACHIEVEMENT at being king of the mountain (e.g., it having reached a higher elevation than the other robots in the battle round).

The other half of the score will be qualitatively based on the robot's

- \* INGENUITY of design (e.g., a smoothly operating machine)
- \* AESTHETICS of visual presentation (machine AND contestant)
- \* CLEVERNESS of approach to solving the problem
- \* ENTERTAINMENT VALUE as judged by audience reaction to the machine's performance.

The three impartial judges will be available for questioning at the robot-building sessions, especially for questions on what they are looking for in these qualitative areas.

#### THE RULES

-----

1. In your quest to be King of the Hill, you may decide to equip your vehicle with various offensive and/or defensive weaponry. Violence (in good spirit, and only between robots) is encouraged. Absolutely no violence is allowed between people or people and machines. This means that dangerous machines (or contestants) will not be allowed.
2. All vehicles must be controlled through the robot serial interface provided.
3. All entries must mount their interface on the top of their vehicle, in a position where the interface connector can easily be accessed.
4. LEGO is the sole material to be used for structural means.
5. Non-LEGO materials (such as motors, sensors, and other hardware) must be "LEGOized" by gluing LEGO parts to them, and then mounting them on your machine using LEGO.
6. Only the motors that are provided may be used to power your vehicle. The motors must be operated with the power supply included in the basic kit, and in the standard way.
7. To help machines find each other, each of them will carry a halogen light bulb and an electronic buzzer.
  - a. Both of these items must be mounted in an unobstructed fashion. The light must be mounted so that it is visible omnidirectionally, and the buzzer so that its sound is heard (approximately) equally in all directions.
  - b. The lamp and buzzer will blink and sound synchronously at a standard rate of about 4 Hz, using a circuit provided.
8. To encourage different designs, you will be given a sum of "contest discretionary funds" which may be used to purchase components to supplement the basic package already described. With this "funny money" you may wish to purchase additional LEGO parts, sensors and electronics, or other materials that will be available.
9. Parts trading between mutually consenting contest participants is NOT encouraged. Exceptions may be made in the case of (a) a shortage of contest parts, and (b) items being traded of comparable contest value.
10. Only materials provided by the contest sponsors may be used on the vehicle.

With regards to all issues of machine customization and standardization, decision of the judges is final. These rules are

subject to amendment and interpretation by the judges.

#### ADMINISTRATION

-----

Robot building sessions will take place Tuesdays, January 10, 17, 24, and 31, in the Electronic Classroom (37-312), from 6-9pm (with possible additional sessions on Tuesdays, February 6 and 13).

The Electronic Classroom is equipped with a MicroVax Athena Workstation for each contestant to test the control of his robot. (Contestants can also bring in their own computer.) We will also be building THE MOUNTAIN for contestants to test their robots.

Attendance to all sessions is expected. Here is a tentative schedule:

- 10 Jan - Introduction. Passing out Kits and Rules
- 17 Jan - Building the computer interface and sensors
- 24 Jan - Building the LEGO robots, Mountain available for testing
- 31 Jan - Programming the robots
  
- 6 Feb - Testing and optimizing  
Contest food, ad, and prize planning
- 13 Feb - "Dress Rehearsal" Contest  
Mega contest advertising
- 20 Feb - Contest! 7pm in 34-101

This \$5500 contest is being extensively funded by the EECS Department and Microsoft Corp. However, the department has requested that each contestant cover a small portion (\$50) of the \$150 of LEGOs and electronics being given him, to indicate his seriousness and to help cover costs. This is actually a great deal, since you get to KEEP all the LEGOs and electronics and any prizes you win!

This \$50 entry fee is due and payable now. Please make checks out to "MIT SIX APPEAL" and write "6.270 Entry Fee" in the comment area (your check is your receipt). Please send checks to Michael B. Parker (address below). YOUR ENTRY FEE MUST BE RECEIVED BEFORE THURSDAY, 22 DECEMBER, 1988 IN ORDER TO GUARANTEE YOUR ENTRY IN THE CONTEST!

Happy Battles!

PS: Check your e-mail regularly for further notices!

#### CONTEST ORGANIZERS

-----

Michael Parker (contest coordinator)  
East Campus Monroe Rm 303  
phone 225-6303



E-mail mbparker@athena.mit.edu

Randy Sargent (technical coordinator)  
Senior House Runkle Rm 604  
phone 225-6654  
E-mail rsargent@athena.mit.edu

Fred Martin (technical coordinator)  
Bexley Rm 401  
phone 225-9641 253-9783  
E-mail fredm@media-lab.media.mit.edu

## **B.2 Robo-Puck, 1990**

### **OBJECT**

To design and build an autonomous robot to keep the “puck” away from you opponents. The winner is the one who makes the last contact with the “puck.”

### **Puck**

- The puck will be a hockey puck that will be on some type of ball bearings or wheels.
- The puck will have batteries on it so that it can emit infrared light. The infrared light will be one way machines can detect the puck.
- The puck’s infrared beacon will be placed 1 foot above the playing surface. Best detection will be in the 1 foot plane.
- The infrared beacon will be broadcasting at a 40 KHz. This frequency will be easily detected by the Sharp IR detectors that are supplied.
- The puck may not be altered or destroyed in any way.
- The machines may not obstruct the infrared beacon from other opponents.
- The puck may not be intentionally flipped.
- If the puck inadvertently flips over, the judges will immediately upright it at the point of inversion.
- The puck may not be lifted.
- The infrared beacon must always remain in the 1 foot plane.

### **Period of Play**

- A period will last 100 seconds. Motor and actuator power will be allowed during the first 90 seconds, and other stored energy will be allowed during the remaining 10 seconds.
- The round will be started by the judges turning on the starting lights for the beginning 5 seconds.
- The judges will place the machines on the playing field at a random angle with respect to the center. The angle at which the machines will be placed will be the same for all machines during the same period.
- The contestants will have 30 second to place their machines on the field from the time the judges call them.

# ROBOPUCK

## 6.270 Design Project Rules and Restrictions

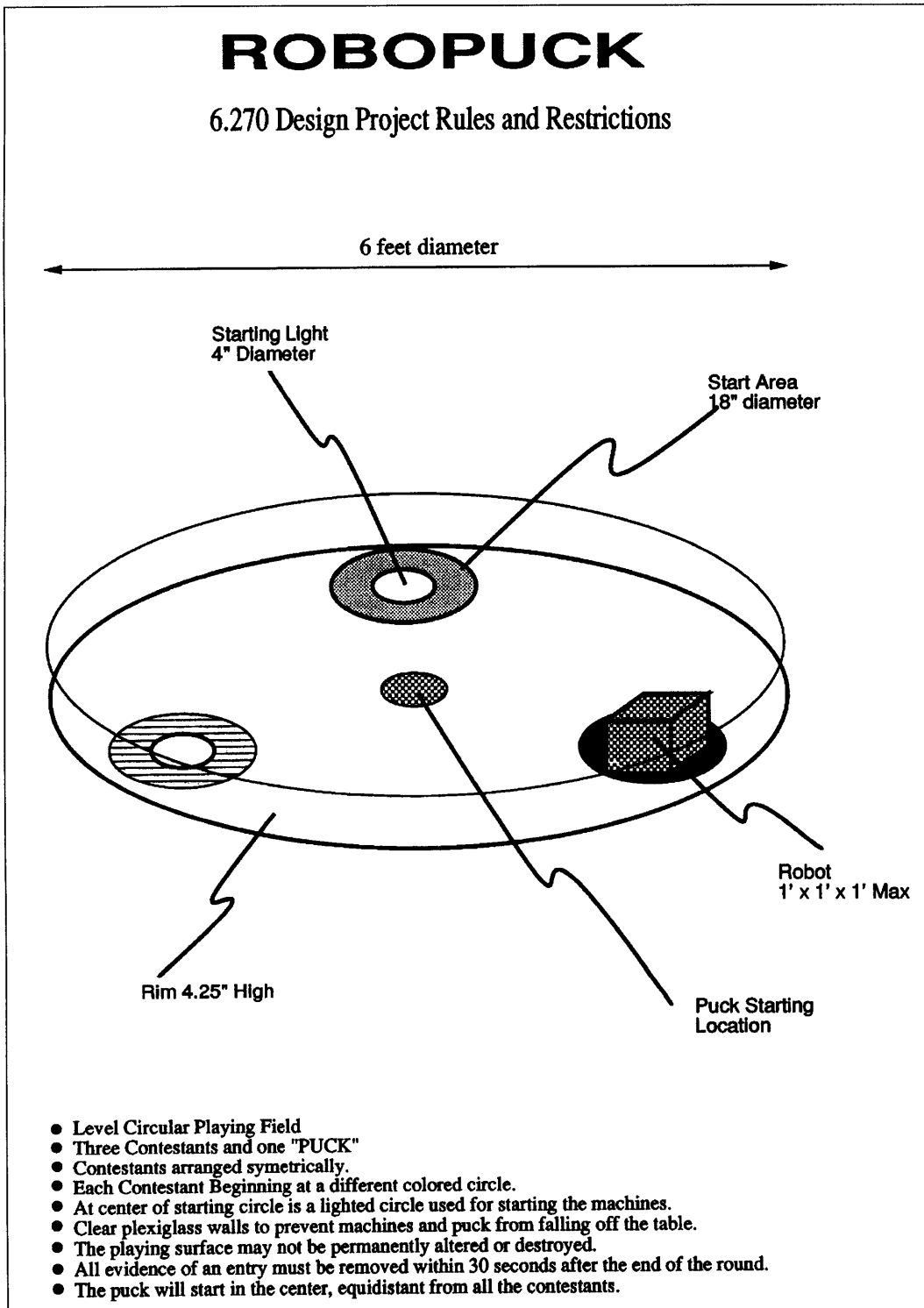


Figure B-1: *Robo-Puck* table specifications

- The contestants must stand a given distance away from the playing field. Any contestant that makes an attempt to touch their machine during the period of play will automatically be disqualified from the period.
- The machines must have their own internal clock which cuts off power to the motors at the end of 90 seconds. Any machines that continue to supply actuator power after 90 seconds will be disqualified.
- The contest will be an double elimination depending on the time constraints on the contest.
- All periods will have three players except those periods due to the elimination format only two people are scheduled to a period. In this situation, a placebo constructed by one of the organizers will be the third machine in the period.
- Machines may not be allowed to destroy their opponents microprocessor board.
- Machines may not try to destroy other machines broadcast or detection beacons.

### **Control**

- All entries must be solely controlled by their onboard computer. There is no human intervention once the period begins.
- All entries must be capable of broadcasting visual light at 2, 3, and 5 Hz. This visual light beacon will be used to detect other machines. Machines failing to meet this specification will be disqualified.
- Judges will assign frequencies for visual lights to the machines in the beginning of each period.
- After the start of the contest, there can be no change to the robot's program, or configuration switches made by the contestants.
- Infrared light may not be emitted from the robots.
- Machines may not use reflective material such as metal to reflect the infrared light from the puck.
- Parts that are likely to damage the internals of machines (ie. small pieces of wire, or barb) may not be dumped on the playing surface
- Any lego that is advertently dumped on the playing surface as obstacles will become the property of the 6.270 course.
- The two 6V batteries may be connected in parallel to provide greater currents, but may not be used in any fashion to provide greater voltage.

### **Structure**

- All entrants will be given an equal amount of Lego and Non-Lego supplies including:
  - \* an ample supply of LEGO, including beams, axels, gears, wheels, and other parts from the LEGO Technics line.
  - \* electronic hardware and plans for building a robot controller card, which can be programmed through any standard serial line.
  - \* actuators, including motors, and solenoids.
  - \* sensors, including IR dectors, touch sensors, photocells.
  - \* string, rubberbands, wire, solder
  - \* virtual material–glue, epoxy, WD40
  - \* The package will be valued at approximately \$300.
- The visual broadcasting beacon be at least 2 inches above the highest structural piece in the verticle plane above the machine. The light must be carried by the moving part of the robot.
- Only Lego may be used as structure.
- No Legos may be glued together.
- No Legos except for the base plate may be altered in any way.
- A non-lego part may be attached to at most 5 lego parts via glue.
- Wire may only be used for electrical purposes, and not structural.
- Rubber bands may be glued to wheels or gears to increase the coefficient of friction.
- Only Lego rubber bands may be used to provide energy.
- Contestants may not alter the structure of their entry once the contest has begun, but may repair broken components between rounds if time permits.
- The dimension of the machine may not exceed an imaginary 1 foot cube at the start of each period. (an exception is the transmitting and recieving beacon) Entries may however expand once the period has begun.
- Entries must be built completely from kit parts. All other parts must be removed before the beginning of the round.
- Lubricants may be used only to reduce friction internal to the machine.
- For orienting the machines, an arrow pointing to the front of the machine must be placed at a visible location for the judges.

## **Scoring**

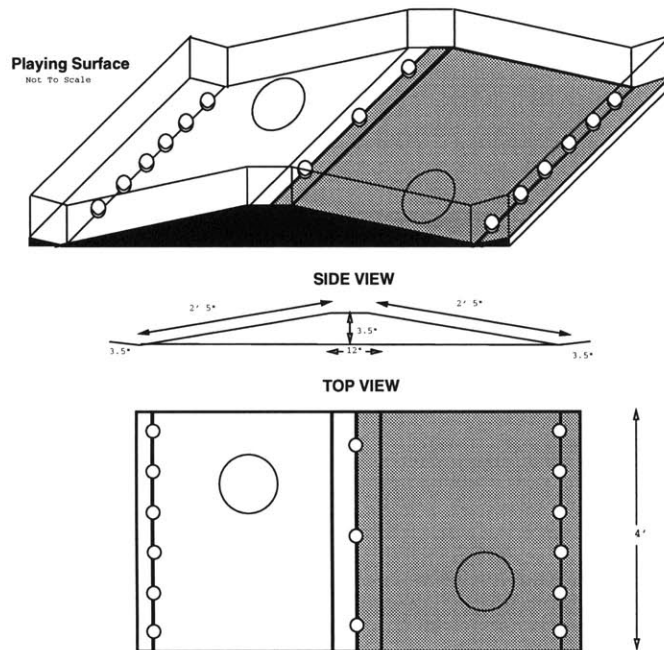
- The winner(s) will be the last machine to come in contact with the puck at the end of a given period.
- Contact with the puck is as follows: The microprocessor board must touch the puck through LEGO. Therefore if the machine is touched by a piece of lego that is connected to the rest of the machine by a piece of string, it is not counted as a legal contact.
- Legal contact of the puck is contacting the puck only (touching the electronics or the beacon is not a legal contact).
- The judges will decide any discrepancies in the period.
- One point will be awarded to the losers of the period.
- Entries will compete only against other entries with the same score. ie. Machines with 1 point will compete against other machines with one point, and undefeated machines will compete against other undefeated machines.
- Every round, a person with two points will be eliminated.
- In a round containing a placebo, the contact made by a placebo will not count. Therefore the last machine to make contact with the puck will be determined from the other two machines in the round.

Final arbitration of any rule disputes before the day of the contest (Feb 7) will be decided by the Contest Organizers — Fred Martin, Pankaj Oberoi, and Randy Sargent.

All machines that appear to be a safety hazards will be disqualified from the competition.

Contestants may approach the organizers listed above in privacy with questions about possible design that may be questioned under the existing rules. The designs will not be divulged to any of the other contestants.

## 6.270: The LEGO Robot Design Competition "ROBOPONG"



**OBJECT:** At the end of a 60 second round have fewer balls on your side than your opponent has on his side.

- Contestants begin in diagonally opposite circles marked on the table.
- Two Contestants and 15 balls
- 6 balls on each side of the table and 3 balls in the middle plateau
- 4.5 inch high clear plexiglass rim surrounding the playing surface
- The playing surface will be divided into a dark region and light region
- The playing surface may not be permanently altered or destroyed
- All evidence of an entry must be removed within 30 seconds after the end of the round
- Robots may not exceed a 1'x1'x1' Max at the beginning of the round

Figure B-2: *Robo-Pong* table specifications

## B.3 Robo-Pong, 1991

### Object

To have fewer balls on your robot's side of the table at the end of a 60 second round.

### Balls

- The balls will be practice golf balls. These are similar in weight to ping pong balls, but are slightly larger and have less bounce. They will be painted.
- There will be 15 balls on the playing surface at the start of a round which will start at predetermined locations. Three balls will start at the top plateau. Each side of the table will have 6 evenly spaced balls in the player's flat area. There will be small depressions in the playing surface to hold the balls at the beginning of a round.

- The balls are inert and all have identical properties.
- Robots may gather balls “into” their body.
- The balls may not be altered or destroyed in any way.
- If a ball is removed from the playing space, then the point at which it leaves the space (crosses the rim) will determine its permanent position on the playing field. E.g.: if your robot pushes a ball off the opponent’s side of the table, it counts as being permanently on that side of the table (for that round).

### **Period of Play**

- The powered portion of a round will last 60 seconds: After the machines are started, they will have 60 seconds to apply battery power to their actuators (defined as motors and solenoids).
- The round ends when all machines and balls come to rest.
- The round will be started by the judges turning on the starting lights, located underneath the table in the center of each robot’s starting circle, for the beginning 5 seconds.
- The contestants will place the machines on the playing field at a random angle with respect to the center per judges’ instructions. The angle at which the machines will be placed will be the same for all machines during the same round, but will vary between rounds.
- The contestants will have 30 seconds to place their machines on the field from the time the judges call them.
- The contestants must stand a given distance away from the playing field. Any contestant that makes an attempt to touch their machine during the round of play will automatically be disqualified from the round.
- The machines must have their own internal clock (software will be provided to do this) which cuts off power to the motors at the end of 60 seconds. Any machines that continue to supply actuator power after 60 seconds will be disqualified.
- The contest will be an double elimination competition held over two nights. Machines must qualify for the second night of competition, as follows:
  1. **Night 1.** All machines will play one round. If a machine loses its round against an opponent, it will run against an inert placebo. If it cannot win against the inert placebo after two tries, it will not qualify for the second night of play.
  2. **Night 2.** The main competition. Machines will play until they lose twice. Loss against opponent from the first night is included.
- All rounds will have two robot players.



- Machines are not allowed to destroy their opponent's microprocessor board.
- Machines cannot try to destroy other machines' broadcast or detection beacons.

### **Control**

- All entries must be solely controlled by their onboard computer. There can be no human intervention once the round begins.
- All entries must be capable of broadcasting infrared (IR) light at two specified frequencies (to be determined). This IR light beacon will be used to detect other machines. Machines failing to meet this specification, or in any way modifying their transmission frequency during the round of play, will be disqualified.
- Judges will assign frequencies for IR emitters to the machines in the beginning of each round.
- After the start of the contest, there can be no change to the robot's program or configuration switches made by the contestants.
- Parts that are likely to damage the internals of machines (ie. small pieces of wire, barb, or fluids) may not be dumped on the playing surface.
- Any LEGO parts that are deliberately dumped on the playing surface (e.g., as an obstacle) will become the property of the 6.270 course.

### **Structure**

- All entrants will be given an equal amount of LEGO and other supplies including:
  1. An ample supply of LEGO, including beams, axles, gears, wheels, and other parts from the LEGO Technics line.
  2. Electronic hardware and plans for building a robot controller board, which can be programmed through any standard serial line.
  3. Motors.
  4. Sensors, including but not limited to IR detectors, touch sensors, photocells, and level-sensing mercury switches.
  5. String, rubberbands, wire, solder, and glue.

The package will be valued at approximately \$500.

- The IR broadcasting beacon must be located at exactly one foot (12 inches) above the surface of the playing field when mounted on the robot. The beacon must be mounted on the largest (dimensionally) part of the machine that traverses across the playing field (if the robot traverses at all).

- Only LEGO parts and connectors may be used as robot structure. LEGO rubber bands are counted as LEGO parts; therefore, LEGO rubber bands *may* be used to provide structural support to your machine.
- LEGO pieces may not be glued together.
- LEGO pieces may not be altered in any way, with the following exceptions:
  1. The grey LEGO baseplate may be altered freely.
  2. LEGO pieces may be modified to facilitate the mounting of sensors and actuators.
  3. LEGO pieces may be modified to perform a function directly related to the operation of a sensor.
- A non-LEGO part may be attached to at most 5 LEGO parts via glue.
- Wire may only be used for electrical purposes, and not structural.
- Rubber bands may be glued to LEGO wheels or gears to increase the coefficient of friction.
- Only the LEGO rubber bands and thin rubber bands may be used to provide stored energy.
- Contestants may not alter the structure of their entry once the contest has begun, but may repair broken components between rounds if time permits.
- The dimension of the machine may not exceed an imaginary 1 foot cube at the start of each round. The IR transmitting and receiving beacons are exempted from this rule. Entries may however expand once the round has begun.
- Entries may not drag wires between two or more structurally separate parts of their robot, unless those wires are part of a LEGO chain link between the various parts of the robot.
- Entries must be built completely from kit parts, with the following exception: Contestants may spend up to \$10 for the purchase of up to 10 electronic components used in their design. No single part may cost more than \$2. Contestants must show receipts upon request.
- No lubricants may be used, at all.
- For orienting the machines, an arrow pointing to the “front” of the machine must be placed at a visible location for the judges.

### **Scoring**

- The winner(s) will be the machine with fewer balls on its side at the end of the round.

- A clear division between the two sides will be noted by having the surface of the two sides be painted in contrasting colors. Robots will be “told” which side they are starting on by the setting a DIP switch before the round begins. Dynamically, robots will be able to determine which side they are on by using reflectivity sensors aimed toward the playing surface.
- In rounds containing a placebo, the contestants’ robot must push at least one ball over to the placebo’s side in order to be declared the winner of that round.
- The judges will decide any discrepancies in the contest play.
- Final arbitration of any rule disputes before the day of the contest (February 5th) will be decided by the contest organizers—Fred Martin, Pankaj Oberoi, and Randy Sargent. All machines that appear to be a safety hazards will be disqualified from the competition. Contestants may approach the organizers in privacy to consult about possible designs that may be questionable under the rules listed above. These designs will not be divulged to any of the other contestants.

## **B.4 Robo-Cup, 1992**

This year's contest is "Robo-Cup Soccer," a game played by two autonomous robots.

### **Object**

Have your robot place more balls into its goal within a 45 second period.

### **Balls**

- The balls will be practice golf balls. These are plastic balls, similar in weight to ping-pong balls, but slightly larger and with less bounce.
- One point will be awarded for each ball in your robot's goal when the contest ends. It does not matter which robot caused a ball to enter your robot's goal.
- The balls will be dispensed six inches away from the wall at designated drop points. The balls will be dispensed when the corresponding touch sensor has been activated. The dispenser will allow one ball to be dispensed once every five seconds.
- You may place one ball into your robot before the start of the contest round.
- The balls are inert and all have identical properties.
- Robots may gather balls "into" their body.
- The balls may not be altered or destroyed in any way.

### **The Goal**

- Your robot's goal is defined as the goal which is furthest from the robot's starting position.
- The goal is one foot wide and eight inches high with a barrier post that is one inch wide at a height of three inches above the surface.  
See Figure B-4 for an illustration of the goal.
- Your robot may not advertently place anything inside either of the goals except the balls.
- The surface inside of the goal will be sloped so that the balls will roll into the goal when they cross the goal line.

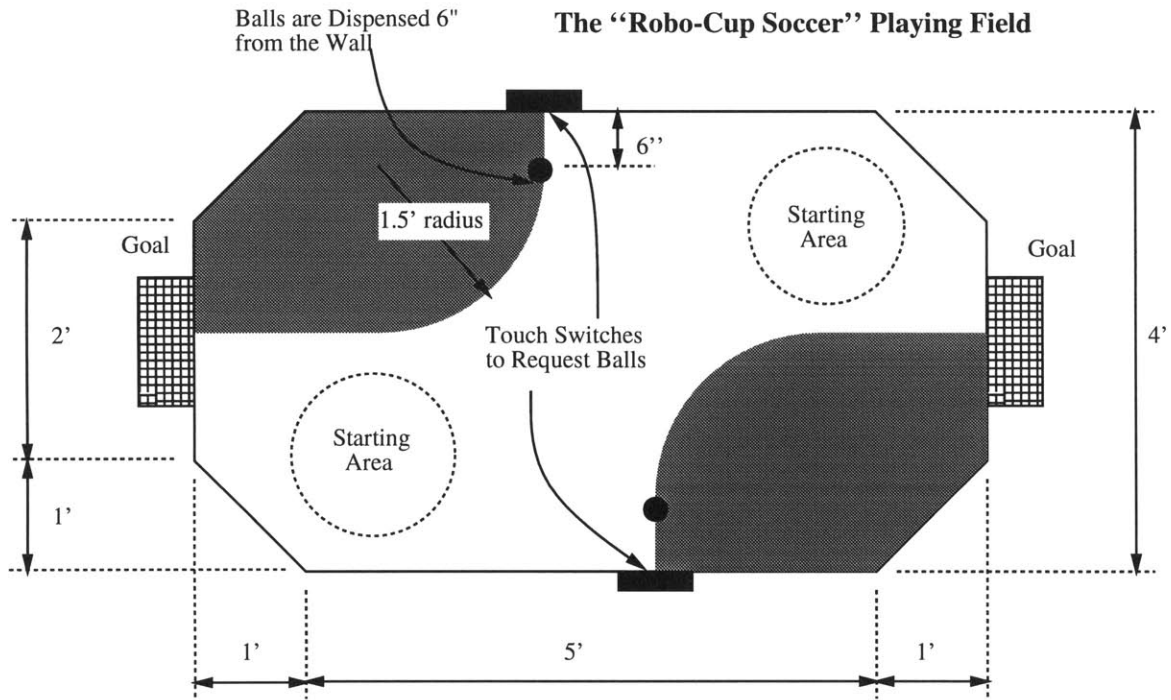


Figure B-3: *Robo-Cup* contest playing field specification

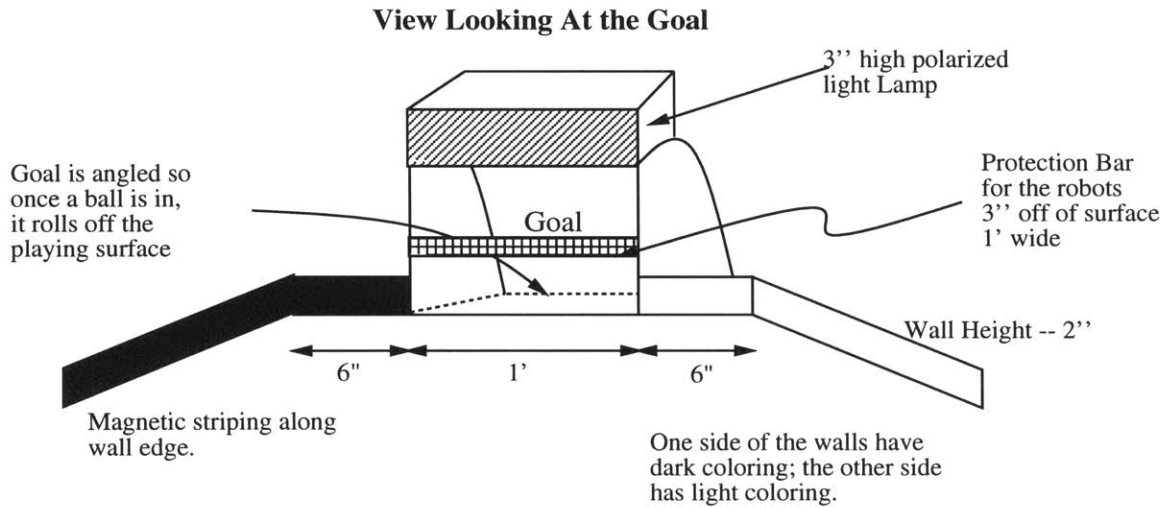


Figure B-4: *Robo-Cup* contest goal specification

## Period of Play

- The powered portion of a round will last 45 seconds. After the machines are started, they will have 45 seconds to apply battery power to their actuators.
- The round ends when all machines and balls come to rest.
- The round will be started by the judges turning on the starting lights, located underneath the table in the center of each robot's starting circle, for the first one second of the round.
- The contestants will place the machines on the playing field within the designated starting circles. The robots may be placed in any orientation within the starting area.
- The contestants will have 30 seconds to place their machines on the field from the time the judges call them.
- The contestants must stand a given distance away from the playing field. Any contestant who touches their machine during the round of play will automatically be disqualified their robot from the round.
- The machines must have their own internal clock (software will be provided to do this) that cuts off power to the motors at the end of 45 seconds. Any machines that continue to supply actuator power after 45 seconds will be disqualified.
- The contest will be an double elimination competition held over two nights. Machines must qualify for the second night of competition, as follows:
  - *Night 1.* All machines will play one round. If a machine loses its round against an opponent, it will run against an inert placebo. If it cannot win against the inert placebo after two tries, it will not qualify for the second night of play.
  - *Night 2.* The main competition. Machines will play until they lose twice. Loss against opponent from the first night is included.
- All rounds will have two robot players.

## Control

- All entries must be solely controlled by their onboard computer. There can be no human intervention once the round begins.
- After the start of the contest, there can be no change to the robot's program or configuration switches made by the contestants.
- No parts or substances may be deliberately dumped, deposited, or otherwise left to remain on the playing surface. A machine that appears to have be designed to perform such a function will be disqualified.
- Machines are not allowed to destroy their opponent's microprocessor board.

- Machines cannot try to destroy other machines' broadcast or detection beacons.

## **Infrared Beacon**

All robots are required to carry an infrared transmitter. This transmitters acts as a beacon so that robots can locate each other on the playing field. The following rules describe the functionality of the infrared beacon.

- All entries must carry an infrared beacon that is capable of broadcasting infrared (IR) light at modulated at either 100 Hertz or 125 Hertz with a 40,000 Hertz carrier (hardware is provided to do this).
- Machines failing to meet the infrared transmission specification, or in any way modifying or jamming their transmission frequency during the round of play will be disqualified.
- Judges will assign frequencies for IR emitters to the machines in the beginning of each round by setting the robot's DIP switch 1. If the switch is one, the robot must broadcast 100 Hertz infrared light. If the switch is zero, the robot must broadcast 125 Hertz infrared light. Software will be provided to do this.
- The IR broadcasting beacon must be located at exactly one foot (12 inches) above the surface of the playing field when mounted on the robot.
- The beacon must be located so that its center is never more than four inches (measured horizontally) from the geometric center of the microprocessor board.
- The beacon cannot be deliberately obstructed, or be designed in such a way that "accidental" obstructions are probable.

## **The Contest Playing Table**

Figure B-3 illustrates the Robo-Cup playing field.

### **Polarized Light Goal Lamps**

- Goals will have panels that emit polarized light at one of two orientations: a +45 degree and -45 degree (with respect to the vertical) polarization (see Figure B-4).
- The setting of DIP switch 1 will indicate which polarization angle is emitted by the robot's goal. If the switch is one, the robot's goal will be the one with the positive polarization. If the switch is zero, the robot's goal will be the one with negative polarization.

## Wall Striping

- The two sides of the playing field will be coded with light or dark visible striping on the lower inside edge of the playing field wall.
- All wall edges will also be coded with magnetic strips.

## Ball Dispensers

- The ball dispensers will drop balls at a distance of 15 inches above the designated drop points.
- Robots may obtain balls from either ball dispenser.
- Ball dispensers will dispense balls at a rate not greater than one ball per five seconds.
- Depressing the touch bumper near to the ball dispenser causes a ball to be dispensed as soon as is possible given the dispensing rate mentioned above.
- The touch bumper must be released before a subsequent ball can be dispensed.

## Structure

- All kits contain exactly the same components, with the exception that some LEGO parts may be colored differently in different kits.
- Only LEGO parts and connectors may be used as robot structure. LEGO rubber bands are counted as LEGO parts; therefore, LEGO rubber bands *may* be used to provide structural support to your machine.
- LEGO pieces may not be glued together.
- LEGO pieces may not be altered in any way, with the following exceptions:
  1. The grey LEGO baseplate may be altered freely.
  2. LEGO pieces may be modified to facilitate the mounting of sensors and actuators.
  3. LEGO pieces may be modified to perform a function directly related to the operation of a sensor. An example: Holes may be drilled into a LEGO wheel to help make an optical shaft encoder.
- String may not be used for structural purposes.
- The wooden dowel may be used only as a tower to mount the infrared transmitters and any receivers.
- A non-LEGO part may be attached to at most five LEGO parts via glue.



- Cardboard, other paper products, and tape may be used for the purpose of creating optical shields for light sensors.
- Wire may only be used for electrical purposes, and not structural.
- Rubber bands may be glued to LEGO wheels or gears to increase the coefficient of friction.
- Only the LEGO rubber bands and thin rubber bands may be used to provide stored energy.
- Contestants may not alter the structure of their entry once the contest has begun, but may repair broken components between rounds if time permits.
- The dimension of the machine may not exceed an imaginary 1 foot cube at the start of each round. Only the IR transmitting and receiving beacons and the bend sensors may protrude outside this volume. Entries may however expand once the round has begun.
- Entries may not drag wires between two or more structurally separate parts of their robot.

One portion of the robot is considered structurally separate from another if when the machine is lifted from a supporting surface and held from the other portion, the two portions are supported mainly by wire.

- No lubricants may be used.
- Cable ties may not be used for structural purposes.
- Some parts in the 6.270 kit are considered tools and may not be used on the robot. Examples are: the red plastic parts container; the small rectangular parts container; the soldering iron sponge. If there is any question about whether an object is a “kit part” or a “tool part,” ask the organizers.
- Any machine that appears to be a safety hazard will be disqualified from the competition.

## **The \$10 Electronics Rule**

To encourage creativity, contestants may spend up to \$10 of their own funds for the purchase of additional electronic components used in their design. Other than this rule, robots must be designed completely from standard kit parts. The following conditions apply to all non-kit-standard electronic additions:

- The following components, categories of components, or varieties of circuitry are *disallowed*:
  - Batteries of any variety.

- Motor driver circuitry, including relays, power transistors, or any other replacements or modifications to the standard motor driver circuitry.
- No single part may cost more than \$2.
- Resistors rated less than 1 watt and capacitors valued less than 100  $\mu\text{F}$  may be used freely, without accounting toward the \$10 total.
- Contestants who add *any* non-kit parts to their project must turn in a design report that includes: description of the modification, schematic of all added circuitry, and store receipts for parts purchases. This design report must be turned in to the organizers by 5:00 pm, Friday, January 31, 1992. *Any machines found with added circuitry that has not been documented in this fashion will be disqualified.*
- If a contestant wishes to use an electronic part which has been obtained through other means than retail purchase, an equivalent cost value to the part will be assigned by the organizers. Contestants must obtain this cost estimate *in writing* from the organizers and include it in the design report mentioned above.

## Scoring

- Each ball entering the upper goal area will be awarded three points. Each ball entering the lower goal area will be awarded two points. The winner will be the machine with points at the end of the round.
- In rounds containing a placebo, the contestant's robot must score at least one ball into its goal in order to be declared the winner of that round.
- If no goals are scored a double loss will be awarded by the judges.
- If there is a tie at the end of the round, the win will be determined as the robot that has more balls on the half of the playing field nearer to its goal.
- A double win may be awarded at the judges' discretion.
- The judges will decide any discrepancies in the contest play.

## Organizers

- Contestants may approach the organizers in privacy to consult about possible designs that may be questionable under the rules listed above. These designs will not be divulged to any of the other contestants.
- Final arbitration of any rule disputes before the day of the contest (February 3rd) will be decided by the contest organizers—Fred Martin, Pankaj Oberoi, and Randy Sargent.

# Appendix C

## Administrative Considerations

This appendix is included for readers who may be considering setting up their own versions of the design course presented in this dissertation. I discuss some of the concerns that we faced when we pursued the possibility of offering formal course credit to students who participated in the Robot Design project. While we wanted to offer students formal recognition of the work they were doing in the project, we did not want it to adversely affect the quality of their participation.

Additionally, the issues of kit ownership, student costs, lotteries, and participation by non-MIT members of the academic community are discussed.

### C.1 Granting Academic Credit

When the Robot Design project was first getting started, participation was on a non-credit basis, as was the case with most activities in MIT's Independent Activities Period (IAP), the academic session in which the Robot Design project existed. In *Robo-Pong*, participants in the Robot Design project had the option of registering for three units of MIT credit (full term MIT classes generally are worth nine or twelve units). Beginning with *Robo-Cup*, we negotiated six units of credit. This credit was offered on pass/no-grade basis, meaning that students would either receive an ungraded mark of "pass" if they successfully completed the course, and there would be no record of their having registered if for some reason there was a problem meeting the course requirements, or if they simply changed their minds.

Simply put, the rationale for awarding credit for participation in the Robot Design project was that students were doing a lot of work and they may as well get credit for it. Two faculty members of the Electrical Engineering and Computer Science department, Professor Leonard Gould and Professor and Department Head Paul Penfield, evaluated our description the activities performed by students in the class to determine they were worthy of academic credit, decided that they were, and then worked with us to develop evaluation criteria to make sure that credit was fairly awarded.

We did not want to subject students to the performance pressures of the traditional academic term; to the contrary, we wanted to ensure that there wouldn't be any additional pressure on students to get their robots working other than that which they put on themselves. Since the credit was ungraded, the idea of "working for an A" did not exist; we simply had to determine that students had made a reasonable effort and they would get credit. Since students could bail out at any time without penalty, students would not have to make a commitment to something before they knew they were interested in it.

This matter of academic credit was an issue in which the three of us who were organizers had significant differences of opinion. In my interpretation, Pankaj Oberoi was the most "establishment oriented" of the three of us, seeing the least difficulty with offering credit, while Randy Sargent was the most "anti-establishment." While Sargent understood the value of the MIT administration recognizing the project as a credit-worthy activity, he also saw most strongly the potential for harm if students were to feel unnecessary pressure because of it. We all knew that we would have to safeguard against a student who tried to take a "free ride" with their teammates, but Sargent and I were additionally concerned with how strongly we should attach the awarding of academic credit to general participation in the project. For example, should students be *required* to take the course for credit, or should it be an option? Sargent and I would not have been supportive if we had decided on mandatory credit, but even after we agreed that credit should be an optional feature of participation, the question remained of how actively we should promote it. I thought that as long as we made clear to participants that (1) credit was optional and (2) they could remove themselves from the credit list at any time without penalty, it was appropriate to advertise that credit was available, since it would make the project's standing in the eyes of

the faculty more solid. Sargent agreed to this, but I believe he continued to be concerned about more subtle ways that credit would affect students' attitudes and participation in the course.

As it turned out, the necessities of tracking students individually for purposes of ensuring that credit was awarded fairly dovetailed nicely with my interests in taking data on students' work for the purpose of this doctoral research.

In developing the requirements for credit registrants, we wanted to achieve a fair balance between the amount of information collected and the additional burden it would place on the participants. If something were to require an inordinate amount of time from the students to provide to us, then we would not make it a requirement; we wanted the additional work required to receive credit to be as little as possible.

After discussion, we agreed to the following requirements:

**Weekly Written Reports.** One of our concerns was having some basis for distinguishing who was doing what on a given team of robot builders. By asking students to prepare a series of three written reports, which focused on their own thinking and work on the design project, we would have an objective way of ensuring that each individual on a team was doing his or her appropriate share.

We offered students the option of keeping a daily journal log of their work, which they could photocopy and submit in lieu of a prepared report. Only a few students chose this option; we didn't want to make the journal format mandatory in that we did not believe it was a preferred working style for the vast majority of the class.

**Weekly Video Reports.** As an additional vehicle for collecting data, we set up a video camera in a spare room of the electronics lab and required that students make a weekly presentation in front of the video camera. The purpose here was to capture a dynamic expression of team interactions that wouldn't be possible with the written reports; all members on a given team that were registered for credit were required to participate.<sup>1</sup>

These two items were the primary means by which we determined that students were in fact doing work—at least for the ones that we hadn't gotten to know simply by the extent

---

<sup>1</sup>This concept was originally suggested by Mitchel Resnick.

and visibility of their general participation. We also established the following two criteria, which did not require any additional work on the students' part:

**Completed Robot.** We figured it was reasonable to ask that teams registered for credit complete their design project to the extent that they produced a tangible artifact which they could point to and say, "This is the robot." We did not require that it solve the contest in order for students to receive credit.

**Program Listing.** As a record of their work, students were required to submit copies of the program they had written to control their robot. (One of the wonderful things about software is the simplicity by which copies of the work may be made; I wish it might be so easy to ask for so complete a recording of the robot design itself!)

Even as we developed a structure to allow students to receive academic credit for participation, we were extremely careful to make sure their motivations for their participation weren't affected by traditional academic ambitions. For this reason we did not wish to give out letter grades; we also wanted to make it the case that a variety of styles of participation would still be encouraged under the evaluation criteria for awarding credit.

## C.2 Student Costs

From the start of the project, we were concerned that financial considerations should not cause any student to hesitate to participate in our project. The first year that hardware was involved, we chose to collect \$50 from each team to defray the costs of buying the parts that would comprise the robot-building kit. We estimated that this figure would be small enough, when split among the two to four team members, to be negligible in any student's financial planning. That year Microsoft Corp. was the patron sponsor of our project, providing over \$1000 of cash support.

In the subsequent year we attempted to raise cash and part donations from numerous corporations in our commitment to keep student costs to a minimum. We also solicited substantial cash support from MIT's Electrical Engineering and Computer Science (EECS) department. As it turned out, Motorola Semiconductor donated expensive components

that were instrumental in the 1990 project year, Microsoft continued as a substantial cash sponsor, and MIT became the major underwriter of the project costs.

Over the subsequent years of the contest we were fortunate enough have the continued commitment of our initial sponsors while recruiting about a dozen others (who contributed primarily through part donations). We also had the support of Professor Gould in our commitment to keep student costs down; the kit fee has remained at \$50 per team since the start of the project.

### **C.3 Kit Ownership**

The fact that students kept their kit at the end of the course had a positive motivational effect. One of MIT's other project-oriented courses, *Digital Design Laboratory*, requires that students disassemble their projects essentially immediately after they have finally demonstrated them to work. After spending six weeks on a project and a final intensive period of several days, it is a painful event to have to rip the wires out of one's project in a furious burst just after the project's completed. By allowing students to keep their robots, we encouraged them to work further on them as time allowed. Each year a number of students got their robots in functioning order for a re-match contest that we held a few months later at Boston's Computer Museum.

The low entry fee combined with the kit ownership policy caused an unforeseen complication, however, as the degree of subsidy increased each year. By the 1991 project year, we were providing students with a robot-building kit that had a retail value of about \$500. The question of who keeps the robot or how to split up the kit after the class became nettlesome in a number of cases.

Sometimes these disputes took on a sad although comical dimension, resembling bitter child custody battles that ensue when marriages fail. One student pays for the whole kit, but becomes less involved in the project than his partner. Both feel they have a valid claim to keeping the kit afterward. The student who paid the course entry fee believes he bought the kit; the other argues that the entry fee is only a fraction of the kit value, so it should not be the determining factor. After witnessing a number of these situations occur, we made a

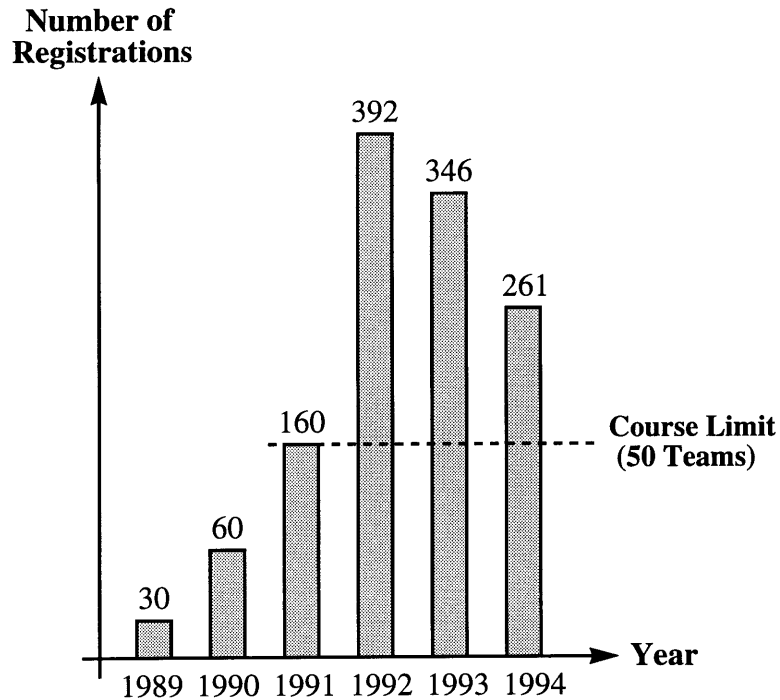


Figure C-1: Number of registrants by year, 1989 through 1994

point of having students talk the issue through before the class got underway, and come to some kind of agreement about how the matter would be resolved beforehand. This seemed to ameliorate the problem.

## C.4 Lotteries

Another problematic matter arose from the success of the class, beginning in the 1991 contest year, when more students signed up for the class than resources would permit. For several years, the registration for the class was on an exponential growth curve: 30 students in 1989, 60 in 1990, 160 in 1991, and 392 in 1992 (see Figure C-1). (Since the peak in 1992, the registration count fell to 346 in 1993 and 261 in 1994.) For the first three of these years, we expanded the class to accept all interested persons. But after the 1991 year, we knew that we did not have the resources to support more than fifty teams of students. It wasn't a question of monetary resources, but that of personnel to run the class; we felt stretched very thin by the 160 student class.



After 1991, when we were able to accept all of the registered students, we knew we would have to come up with a mechanism for dealing with oversubscription. We determined that a simple lottery was the most appropriate way to deal with the matter. This would avoid having to decide on individuals on a case-by-case basis and subject our project to the unseemly appearance of letting people into the class because of personal relationships.

In the years of 1992 and 1993, all registrants had equal weight in the lottery. The organizers in 1994 decided that it would be appropriate to give students who had lost previous lotteries a higher chance of getting in, and a complex weighting system was established in which a person's chance of getting into the class depended on the weighted average of previous lottery losses across a team entry. Still, there were cases in which persons who were graduating seniors and had lost prior lotteries were denied admission. It is clear that the lottery system is not ideal, though there continues to be general agreement amongst the organizers that it is better than trying to resolve cases individually.

## **C.5 Non-MIT Participation**

By the 1993 contest year, knowledge and excitement about the Robot Design project had spread to students at Harvard University and Wellesley College. A number of students from each of these institutions inquired as to whether they might register for the class.

The organizers at the time made the decision that non-MIT students could register, but they would only be accepted into the class if there was not enough demand from the MIT community. Effectively, this was equivalent to denying them admission; there was and still is no foreseeable end to the oversubscription problem.

I was no longer an active organizer of the class at this time and had little effect on the discussion of this matter. While I can see the reasoning behind the organizers' decision, I believe it was parochial. It is painful to deny any person admission to the class, and it would seem unfair to allow a non-MIT student admission when so many are denied it. Harvard and Wellesley, however, are institutions that enjoy a particularly close relationship with MIT. Students from all three institutions take classes at the others. It would have been more farsighted to allocate one to two team slots for students from these schools.

# Appendix D

## Technology Development

This appendix chronicles the development of the technology for the Robot Design project, providing additional technical details and conceptual motivations behind the three stages of technology that were developed: the Remote Controller (1989), the Assembly Language Controller (1990), and the C Language Controller (1991 to present).

### D.1 The Remote Controller

In the first year that we used real electronics in the course, parts cost and development time were extreme constraints. We were aiming for materials for ten to twelve teams of participants that should cost between one and two thousand dollars total (we had gotten a commitment of \$1000 as a sponsoring donation from the Microsoft Corporation, and we anticipated raising additional money through MIT sources and registration fees from students taking the class). We had just about one month to design a system and procure ten to twelve sets of materials for the preregistered participants.

We first agreed on what was a necessary core of a hands-on robotics experience. We determined that it would have to include sensors, motors, and programming. So our minimal system would have to allow project participants to build a machine that incorporated motor control, sensor input, and programming to tie it together.

Randy Sargent hit upon the idea of using an inexpensive communications chip (a universal asynchronous receiver/transmitter, or UART). Using the UART chip, we designed

a hand-wired board that added a couple of motor driver chips I was familiar with, having used them in the Programmable Brick. Our solution thus sacrificed on-board computation, allowing us to forgo implementing a full-blown microprocessor design and use the computational power of desktop machines instead. This approach afforded us a solution that was feasible within the extreme cost and time constraints that were present.

### **D.1.1 Hardware**

Since little could be done without it, the first task we gave the participants of our project was the one of building their own copy of the controller board. Each board had to be individually hand-wired on blank perfboard panels; we created a chip layout diagram and wire list so that each team (we organized the participants into teams of one to three persons) would build the same circuit. Many of the participants had never soldered before, making the board assembly process fairly difficult. Our troubles in assisting the students were compounded when we realized that about 20% of the blank perfboards were numbered with slight differences from the one we had used to formulate the wirelist.

Figure D-1 shows a photograph of the completed board, along with a couple of accessory boards. The D-shaped connector along one edge of the board provided the connection to the off-board computer and power supply. The board was in principle capable of driving up to four motors independently, but the power supplies that we purchased were underpowered for dealing with the motors we had acquired, making sufficient power available for driving only two motors simultaneously.

For sensors, the board was able to accept data from up to eight digital (“on” or “off”) sensors. The primary sensor to be used in the contest would be a rolling ball switch device that would allow robots to detect their inclination on the playing surface. This sensor was easy to interface to the board, though other unforeseen complications arose with its use (vibrations made it difficult to obtain readings from the device).

Though it was not strictly necessary in solving the contest challenge, we hoped that the participants in the class would want to use analog light sensors. We anticipated two possible uses for these. The first was in detecting the light bulb that was embedded in the peak of the “mountain”—a convenient way for the robot to determine that it had reached

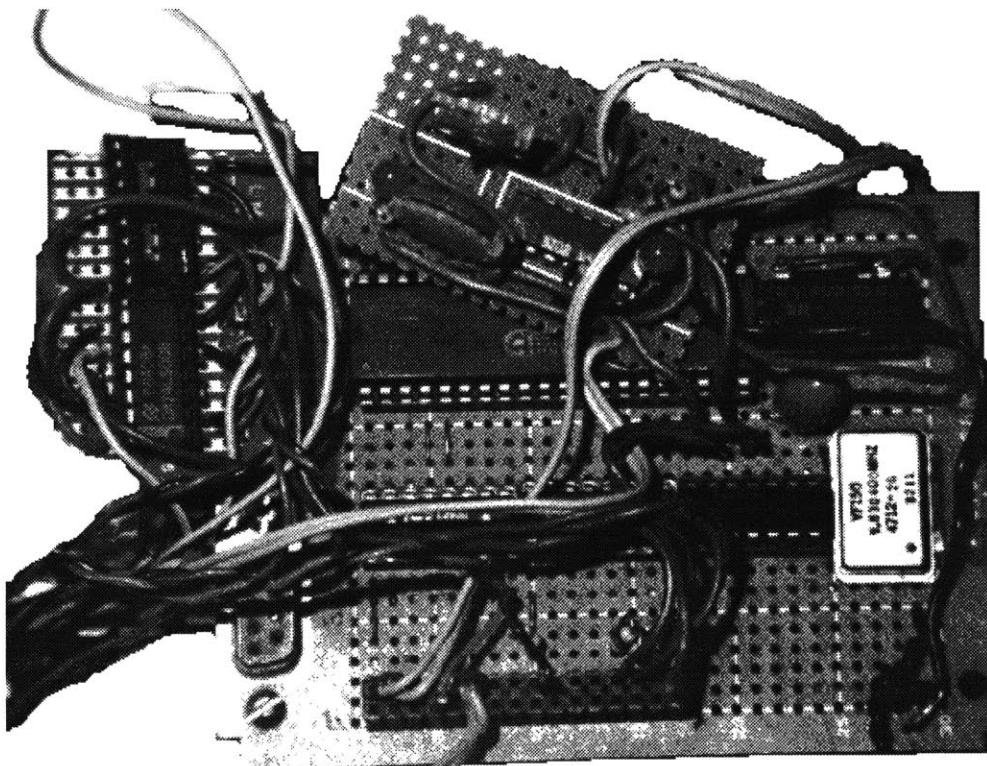


Figure D-1: The 1989 Remote Controller Board

the top of the hill. The other possible use was in detecting the two opponent robots: it was mandated that robots carry illuminated flashlight bulbs, so that other robots might be able to detect their light and thereby their presence, and then react in some useful way.

To facilitate the use of analog light sensors, we provided the participants with the parts and a diagram for building a circuit that would convert the analog signal provided by the light sensors into a digital pulse train that could be accepted by the interface board. The use of this circuit would require special software to time the length of the pulse, which would correspond to the value of the analog sensor (see Figure D-2).

It turned out that not one group of students used the analog sensing capability, though several did experiment with it. Getting basic functionality from their robots (i.e., a robot that could climb the hill) was so fraught with problems that there was no time to add the light sensors to their robots. It was sufficiently difficult that there was no point in doing so.

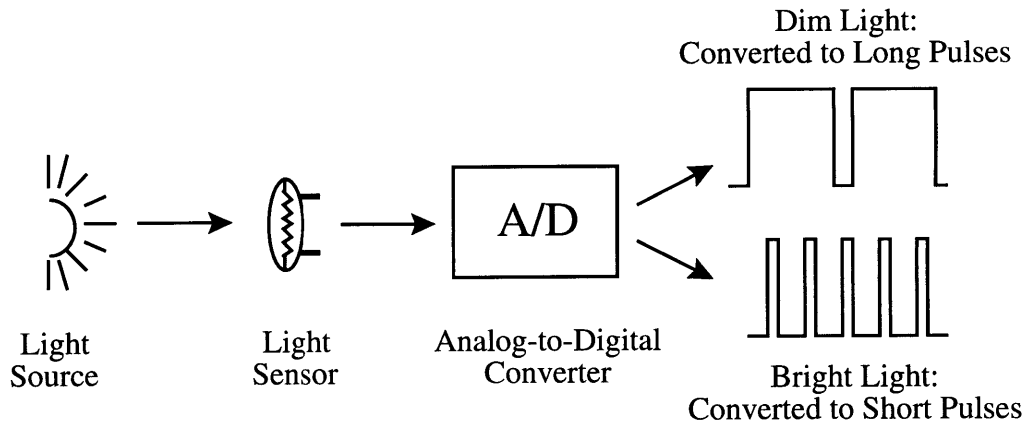


Figure D-2: Converting analog light reading to digital pulse stream

### D.1.2 Software

The programming environment that went along with the Serial Line Controller Board consisted of a diverse set of options. We planned on obtaining permission to use campus-wide Project Athena workstations for programming, but had failed to do so.<sup>1</sup> We were thus forced to cobble together a collection of personal computers, consisting mostly of IBM-PC compatibles in various configurations—some with hard disks, some without—and a few Macintosh computers which were provided and used by students who personally owned those machines.

We were unable to provide an officially supported programming language either. Students used whatever language with which they were most comfortable with or to which they could most easily gain access. This turned out to be various versions of the C, Pascal, or BASIC programming languages.

Regardless of the language used, we recommended an approach to programming whereby a main “event loop” would be used. Inside this loop, the control program would sample the state of the sensors and choose a desired action. Through repetitive evaluation of the loop, the robot’s behavior would be implemented.

This approach was adequate. The primary challenge that year was simply to get

---

<sup>1</sup>Project Athena administrators were concerned that damage might result from our hardware being connected to the machines’ serial ports. Two years later, we were able to persuade them otherwise, which turned out to be crucial in our plan to provide a consistent software environment for future students.

something working at all. Sophisticated robotic control ideas were not important; basic functionality was. This largely consisted of successfully debugging the controller board and getting communications between the desktop PC and the controller to function properly.

One of the issues we encountered in this first class/contest experience, which would return in future versions, was the interrelatedness of contest and hardware. Our desire and intent was to create and provide interesting materials for the students, but we saw the need to make them easy to use and provide motivation for their use. The failure of the students to use the option for analog circuitry was a case of the technology simply being too difficult to use as well as being insufficiently motivated.

## **D.2 The Assembly Language Controller**

Among the difficulties in using the hand-wired remote controller, the greatest was the fact that it had to be tethered to the desktop computer. Our primary desire in upgrading the hardware used in the project was to create environment in which robots could have on-board control. This was the single most important improvement to the materials that could enhance the excitement of the project and capture the students' imaginations. By providing an autonomous controller, not only would the level of technology would be raised, but the students would become involved in more interesting robotic and design issues.

The design of a controller board for our purposes was made feasible by the availability of a new, highly integrated microprocessor chip that incorporated into a single integrated circuit the functionality of what typically required several chips. This chip, the Motorola 6811, was literally a "single chip computer" that included important features for robotic control applications, such as direct analog input, serial line communications hardware, and an internal programmable memory.<sup>2</sup> Whereas the Programmable Brick design used seven integrated circuits and two printed circuit boards, we realized that we could easily design a far more simple controller using the Motorola 6811 that would be within the budgetary

---

<sup>2</sup>We owe thanks to Henry Q. Minsky of the MIT Artificial Intelligence Laboratory, who helped us get started using the Motorola 6811 by providing us with a prototype board of his own design, development software, and other technical support.

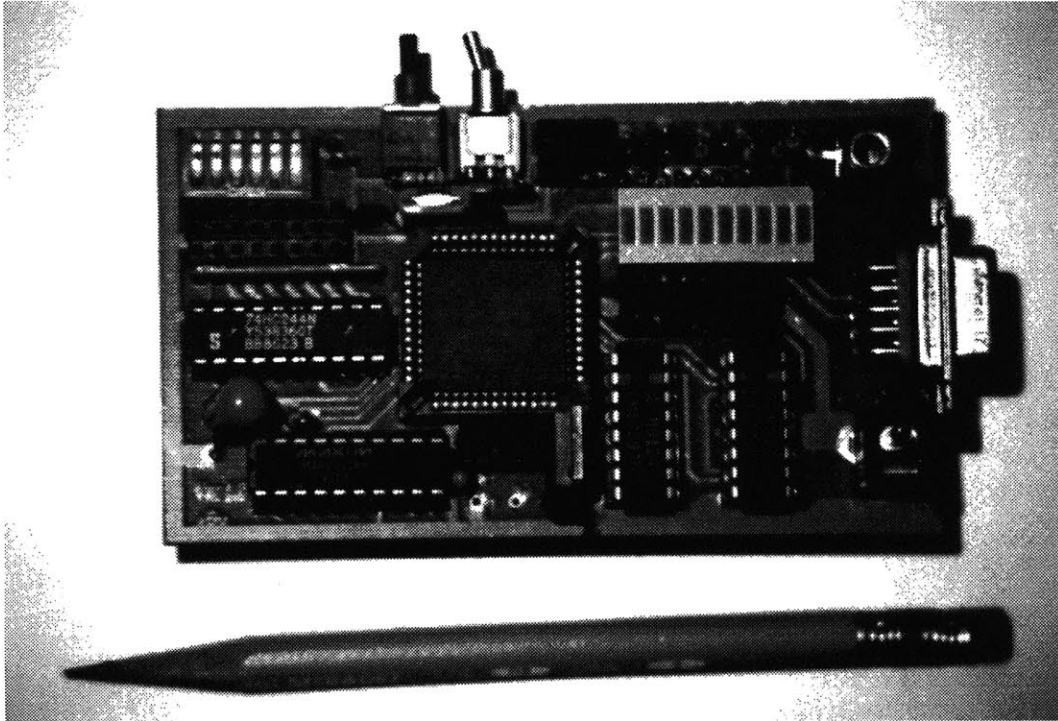


Figure D-3: The 1990 Assembly Language-based Controller Board (actual size)

scope of the Robot Design class.

## D.2.1 Hardware

We attempted to make a controller board that would be as easy to use as possible. Here the experience of having worked on the Programmable Brick controller was valuable.

The LEGO/Logo controller interface had status lamps on the motor outputs; when a motor was being driven, the corresponding status lamp would light. The Programmable Brick did not have these. Having worked with children using both systems, it was apparent that the status indicators were a positive factor in helping users understand the relationship between computer commands (e.g., turning on a motor) and tangible results (e.g., seeing the motor turn on). Also, the status indicators were invaluable when debugging: if the user believed a motor should be turning, a glance at the output lamps would tell whether or not the computer “thought” the motor should be on. It made it easy to trace problems that were due to faulty wiring versus other causes.

So we designed motor status lights into the new board. There were a few other features

that helped make the board easy to use, including a serial connector port compatible with the IBM personal computer standard, and sensor connectors that allowed both digital and analog sensors to use the same connector style. Figure D-3 shows the controller board we designed for *Robo-Puck*, the 1990 version of the Robot Design project.

## D.2.2 Software

For programming the 6811 chip, we obtained the standard software products used by embedded control applications engineers. Using these software tools was a tedious process, even for an experienced programmer. First, the target program—the program to be run on the 6811 microprocessor—would be composed on a text editor. This program was written in assembly language, a code that uses word-like mnemonics to represent the primitive instructions of the computer chip. This program was then translated into a binary object code form with the use of an assembler program; syntax errors and other coding mistakes would generate errors at this point that would need to be resolved. After this sort of error was fixed, the resulting object code file could be downloaded to the microprocessor (using a downloader program). Finally, the target program could be run on the robot.

There were a couple of serious troubles with this sort of system. The first problem was that the system provided little interactivity between the computer and the programmer. For example, there is no way for the computer program to display results for the programmer to examine. The only “results” of program execution that are readily available are the state of the motors, which are often the thing being debugged! Second, it is notoriously easy to write an assembly language program that fails absolutely and without warning, due to the nature of the language itself. When such a program crashes, it is difficult to tell where in the code stream the problem lies.

Traditional assembly language development environments address these problems with the use of what is known as a *breakpoint monitor*. This system allows the programmer to single-step through the program execution and set breakpoints where the program is halted so that the programmer can interact with the microprocessor to examine execution status and display other results. Our controller board, however, did not have enough memory to support the traditional breakpoint monitor. So we were stuck with the primitive assembler



and downloader because there was not time to develop a better environment. We hoped that the students who participated in our workshop would be able to be successful using the system despite its inherent user unfriendliness.

It was nevertheless apparent that some method for easily interacting with the robot's hardware (i.e., motors and sensors) would be valuable. We realized that a limited sort of monitor program, that focused on providing interaction with the motor and sensor features, could be supported by the capabilities of the controller board.

Following up on the idea, we created a monitor program named *RM-11* (Robot Monitor for the 6811). The program allowed users to type simple keyword commands to turn motors on and off and display sensor values. It was feasible in terms of the memory limitations because it did not support the typical monitor functions of single-stepping and breakpointing. Instead, it was customized for the particular hardware of the controller board and provided a straightforward interface for interacting with the special features of our board.

The assembly language programming environment proved to be quite problematic. Most of the workshop participants didn't get to the programming stage of their projects until there were only a few days before the contest. Many of them had not seen their machine's motors actuated by the controller board by this time, and hence their design ideas were largely untested at a point when there was insufficient time to make substantial revisions.

The conceptual overhead involved in doing the most minimal programming task (e.g., proof-of-concept code that turned on a motor based on a sensor reading) was significant: even a trivial task required the understanding of a variety of mundane programming details. The result was that students didn't attempt programming until all other aspects of their robot design had been completed, at least as far as they believed.

Not only was the overhead in learning to use the assembly language environment considerable, but the interactivity in doing programming was minimal. This led most teams to take top-down strategies toward the programming project. They prepared their program conceptually or on paper, wrote the entire program code, and tried to debug it.

This style of programming was quite problematic from a technical standpoint of getting

the thing working. Because of the nature of assembly language programming, errors tend to be of the crash-and-burn variety: the program fails without warning, without providing notice as to where it crashed, how or why. In the case of the robotic hardware being programmed, the problem was made even worse because of the variety and intermingling of possible failure modes, including electrical problems with the computer hardware, software errors, and sensor-related failures.

### D.3 The C Language Controller

In analyzing the shortcomings of the assembly language controller system, we perceived two conceptual areas in which it could be changed:

**Observability.** The assembly language system was difficult to work with and debug because the tools available for observing the state of the system were minimal. In conventional assembly language development environments, the breakpoint monitor is used to observe progress of program execution. This tool is itself difficult to work with and limited, but even it was not available.

In order to make a system that was more user-friendly, that would encourage users to explore, play with, and understand the technology, it would need to provide many more ways of observing the internal state of the robot, including both sensor values and program execution status.

**Level of Abstraction.** The assembly language system did not insulate users from the low-level details of 6811 programming; in fact, it forced them to deal with various details in order to accomplish their programming. This situation could be seen as either a valuable pedagogical feature, encouraging students to learn about the technology in a thorough way, or an unnecessary conceptual burden on the users, who are simply trying to build interesting robots.

We decided that it would be advantageous to give students a higher level interface to their robot design work, shielding them from details of 6811 programming but offering them the possibility to express more complex ideas. This was a difficult decision, as we felt that

the assembly language programming experience could be quite valuable for students, but it was in keeping with the overall philosophy of the project. In giving students a predesigned controller board and predesigned sensors, we were abstracting them away from low-level details of digital and analog electronics; if we provided them with a higher level software interface, it would be continuing an established trend. The value of the giving students a higher level software environment would be evaluated by seeing the sorts of problems in which they became engaged, in comparison to the sorts of problems they encountered in the earlier assembly language environment.

The question of observability was less controversial. It was evident that the assembly language system was quite poor in this regard and providing better tools here would not have any negative consequences. To the contrary, having a system that was more observable would give students a greater sense of power and control over the materials, allowing them to create more complex systems and have greater understanding of their work.

Between the 1990 and 1991 years of the class, Sargent and I performed the development work required to bring about a more sophisticated set of robot development hardware and software. In doing so, we developed a system that had a strong resemblance to the original LEGO/Logo Programmable Brick. The development effort had two parts: the creation of a more powerful controller board and the design of a custom programming environment.

### **D.3.1 Hardware**

As a foundation, we designed a new controller board with significant increases in capability and versatility over the assembly language board design. In order of importance, the changes we made were:

**More Memory.** The most important change was an increase in on-board memory. The assembly language controller was limited to 2K bytes (2048 bytes) of memory for user programs. The new board had 32K bytes of memory. Not only did the increase allow students to build longer, more complex programs, but it was required to support the new programming environment we created.

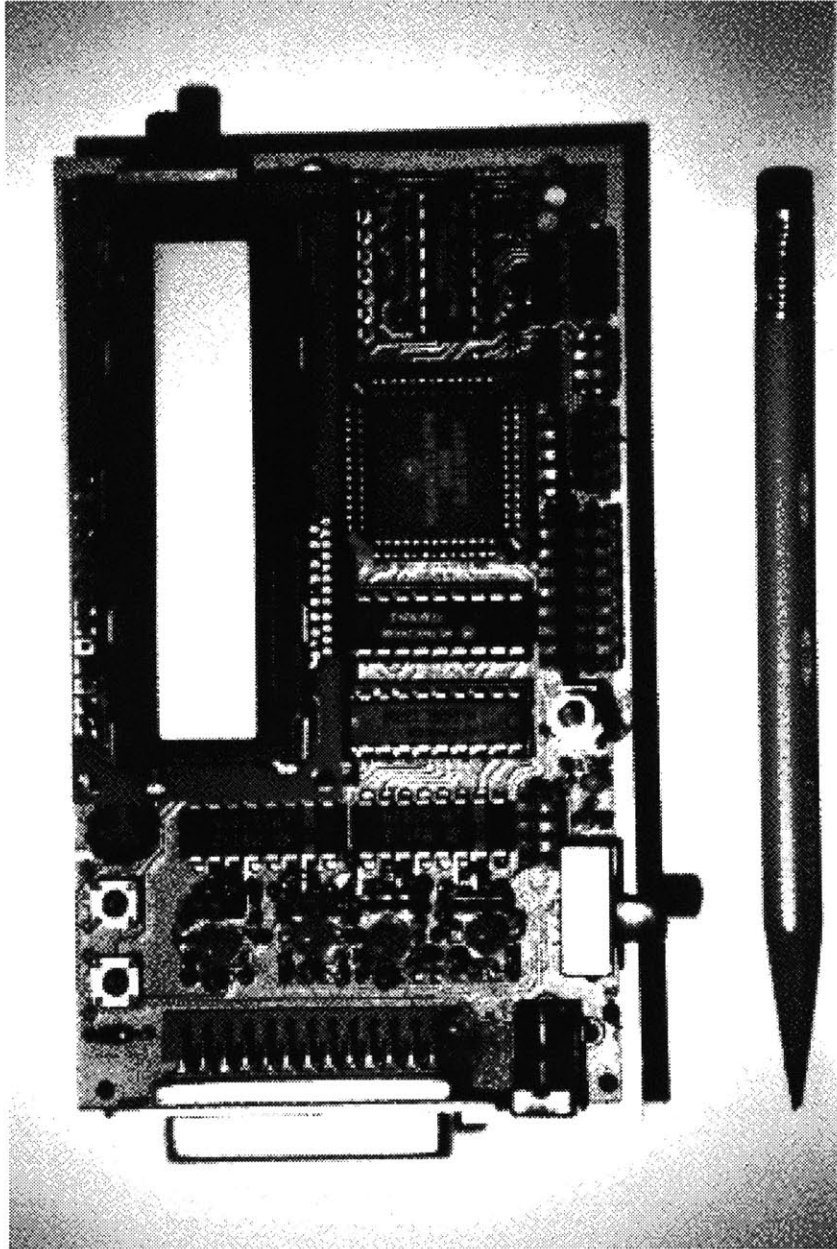


Figure D-4: The 1991 C-language-based Robot Controller Board (actual size)

As with the assembly language board, the memory was *non-volatile*, meaning that the robots would retain their program when switched off. We had found this feature quite valuable; it allowed students to program their robots and test them for extended periods away from a development computer, without needing to constantly reload the program from the computer. This was the feature that was noticeably lacking from the LEGO Programmable Brick.

**Expandability.** The new board was designed for expandability, allowing students to control more motors, receive data from more sensors, and have a general purpose prototyping space for developing their own circuits. A daughter board that plugged on top of the main board provided these additional features.

**LCD Character Display Panel** With the assembly language board, there was no manner for the hardware to provide feedback to the students about its internal state. That is to say, the students could program the board to control their robot's motors, but there was no easy way to provide any status information about the state of program execution (other than by observing the state of the motors, which was often not terribly informative).

The new board supported a 15-character LCD display panel. The software we developed included a programming statement to write data to the display (both text strings and numeric output). This vastly changed the “observability” of program execution: it became easy to add a print statement to a program and thereby monitor its internal status.

Also, it became much easier to experiment with the operation of sensors. Students could write a one-line program to repeatedly print the value of a given sensor to the display. The robot no longer needed to be connected to the desktop development computer in order to display results—students could disconnect the robot from the computer, bring it to the contest playing table or other location, and manipulate the robot or conditions in its environment and directly observe changes to sensors' values.

**Electronic Beeper** An electronic “piezo” beeper was provided with simple routines for making beeps of varying pitch and duration. The assembly language board had included a beeper, but it was difficult to operate from software and was not utilized by students generally. With the new system, we saw an explosion of “musical robots” that used sound output for both entertainment and informational purposes.

**User Buttons and Knob.** To facilitate interactions with the robot’s hardware, we put two pushbuttons and an adjustable knob on the board. This allows users to create menu-based programs using the LCD display. Choices could be selected with the buttons, values could be entered using the knob, and results could be displayed on the LCD for immediate viewing. This allowed for a new sort of interaction with the robot when it was detached from the development computer.

**Infrared Capability.** Each robot was given the ability to broadcast modulated infrared light. This feature is discussed later in this appendix (Section D.4).

### D.3.2 Software

Sargent spearheaded the design and implementation of a custom programming language for use with our new board; I assisted as a sparring partner in the conceptual design of the system and as an implementor of certain subsections of it.

We had to determine which programming language we would implement. We knew that we wanted to implement a language based on the procedural metaphor because this was the most common, proven useful, and generally accepted form of programming. The actual language on which to base our system was not immediately obvious.

We implemented a trial version of the programming environment based on the Logo language. The implementation supported procedures, recursion, and integer variables. We saw Logo as desirable because of its learnability and the simplicity of the Logo syntax.

Other than Logo, the C language, another popular procedural language, was a front-runner in our considerations. We considered the following factors in deciding which language to use:

**Data Types and Structures.** The C language provides for a richer set of data types

than does Logo. For the full version of the language, we wanted to support floating point numbers and arrays. The traditional syntax for the Logo language does not provide for different data types. While Logo's primary data structure (the recursive list) is a superset of the functionality of the C-language array data structure, for implementation reasons it was impractical to support the Logo list data type, leaving us with the choice of non-standard representations in the Logo option.

**Control Structures.** C provides more types of control structures in its base language specification than does Logo. While various control structures could easily be added to Logo, they would be idiosyncratic modifications rather than standard features.

**Reputation.** The Logo language is thought of as being a baby language, suited primarily for use by elementary school children. This common perception of Logo is unfortunate and indeed incorrect—Logo is actually an elegant and powerful computer programming language—but the idea of Logo as a kid's language, in the belittling sense of "kids," is widely held.

On the other hand, the C language is generally perceived as being a serious, professional development language, having some learnability problems but otherwise being an indispensable tool for computer professionals.

**Learnability.** The syntax of C is more obscure than that of Logo; this is definitely a drawback for a novice to the language. However, the two most objectionable properties of traditional C environments—the lack of interactivity in program development and the ease with which it is possible to write a C program that will crash—were factors that we would deal with separately from the choice of language grammar and data types.

We expected that many MIT students would already know how to program in C, while those who did not would not object to having to learn it.

For a combination of these reasons, we chose to use create a version of the C language for our project. However, rather than being constrained by the traditional C model, we

implemented some features that would make our system ideal for learners, including *interactivity* and *uncrashability*. (These two features are characteristic of Logo systems but not C systems.) In addition, we added *multitasking* capability (standard to neither C nor Logo systems). So in a sense, the system we created combined the best features of both languages: the ease of use of a Logo system with the expressive power and versatility of the C language.

## **Interactivity**

The biggest problem with the software environment of the assembly language system was its batch-mode metaphor: first you would write a program, then assemble it, then download it, and then see if it worked. In contrast, the Logo Programmable Brick system provided a Command Center, in which users could type commands which would be executed immediately after being typed.

We gave the our new system a feature quite like the Programmable Brick's Command Center—an interactive command line interface. By typing functional calls and compound statements at the command line, users were in interactive control with their robot, and could directly command motors settings and examine sensor values, as well as being able to download and execute procedures.

## **Uncrashability**

One of the difficulties in working in both traditional C and assembly language, as compared to an interpreted language like Logo, is the ease with which it is possible to write programs that fail completely, causing the computer to latch up and becoming unresponsive (i.e., to crash). When compounded with a hardware environment that is unstable, debugging becomes nearly intractable because one cannot be sure if hardware troubles, software mistakes, or both are the cause of a failure. This was perhaps the most frustrating aspect of working with the assembly language board system. If a software environment could be provided that did not allow total failures, then crashes could immediately be attributed to hardware causes, simplifying debugging tremendously.

Our goal for providing an uncrashable environment should not be interpreted to mean



that programming errors could never occur, but rather that when they did, the system should respond in a controlled way, informing the user that the program has encountered some sort of error condition. In contrast, typical assembly and some C coding errors result in the computer's latching up, which gives no particular information about what part of the program caused the error to occur.

We designed specific hardware and software features into our system to inform the user about the state of program execution. The main portion of the work was done in the software, which protected users against types of errors that would traditionally cause program crashes. For example, in the conventional specification of arrays in the C language, the program is free to make array references outside of the region for which the array is declared. Put other way, it is the responsibility of the programmer to make sure that the program does not inadvertently violate array bounds. Failure to respect array bounds can result in immediate "segmentation violation" errors on some computer operating systems, or more insidious, delayed failure modes on others.

Our system was designed to report this sort of error to the user immediately, with an error code designed to assist in debugging. The result of our design was that our system would never simply crash unless a hardware condition were to blame; any type of software failure would result in an error code being reported to the user.

To assist users in determining the execution status of their programs, we designed a "system heartbeat" monitor on the LCD screen. When the hardware was functioning normally, one character of the LCD screen would blink between two different states. If the blinking stopped, then the system has crashed.

## **Multitasking**

As a side effect of the overall architecture we used for the Interactive C language system, it was relatively simple to add multitasking capability to the system. The system allowed up to about ten procedures to be interpreted in a time-sliced fashion, giving the illusion that they all are executing simultaneously.

This feature was not motivated from a usability point of view, but rather as a tool to give students a richer way to express their control ideas.

### **D.3.3 Results**

Interactive C and our new hardware system provided a quantum leap in the complexity of robots built by students, and the sophistication of ideas they developed in doing so. It became possible for students to approach the design of their robot through bottom-up rather than top-down means; the implications of this change are discussed in Chapter 5.

The Interactive C software was implemented on the campus network of Unix machines (the Athena network). The existence of an ample supply of these computers, as well as a built-in distribution channel for the software (the campus network) greatly facilitated the use of the software. The campus computer clusters became centers of creative robot development activity; also, since Athena workstations were installed in many dormitories and fraternities, students would work from their living spaces.

The technology has been improved incrementally since the first year of the Interactive C system, but not in substantive ways. Additions have included a larger LCD display, more analog input channels, and outputs to drive a servo motor.

## **D.4 Sensor Development**

From the start of our work on the Robot Design project, we knew that key to creating interesting environments for robots, and hence for the students who were creating them, was the development of versatile, functional sensor devices. Without sensors, a robot is just a machine; robots need sensors to deduce what is happening in their world and to be able to react to changing situations.

A whole industry is built around the development of sensors for use in industrial settings, such as process monitoring. The research robotics community also has developed its own set of well-understood sensor devices. But few of these sensors would be suitable for our project, because of the cost of obtaining and complexity of employing these devices. We needed to devise sensors that would be inexpensive and simple to use, yet provide a valuable function in the students' robot projects.

Several devices were fairly obvious choices. For example, pushbutton or other types of switches can act as tactile (contact) sensors. The cadmium sulfide cell is another commonly

available and easy to use device that is used to sense levels of light. The mercury switch is an example of a device that seemed simple at first glance, but turned out to have unexpected complications when put into service. Just as vibrations caused problems with the rolling-ball sensor (see Section 3.3), vibrations of the robot as it moved about caused the bead of mercury to bounce within the glass tube, leading to unreliable readings.


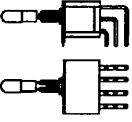
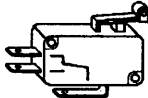


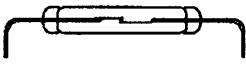


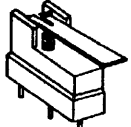
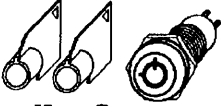

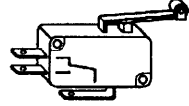
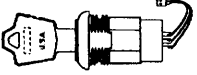

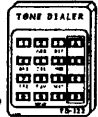
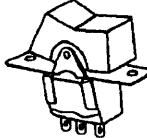
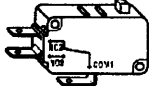

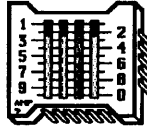

The development of such sensors was an iterative and additive process; each year we would discover, develop, deploy, and evaluate new devices. If they were successful, we would keep them for the next year. If they were not successful, we would either gain the experience needed to make them of value for the subsequent year or discard the errant device and try something different.

This process was often driven by serendipitous discovery as we perused the pages of catalogs from the electronics surplus industry. These companies stocked a hit-and-miss collection of parts at prices typically one-half or one-third of retail prices, and sometimes as low as one-tenth of a commercial distributor's price. The primary drawback of using parts from the surplus industry was not their quality—parts were generally in new condition—but the fact that stock could vary widely from year to year, making it necessary to search for desirable parts on an on-going basis.

The surplus world turned out to be a boon for our project. For example, consider the selection of switches from the catalog page of a typical surplus company, *Marlin P. Jones & Associates*, shown in Figure D-5. Any number of these devices would serve wonderfully as touch sensors for a robot. In fact, we used the “Super Mini Switch” shown in the Figure, catalog number 1408–sw, for this purpose for several consecutive years. A switch comparable to the Super Mini, which *MPJA* sold for 15 cents each in quantities of one hundred or more, would cost about \$5 each from a retail dealer.

The design of each of the contest activities was based on the availability of sensors that would be instrumental in solving it. For example, two of the contests used inclined playing fields (*King* and *Robo-Pong*) and we supplied specific sensors to detect inclination to the students. Other contests (e.g., *Robo-Cup* and *Robo-Pong*) relied on sensing surface reflectance, and sensors were provided for this capability.

The remainder of this section details the development, deployment, and evaluation of

SWITCHES			
 <p><b>PUSH BUTTON SWITCH</b></p> <p>SPST Normally open momentary push button switch 7/8" long X 3/8" dia. and mounts in a 1/4" hole. Mounting hardware supplied, solder terminals, use in test fixtures, alarms, controllers, computers, etc.</p> <p><b>3116-SW \$0.50ea WT.1</b></p>	 <p><b>4PDT LOCKING LEVER TOGGLE</b></p> <p>C &amp; K 7403, 4PDT center/off, Right angle P.C. mount mini toggle. Locking lever to prevent accidental movement of the lever. Gold plated lo-level contacts rated: .4VA Max. Locks in all three positions, pull on lever to release lock.</p> <p><b>5196-SW \$2.50 WT.1</b></p>	 <p><b>ROLLER LEVER SWITCH</b></p> <p>Removed from new equipment, Micro Switch V7 series SPDT roller lever micro switch, contacts rated 10A@125V AC, resistive, operating force =125 grams max, .187" quick connect terminals, 1/2" lever.</p> <p><b>5076-SW \$1.50 WT.02</b></p>	 <p><b>SPDT PADDLE SWITCHES</b></p> <p>Rated 5 Amps @ 120V AC R=Red B=Black C/O=Center Off D/T=DBL/Throw No Ctr/Off</p> <p><b>1736-SW B D/T 4/\$1.00</b> <b>1878-SW R D/T 4/\$1.00</b> <b>1790-SW R C/O 4/\$1.50</b></p> <p>All Weigh=.2</p>
 <p><b>MERCURY TILT SWITCHES</b></p> <p>10A @ 120V mercury tilt switch with two terminal styles, 1/4" male &amp; female quick-connect terminals, 1" dia. metal case is one contact, type 'B' has plastic support, switch closed with terminals up.</p> <p><b>2846-SW A \$1.50 WT.2</b> <b>2922-SW B \$1.75 WT.2</b></p>	 <p><b>MICRO REED SWITCH</b></p> <p>Mini reed switch. Normally Open reed switch. 9/16" long X 3/16" dia. with 1/4" long gold plated formed leads. Contact rated: 3 watts, 25A max, 100V max switching.</p> <p><b>5193-SW \$0.50ea WT.001</b></p>	 <p><b>HALL EFFECT POSITION SENSOR</b></p> <p>Honeywell 3AV3C sensors. Operated by passing a ferrous vane through the .100in. gap between the magnet and sensor. 4.5-5.5 VDC power. TTL 8mA load output. 16in. leads. .75" X .5" X .375"</p> <p><b>4814-MI \$1.95ea WT.02</b></p>	 <p><b>SUPER MINI SWITCH</b></p> <p>SPST normally open, PC mount, super small elastometric pushbutton switch with only 1/4" Sq X 3/8" tall, switch uses an elastic conductive rubber pad to provide a bounce free switching action, an added bonus is that they don't wear out, use to make keyboards, etc. ideal for CMOS. Wt.=.001</p> <p><b>1408-SW 5/\$1.00 100+/\$0.15ea</b></p>
 <p><b>LEVER SWITCH</b></p> <p>Unimax 24LMT99A, SPDT, lever micro switch, contacts rated 1A @ 125V AC 30V DC, operating force=30 grams max. PC board terminals, 5/8" lever, UL/CSA listed.</p> <p><b>4797-SW \$0.85 WT.005</b></p>	 <p><b>KEY SWITCH</b></p> <p>4 tumbler computer mini key switch. Contacts rated 4A@125V AC/28VDC. 90deg. indexing. Solder post terminals. Key removable both position. All keyed alike. Supplied with 2 keys. Fit in 15/32" hole. 1-1/8" long X 5/8" dia overall</p> <p><b>5296-SW \$3.25 WT.06</b></p>	 <p><b>MINI KEY PAD</b></p> <p>IEEKS2501-12-02, 12 button mini key pad, phone type layout with black plastic face, white silicone rubber keys with black numbers, 2-1/2" X 1-7/8" X 5/16", each button connects common to individual output pin (SP12T). Note: Won't Work With 0729-C.</p> <p><b>3085-SW \$3.00 WT.2</b></p>	 <p><b>ROLLER LEVER SWITCH</b></p> <p>VABSCO V7 series SPDT roller lever micro switch, contacts rated 5A@125V AC, resistive, operating force =25 grams max, .187" quick connect terminals, 1" lever.</p> <p><b>5077-SW \$1.50 WT.02</b></p>
 <p><b>KEY SWITCH</b></p> <p>C&amp;K Y1011C0C203NQ, SPDT, 4 tumbler key switch. Contacts rated 4A@125V AC/28V DC. 90deg. indexing, supplied with 2 keys. Solder lug terminals prewired with 4" leads to a 1" header socket. Key removable 1 position. Keyed different.</p> <p><b>4069-SW \$3.25 WT.1</b></p>	 <p><b>DTMF TONE ENCODER</b></p> <p>E.F. Johnson crystal controlled 12 button DTMF touch-tone encoder allows transceivers phone access through fixed station repeaters, +5.0 to 9V DC input (red+ black-), "push-to-talk" key down output (orange), adjustable output level (green), output tone corresponds to keypad buttons, white numbers on black keypad, 2-1/16" X 1-9/16" X 1/4"</p> <p><b>3969-EN \$29.95 WT.03</b></p>	 <p><b>16 BUTTON DTMF TONE DIALER</b></p> <p>Crystal controlled 16 button DTMF touch-tone dialer. Pocketsized dialer contains tone generator, keypad, 2 button cell batteries and a speaker. Allows DTMF tone activation thru any phone. Output tone corresponds to keypad buttons, white numbers on black keypad, 2-7/8" X 2-1/16" X 3/8"</p> <p><b>4830-EN \$12.95 WT.03</b></p>	 <p><b>3PDT ROCKER SWITCH</b></p> <p>C&amp;K 3PDT, rocker switch rated: 5A@125V AC, on-none-on action, the rocker is black and has two predrilled mounting ears, UL listed, ideal for test equipment, or control panels, a quality switch.</p> <p><b>0698-SW \$1.50 WT.1</b></p>
 <p><b>SNAP SWITCH</b></p> <p>Micro Switch V71E1 IE7 SPDT snap switch. Contacts rated .5A @ 125 VDC 10A 1/3hp @ 120/250 VAC .187" quick connect terminals. 120 grams actuation force. 3/32" actuator height.</p> <p><b>5095-SW \$0.85ea WT.015</b></p>	 <p><b>KEY SWITCH</b></p> <p>Cole-Hersee SPST, heavy duty key switch. Contacts rated 2A @ 120 VAC resistive. Key removable in both positions. All keyed different, supplied with 2 keys. Mounts in 3/4" hole. Screw terminals. For panels up to 3/4" thick.</p> <p><b>4141-SW \$4.95 WT.3</b></p>	 <p><b>MATRIX SWITCH</b></p> <p>MFG: AMP P/N: 8324 POLE(S): 4 ACTION: 10 POSITION</p> <p><b>SPECIFICATIONS/FEATURES:</b> PC mount, slide action matrix switch with gold contacts, each pole can connect to one of 10 contacts. L: 1" W: 1" H: 3/16" <b>3704-SW \$3.50 WT.01</b></p>	 <p><b>P.C. MOUNT TOGGLE SWITCH</b></p> <p>C&amp;K #7101, Vertical mount, SPDT (on-none-on) mini toggle switch. Standard bat handle, used on boards for "test/run" "reset" etc, contacts rated .4VA. P.C. solder tail terminals.</p> <p><b>4849-SW \$1.00 WT.012</b></p>

Marlin P. Jones & Assoc.

16

407-848-8236 / Fax: 407-844-8764

Figure D-5: Page from the electronic surplus catalog of Marlin P. Jones & Associates (used with permission)

a sensor technology created to allow robots to “see” each other. This story illustrates the difficulties in designing a new sensor and creating a role for it in the students’ projects to enrich their learning experience.

#### **D.4.1 Robot Detection**

After the *Robo-Puck* contest, we had gained enough experience with the infrared sensor technology to know that it was interesting (see discussion in Section 3.3.1). Having worked out the obscure noise problem, the infrared sensor was an extremely effective way to determine the location of an infrared emitting object (location being defined as an angular coordinate from the position of the sensor toward the object). Surely there would be an interesting way to develop and use the technology in subsequent contests.

One of our thoughts was that it would be a way to provide for multiple game objects, which would in turn allow the development of more complex game-playing strategies. Since the infrared broadcasting and detection worked well for *Robo-Puck*, why not create a game with multiple puck- or ball-like objects, each emitting infrared for the benefit of the robots?

We discussed this idea for future contests and it became clear that there were two significant problems with the concept. The first problem was a technical one. The infrared light from two (or more) objects would interfere destructively when reaching the sensor, meaning that neither (or none) of the objects would be detected. This is to say that if a sensor were trained on light emitted from more than one infrared beacon at a time, it would see no light at all.

The other problem was a more practical difficulty. We were afraid of the job of building a large number of reliable infrared transmitting game objects. We had had enough trouble getting just a few working pucks for the *Robo-Puck* contest, and we had seen the difficulty in helping the group of class participants debug each of their standardized controller boards. The job of designing and manufacturing a collection of transmitters (each with its own battery power—should they use disposable or rechargeable batteries?, etc.) was not one we were willing to stake a contest on.

The alternative of using infrared to detect a single game object had already been tried, so

we came up with another idea: building infrared transmitters into the robots, so that robots could “see” each other on the playing field. This idea seemed like an alternative that was practically feasible, and opened up opportunities for students to develop robot behaviors that would be based on the ability to detect the opponent robot.

We developed a method whereby a robot could selectively broadcast infrared light on one of two different frequencies while simultaneously looking for the presence of light on the other frequency. Hence one robot could broadcast light on frequency “A” while detecting light on frequency “B,” and the other robot would do the converse. This solution meant that a given robot wouldn’t become confused by detecting its own emissions. (It was still necessary to deploy the sensors in a way that shielded a robot’s light emissions from its own sensors due to the destructive interference problem mentioned earlier).

In the role of robot-to-robot detection, the infrared sensor technology became an added feature of the robot’s sensing capability rather than a central one, as it had been in the *Robo-Puck* contest. Only by seeing it in use would we be able to evaluate its value.

#### **D.4.2 Deploying the System**

Implementing the infrared light detection system necessitated additions to the set of rules that students were required to follow in preparing their competitive contest robot. If the detection system was to be useful, all robots would be *required* to broadcast a standardized frequency and “amount” of infrared light. If a robot were to use an infrared-based strategy to find its opponent, it would be severely disadvantaged if the opponent were to cheat.

We therefore installed appropriately strong language in the contest rules requiring robots to broadcast infrared light at the correct frequencies, and provided the necessary hardware and software for them to do so. We also warned the students in no uncertain terms that cheating would not be tolerated and deliberate offenders would be unceremoniously ejected from competitive play.

Nevertheless, we underestimated the cost of voluntary compliance with our plan, both as a responsibility of the students’ and our own. In order to ensure fairness, we needed a way to test robots’ infrared transmission hardware, both in advance of the contest (to help the students be sure that it was functioning properly) and during the contest (to make sure

it didn't fail, either deliberately or inadvertently).

The testing problem was difficult because infrared light is invisible; hence, some sort of specialized equipment was needed to detect its presence. Also, we had to ensure that not only was the hardware functioning, but that the infrared being transmitted was at its nominal full brightness. Each infrared beacon used a total of eight individual transmitting elements; we had to ensure that all eight of them were functioning properly.

In the *Robo-Pong* year, the first year we used the infrared robot detection system, we obtained infrared viewing cameras to look at the robots' infrared transmitters. Using this equipment, we were able to determine that the transmitters were on and appropriately bright, but we were unable to measure the frequency of transmission. This solution was inadequate because the infrared viewers were scarce and students themselves had no easy way of telling if their own circuit was functioning properly.

For the subsequent year, we modified the transmitter device by placing a visible LED lamp in series with each invisible infrared transmitter. This way, one could readily tell which of the eight transmitter LEDs were operating: if a visible LED was illuminated, it was nearly assured that the corresponding infrared one was working too. In order to check for proper frequency, we built a circuit using the same detection hardware as the opponent robot would use and tested for proper transmission using our own detectors.

By the end of the *Robo-Cup* contest, the second year of the infrared robot detection system, it was thus reasonably straightforward to test the students' hardware.

### **D.4.3 Evaluating the System**

Each year only a few student teams fielded final designs which used the infrared robot detection in a central way. There are a number of reasons why this sensor technology did not become a major factor in the students' designs, though most students did experiment with the infrared system.

From the students' perspective while developing their robots' strategy, the infrared detection system was only of significant value if they were designing an aggression-based robot. If they focused on their robot's own task, it usually meant ignoring the presence of the opponent robot and hence its infrared light transmissions. As discussed in Section 3.1,

we were diligent in the design of our contests to create games where brute force aggression would not be rewarded, especially after the oversight in the design of the *Robo-Puck* contest. Further, beginning with the *Robo-Cup* contest, students were required to demonstrate that their robot could beat a “placebo” (i.e., non-functioning) robot in order to qualify for the main contest night. (The placebo did transmit infrared light as a normal opponent would.)

Most teams planned aspects of their strategy to deal with the opponent robot, but these ideas were not the cornerstone of their robot’s strategy. As time ran out, the infrared-based strategy became a disposable “extra” that was indeed disposed. This was evidenced by the vast majority of robots which had infrared sensors mounted and installed, but ultimately not used in the final competitive program version.

For the teams that did complete a design based on infrared detection of the opponent, it was typically critical to their success that the opponent robot was indeed transmitting infrared light. During contest rounds, these teams had a vested interest in making sure their opponents’ transmission was valid, and had no qualms about being vocal about this. (Could there be a correlation between a student’s own personality and his or her choice of an aggressive design for a robot?)

During the contests a number of teams were disqualified because their robots were not transmitting infrared properly. It was understandable that students who had themselves given up on using infrared detection for their own robot’s strategy would be neglectful in ensuring that their own robot was transmitting infrared properly. Nevertheless, we really had no choice other than to strictly enforce the transmission rule. It was generally apparent that a lack of transmission or faulty transmission was due to oversight and not malice, but it was little consolation for students who would forfeit rounds (sometimes well-played ones) on such a technicality.

One particular round during the *Robo-Pong* contest epitomizes the unexpected difficulty in using the infrared detection system. In this round, a well-designed attack robot went up against a typical opponent. The attack robot, named *50/50*, used a scanning infrared sensor that panned like a radar dish, seeking the opponent’s signal. When the sensor detected the opponent, the robot’s body would spin underneath it to face the direction of the sensor, and the robot would ram its opponent.



50/50 had done well in previous rounds, but this time it missed the opponent entirely, and persistently drove into a boundary wall at the edge of the playing field. The robot lost the round as the opponent succeeded in its own task, oblivious of the hazard it had been spared.

Shortly after the round had ended, the robot's designers came up to me in a very agitated state. Other rounds had already begun and it was a very hectic time. The designers claimed that their robot had indeed locked on to the opposing robot's infrared signal—one that was *reflecting off a judge's pair of pants!* They pointed to the accused judge, who was wearing a clean pair of white pants, which are ideal infrared reflectors. He was standing quite near to the edge of the table; he happened to be the judge who was responsible for checking robot's infrared transmissions, and he needed to be up close to do it. The students called for a replay of the round they had just lost, citing this interference from the judge.

I had no way of knowing for certain whether the story I was hearing was true: I had not paid enough attention to the performance round in question. While the story was technically plausible, our predisposition during the contest was to reject complaints; there were too many things going wrong if we paid attention to them all. So I rejected the students' call for a rerun of the round, over their strenuous objections. It was only in later reviewing a contest videotape that I now believe their claim was valid. Technically, it certainly could have happened; the video record suggests that it indeed did. In subsequent years, to minimize the reflection problem, we established a rule that contest judges had to wear black pants. Students continue to use the infrared system; in recent years, there have been several robots that successfully used the system to detect and compensate for the actions of the opponent robot.

The story of the development of each particular sensor types can serve a microcosm for explaining the fashion in which the whole Robot Design project has been developed: We had a sense of what would be an interesting and viable technology; we experimented and developed the technology into workable form; we built it into the design space of the class by employing it in the contest specification; and then we evaluated the result of its use.

Sensor development is the most active portion of on-going technology development in the years of the Robot Design project since those discussed here. The students who have

taken over the administration and development of the project continue to explore sensor technologies for ones that would be viable and instructive to add to the course. The infrared robot detection system remains in place; in recent years, students have employed it to detect their own robot's orientation at the start of a round. By mounting four infrared sensors in ninety degree quadrants, they can detect the direction to the opponent at the start of the round, which yields an indication of their own robot's randomized orientation.

# Bibliography

Abelson, H., & Sussman, G. J. (1985). *Structure and Interpretation of Computer Programs*. The MIT Press, Cambridge, Massachusetts.

Ackermann, E. (1991). The agency model of transactions: Toward an understanding of children's theory of control. In Montangero, J., & Tryphon, A. (Eds.), *Psychologie Génétique et Sciences Cognitives*, chap. 4, pp. 63–75. Fondation Archives Jean Piaget, Genève.

Anderson, P. H. (1991). Amateur radio in an electrical engineering program. In Grayson, L. P. (Ed.), *Proceedings of the Frontiers in Education conference*, pp. 338–345. I.E.E.E. and A.S.E.E.

Arneke, D. (1990). Sports news: Robots play hockey. *Game Player's Sports for Kids*, 1(5).

Asimow, M. (1962). *Introduction to Design*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Banios, E. W. (1991). Teaching engineering practices. In Grayson, L. P. (Ed.), *Proceedings of the Frontiers in Education conference*, pp. 161–168. I.E.E.E. and A.S.E.E.

Breger, L. (1989). A creative design class on piezo- and pyro-electricity. In *Proceedings of the Frontiers in Education conference*, pp. 167–169. I.E.E.E. and A.S.E.E.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA(2), 14–23.

- Bucciarelli, L. L. (1988). Engineering design process. In Dubinskas, F. A. (Ed.), *Making Time: Ethnographies of High-Technology Organizations*, chap. 3, pp. 92–122. Temple University Press, Philadelphia.
- Chandler, D. L. (1992). Robotics soccer gives ideas a fighting chance. February 10, 1992. *The Boston Globe*.
- Connell, J. H. (1988). A behavior-based arm controller. A.I. Memo 1025, MIT Artificial Intelligence Laboratory.
- Ferguson, E. S. (1992). *Engineering and the Mind's Eye*. The MIT Press, Cambridge, Massachusetts.
- Flowers, W. C. (1987). On engineering students' creativity and academia. In *ASEE Annual Conference Proceedings*.
- Freedman, D. H. (1990). Robo-hockey. *Discover*, 11(5).
- Grinther, L. (1954). Interim report of the committee on evaluation of engineering education. *Journal of Engineering Education*, 45, 40–66.
- Grinther, L. (1955). [Final] report of the committee on evaluation of engineering education. *Journal of Engineering Education*, 46, 25–60.
- Jolda, J. G., Barber, P. F., & Lane, W. D. (1991). Teaching electrical engineering to non-science majors at West Point. In Grayson, L. P. (Ed.), *Proceedings of the Frontiers in Education conference*, pp. 682–686. I.E.E.E. and A.S.E.E.
- Jones, J. B. (1991). Design at the frontiers of engineering education. In Grayson, L. P. (Ed.), *Proceedings of the Frontiers in Education conference*, pp. 107–111. I.E.E.E. and A.S.E.E.
- Klein, R. E. (1991). The bicycle project: A vehicle to relevancy and motivation. In Grayson, L. P. (Ed.), *Proceedings of the Frontiers in Education conference*, pp. 47–52. I.E.E.E. and A.S.E.E.

- Maes, P. (1990). A bottom-up mechanism for behavior selection in an artificial creature. In *Proceedings of the conference Adaptive Behavior: From Animals to Animats*.
- Magleby, S. P., Sorensen, C. D., & Todd, R. H. (1991). Integrated product and process design: A capstone course in mechanical and manufacturing engineering. In Grayson, L. P. (Ed.), *Proceedings of the Frontiers in Education conference*, pp. 469–474. I.E.E.E. and A.S.E.E.
- Martin, F. (1992). Building robots to learn design and engineering. In Grayson, L. P. (Ed.), *Proceedings of the Frontiers in Education Conference Vanderbilt University in Nashville, Tennessee*.
- Martin, F. (1993). The MIT robot design toolkit. In Estes, N., & Thomas, M. (Eds.), *Proceedings of the Tenth International Conference on Technology and Education The University of Texas at Austin*.
- Martin, F., & Sargent, R. (1992). Learning engineering through robotic design. In Estes, N., & Thomas, M. (Eds.), *Proceedings of the Ninth International Conference on Technology and Education*, pp. 1191–1193 The University of Texas at Austin.
- Martin, F. G. (1988). Children, cybernetics, and programmable turtles. Master's thesis, The Massachusetts Institute of Technology, M.I.T. Media Laboratory, 20 Ames Street Room E15–315, Cambridge, MA 02139.
- Martin, F. G. (1992). The 6.270 Robot Builder's Guide. Epistemology and Learning Manual 1, MIT Media Laboratory, 20 Ames Street Room E15–315, Cambridge, MA 02139. Epistemology and Learning Publications.
- Martin, F. G. (1993). The Mini Board 2.0 Technical Reference. Epistemology and Learning Manual 2, MIT Media Laboratory, 20 Ames Street Room E15–315, Cambridge, MA 02139. Epistemology and Learning Publications.
- Meeden, L. A., McGraw, G., & Blank, D. (1993). Emergence of control and planning in an autonomous vehicle. In *Proceedings of the Fifteenth Annual Conference of the*

- Cognitive Science Society*, pp. 735–740 Hillsdale, New Jersey. Lawrence Erlbaum Associates.
- M.I.T. (1961). Report on engineering design. *Journal of Engineering Education*, 51(8), 645–660.
- Neumann, P. G. (1993). Inside risks: Modeling and simulation. *Communications of the ACM*, 36(4), 124.
- Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc.
- Papert, S. (1986). Constructionism: A new opportunity for elementary science education.. Proposal to the National Science Foundation. MIT Media Laboratory.
- Papert, S. (1991). New images of programming: In search of an educationally powerful concept of technological fluency. Proposal submitted to the National Science Foundation from the Massachusetts Institute of Technology, The Media Laboratory, Epistemology and Learning Group.
- Papert, S., & Solomon, C. (1971). Twenty things to do with a computer. Logo Memo 3, Massachusetts Institute of Technology, 20 Ames Street Room E15–315, Cambridge, MA 02139.
- Prados, J. W. (1992). Can ABET change its spots?. *ASEE PRISM*, 2(2), 17.
- Rashid, M. H. (1991). Integration of design in electronics courses. In Grayson, L. P. (Ed.), *Proceedings of the Frontiers in Education conference*, pp. 63–66. I.E.E.E. and A.S.E.E.
- Resnick, M. (1990). Xylophones, hamsters, and fireworks: the role of diversity in constructionist activities. Epistemology and Learning Memo 9, MIT Media Laboratory, 20 Ames Street Room E15–315, Cambridge, MA 02139.

- Resnick, M., & Ocko, S. (1990). LEGO/Logo: Learning through and about design. Epistemology and Learning Memo 8, MIT Media Laboratory, 20 Ames Street Room E15-315, Cambridge, MA 02139.
- Roberts, E., & Berlin, S. (1987). Computers and the strategic defense initiative. In Bellin, D., & Chapman, G. (Eds.), *Computers in Battle: Will They Work?*, pp. 149–170. Harcourt Brace Jovanovich, Orlando, Florida.
- Sargent, R., & Resnick, M. (1993). Twenty things to do with a programmable brick. Internal memo, Epistemology and Learning Group, MIT Media Laboratory, Cambridge, MA 02139.
- Schön, D. A. (1982). *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, Inc.
- Simon, H. A. (1969). *The Sciences of the Artificial* (first edition). The M.I.T. Press, Cambridge, Massachusetts.
- Troxel, D. E. (1968). A digital systems project laboratory. *The IEEE Transactions on Education*, E-11(1), 41–43.
- Turkle, S., & Papert, S. (1990). Epistemological pluralism: Styles and voices within the computer culture. *Signs*, 16(1).
- Vincenti, W. G. (1990). *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*. John Hopkins Studies in the History of Technology. The John Hopkins University Press, Baltimore and London.
- Whitney, D. E. (1990). Designing the design process. *Research in Engineering Design*, 2, 3–13.