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."[1] 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."[2]

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,

[1]Ferguson, *Engineering and the Mind's Eye*, page 159.
[2]*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. [3]

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,"[4] 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 the 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

[3] *ibid.*, pp. 160–161.

[4] "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.[5]

The report noted that "the learning of these things in 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

[5] *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,”[6] 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.”

[6] *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.[7]

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

[7]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 into 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 university 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.[8]

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:

[8] *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.[9]

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

[9]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 . . . [10]

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

[10]*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.[11]

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.[12]

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.[13]

[11]*Engineering and the Mind's Eye*, page 37.

[12]Louis Bucciarelli, "Engineering Design Process," in *Making Time: Ethnographies of High-Technology Organizations*, edited by Frank Dubinskas, page 108.

[13]*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.[14] 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

[14]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

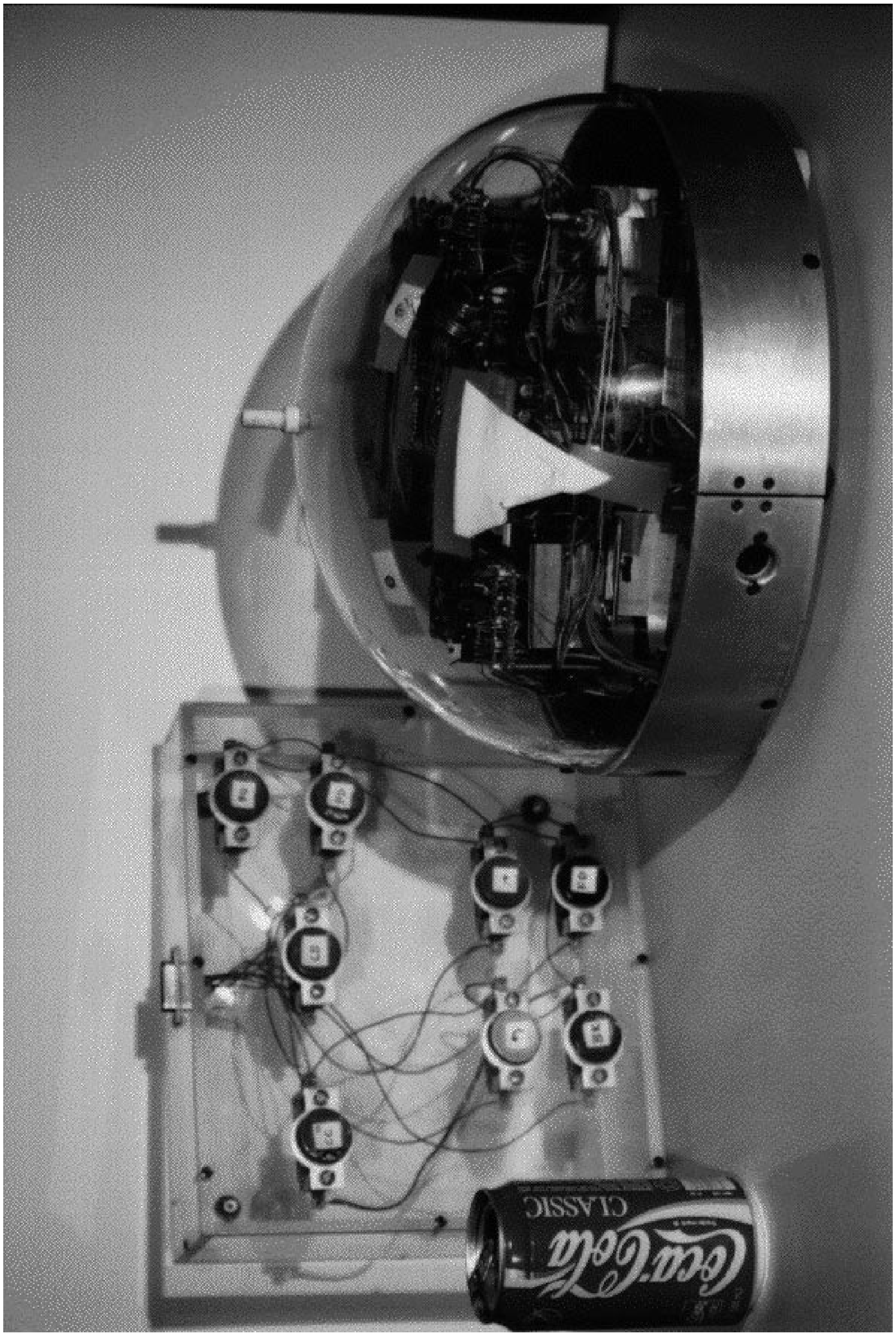

Figure 1-1: Early Logo robot and button panel for direct manipulation

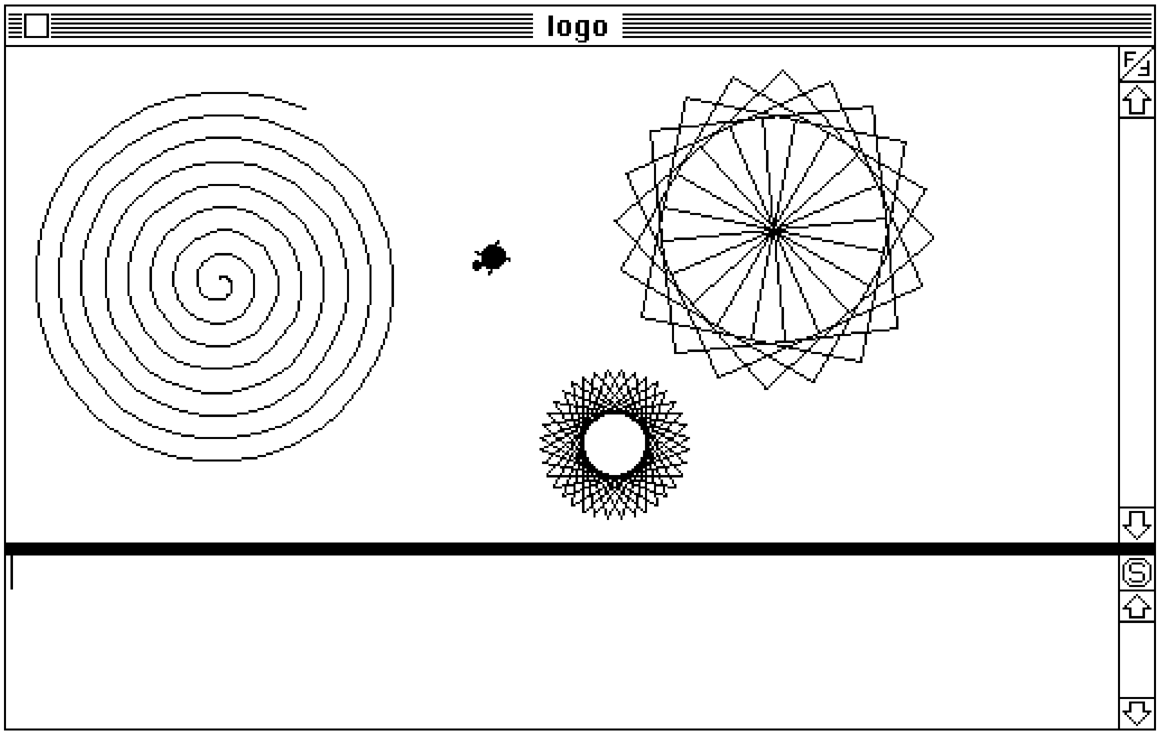

Figure 1-2: Logo turtle and typical Logo drawings made using turtle graphics

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

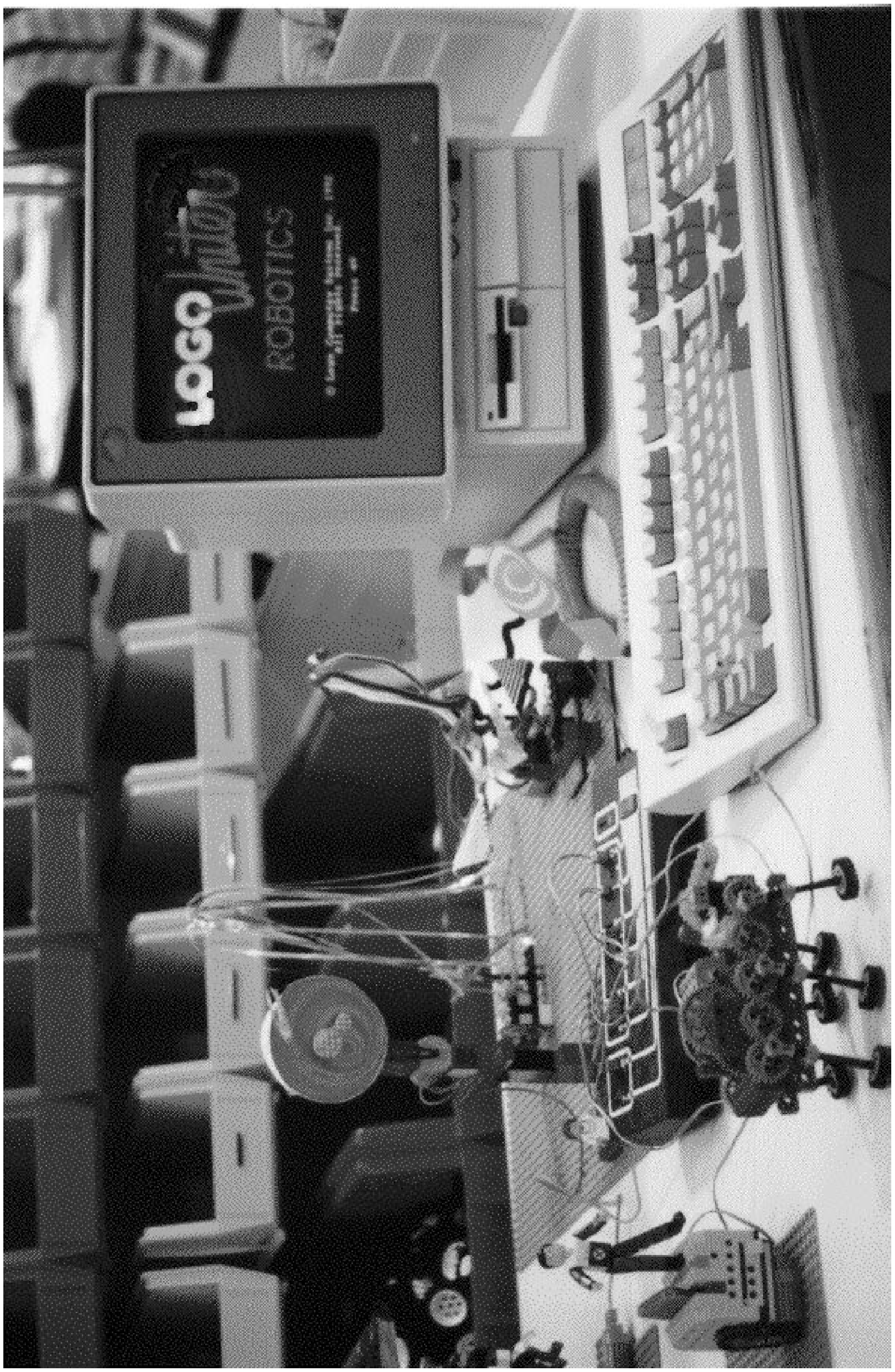

Figure 1-3: The commercial *"LEGO tc logo"* system marketed by LEGO Dacta USA

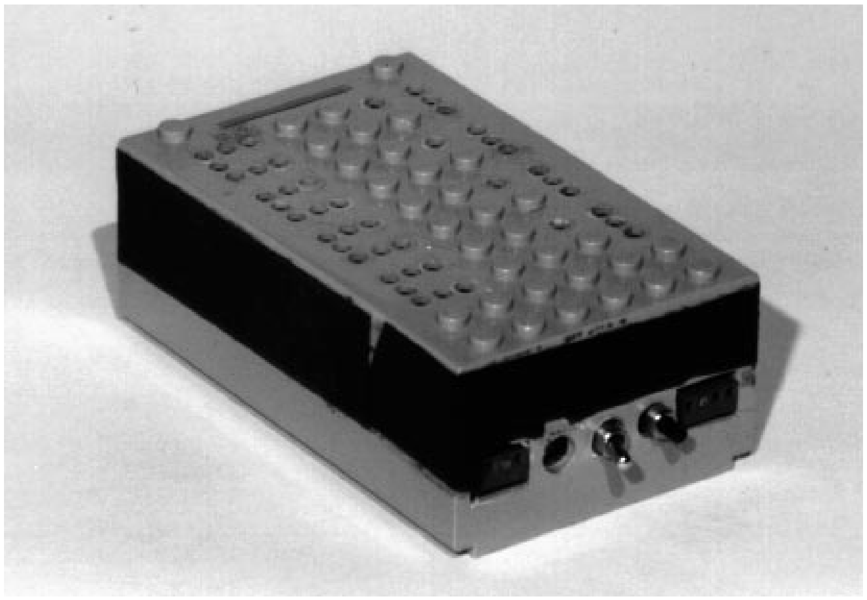

Figure 1-4: The LEGO/Logo Programmable Brick

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,

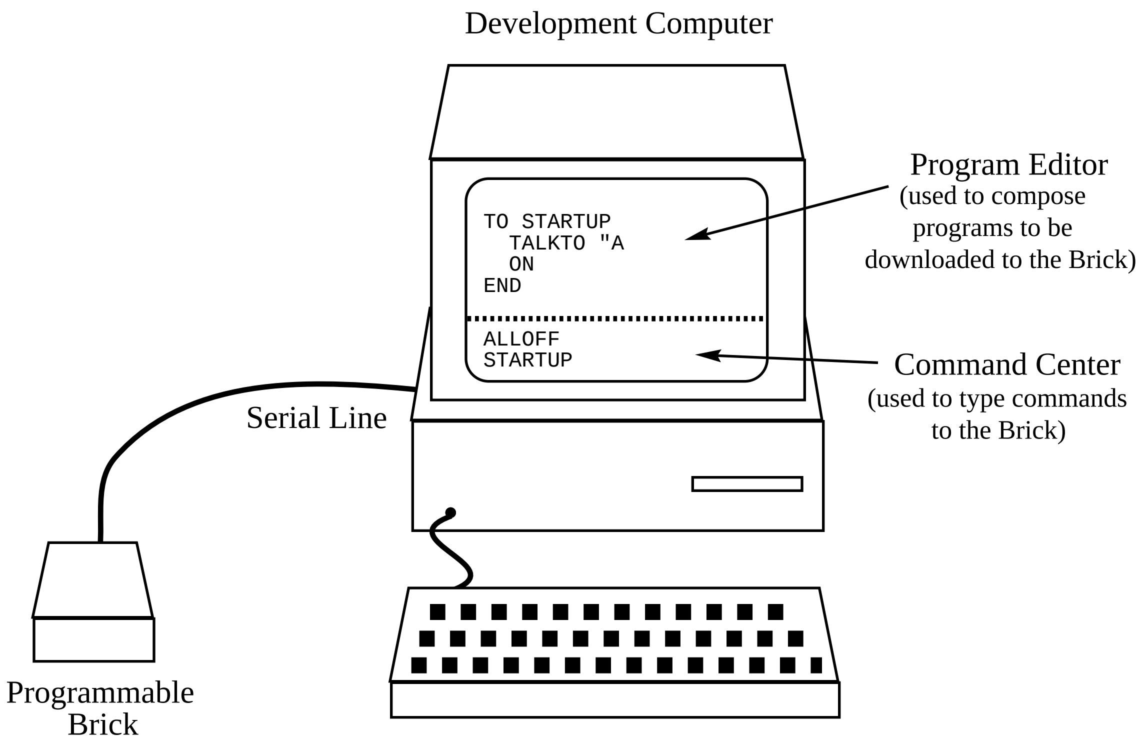

Figure 1-5: The Programmable Brick system

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 tc 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.