A Toolkit for Learning: Technology of the MIT LEGO Robot Design Competition

Fred G. Martin
The Media Laboratory
Massachusetts Institute of Technology
Cambridge, MA
fredm@media.mit.edu

ABSTRACT

The LEGO Robot Design Competition at the Massachusetts Institute of Technology is a hands-on, project-based workshop class for undergraduates developed to excite students about robotic technology as they build a competitive autonomous robot of their own design. This paper analyzes two aspects of the technology created for the workshop: the robotic contest specifications and the hardware and software control technology created for the students' use. The goal is to illuminate the iterative process of designing the workshop class, which led to lessons about educational technology, classroom culture, and its impact on students' learning. The results of this work include recommendations about structuring design problems for students, features of educational technology for maximizing students' learning, and thoughts on the role of design in engineering education.

Each year since 1991, over one hundred and fifty undergraduates at the Massachusetts Institute of Technology have participated in the "LEGO Robot Design Competition," an experimental workshop class based on the central activity of building a fully functional autonomous robot. This paper focuses specifically on an analysis of the technology created for the workshop, and lessons about effectively structuring materials and problem spaces for students' design work.

A brief introduction to the Robot Design workshop will serve as a prelude to the content of the paper. The popular class, taken by students on a voluntary basis during the January semester break, immerses students in an intensive, hands-on design experience. Working in teams of two or three, students encounter key ideas in engineering and robotics: electronic hardware, software design, mechanical design, control theory, and systems integration. More importantly, the workshop gives students the opportunity to design—to take their own ideas from initial conception to implementation, debugging, and application. In addition, as they prepare for the final competitive performance, which is attended by several hundred members of the MIT community (see Figure 1), students confront real-world engineering issues of performance, reliability, and deadlines.

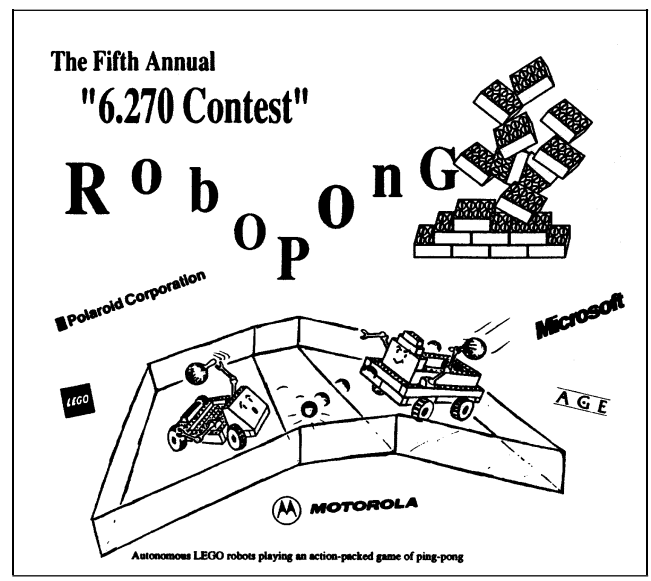

Figure 1: Poster advertising the *Robo-Pong* contest to the MIT community.

The pedagogical approach taken by the Robot Design class

has roots in the constructionist theories of learning developed by Seymour Papert [5]. According to Constructionism, the acquisition of knowledge, skills, and abilities is an active process of creation engaged in by the learner. This process can be catalyzed when the learner is building something in the world in addition to building knowledge inside his or her own mind. The artifact that is created serves as an "object to think with," in Papert's terminology, that allows the learner to reflect on his or her own ideas as they are expressed in the project itself.

Based on this theory of learning, the Robot Design workshop offers a valuable model of a classroom and workshop experience for university-level engineering students. By recognizing the central role that design can have in the learning process, the class inspires high-quality learning, genuine interest, and confidence in students at a variety of stages in their academic careers. Students learn not only about the technical issues mentioned, but also about teamwork and project management, and they experience the engineer's satisfaction and exhilaration of bringing one's ideas into reality.

This paper examines two aspects of the technology created for the Robot Design workshop: the robotic contest specifications and the custom hardware and software materials created for students' use.

CONTEST DESIGN

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

The contests attempt to 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. Additionally, the contests promote a positive social message, and strive to be inclusive of different personal styles.

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 following summarizes the progression of the Robot Design contests from its inception, in 1986, through end of this study, in 1992.[1]

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 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.

The subsequent discussion focuses on three aspects of contest design: the need to encourage a diversity of solutions in the students' designs, finding a balance between structure and difficulty in the contest problem, and the social implications of contest design.

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 in 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.

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).

All subsequent contests were based on physical robot-building materials, which by their very nature are far more

[1] 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.

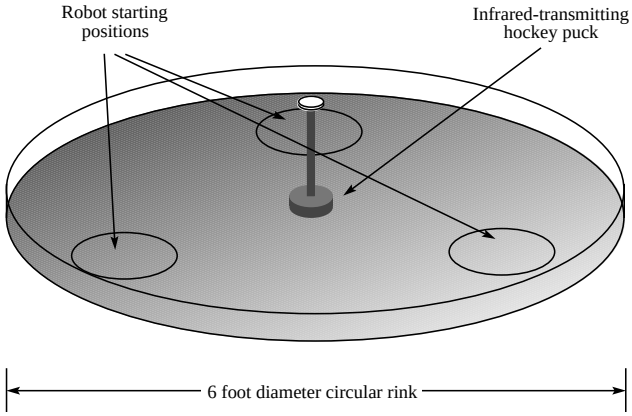

Figure 2: Playing table for the *Robo-Puck* contest.

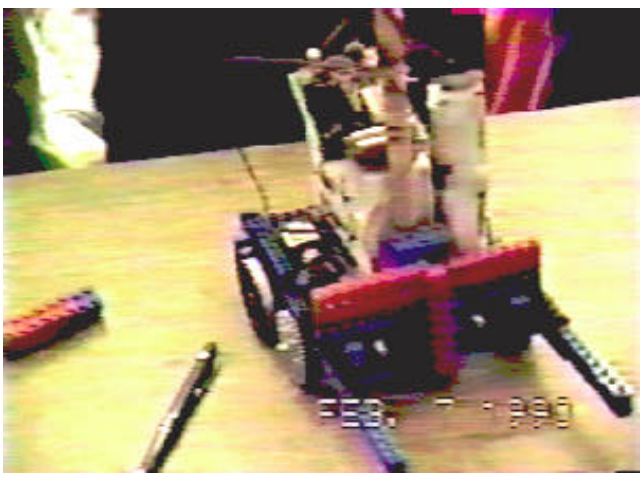

Figure 3: *Bertha*, a simple but successful puck-fetching robot from the *Robo-Puck* contest.

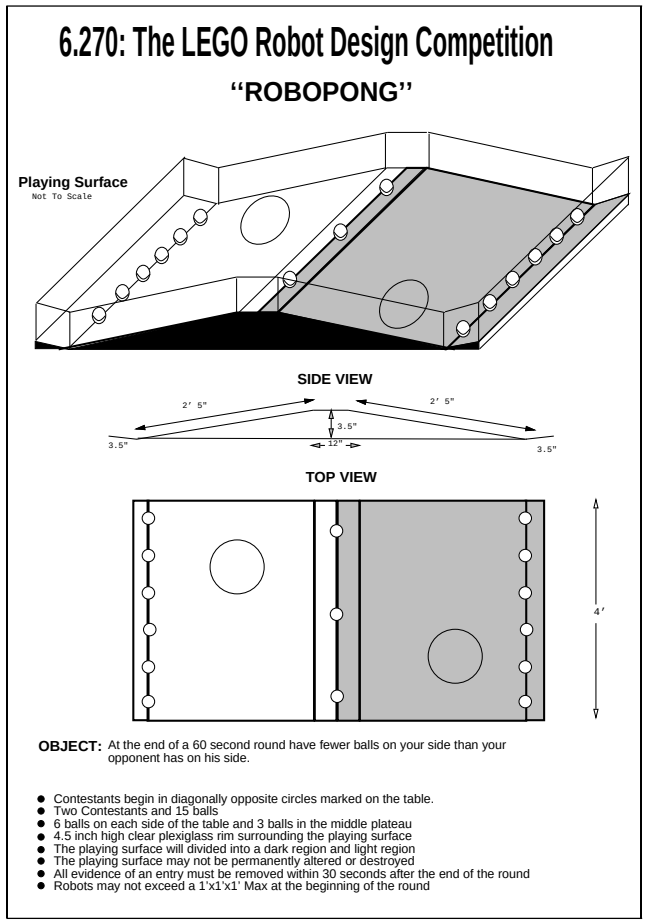

Figure 4: *Robo-Pong* game design

configurable than any predesigned software toolkit. We provided LEGO Technic materials for the mechanical aspect of robot design, a versatile and powerful kit of parts for mechanical design.

In the contests, we encouraged creativity by not overconstraining the task to be performed by the robots. Still, we needed to provide enough structure to focus students on particular robotic tasks. *King of the Mountain*, our first contest with "real" robots, was mostly a proof-of-concept effort. The task of climbing the mountain was performed with the use of inclination sensors. Students' robots were relatively simple, but so also was the control technology that we provided to them.

In *Robo-Puck*, three robots played on a six-foot diameter circular rink and attempted to gain control of an infrared-emitting hockey puck, which was initially placed in the center of the rink (see Figure 2).

Robo-Puck was also a fundamentally simple contest, but it inspired a variety of different solution strategies. The most common design was a puck-fetching robot. In this design, the robot drove toward the puck and attempted to capture it. *Bertha*, a successful puck-fetcher design, is depicted in Figure 3. Other designs included a dual-robot puck-fetcher (a smaller, faster robot grabbed the puck, while its slower, larger brother followed by capturing the first robot); a robot named *Shotgun*, which fired a retractable claw at the puck; and several aggressor robots, which attempted to flip other robots. These latter designs were not successful.

While *Robo-Puck* was based on a single game object (the puck), the *Robo-Pong* contest (see Figure 4) included multiple game objects (a total of fifteen ping-pong balls). This was a specific attempt to encourage different approaches from the students. For example, we placed three balls on the center plateau to entice some students to have their robot play for those balls first.

Results from *Robo-Pong* were quite encouraging. Successful robots demonstrated a variety of competences, such as the ability to climb uphill and downhill, maneuver in the ball trough area, and coordinate activities of collecting and delivering balls. Overall, ball-collecting robots were the most popular design choice, but a number of student teams

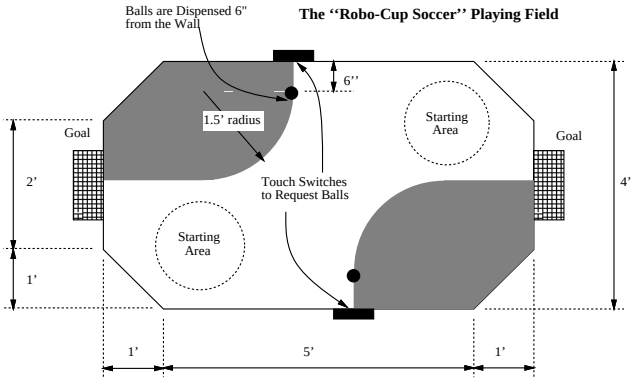

Figure 5: *Robo-Cup* Contest Playing Table.

opted for ball-shooting robots, and several attempted aggressive designs, including ones with arms that would sweep all center balls over the top and try to trap the opponent in one grand gesture.

There were several considerations behind the design of the subsequent *Robo-Cup* contest, largely in reaction to perceived flaws in *Robo-Pong*. We believed that *Robo-Pong* was not sufficiently structured to bring out the best in the students. The other problem 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 5 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.

Robo-Cup succeed in coaxing more reliable solutions from the students, but at the cost of reducing the variety of solution strategies. Nearly all *Robo-Cup* robots were one of two varieties: ball carriers, which shuttled balls from the dispenser to the goal, and ball shooters, which parked themselves at the dispenser and fired balls into the goal. The ball carriers were the dominant design, and many contest rounds ended up being races as each of two carrier robots transported balls into their respective goals, effectively ignoring the other's presence.

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.

For example, one of the *Robo-Puck* robots was intended to 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, the students building this machine had difficulties both in implementing their idea and in working together as a team. When there were just a couple of days before the contest and their robot was far from working as they had hoped, the students 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.

To the genuine surprise of all concerned, the robot (named *Stupid Scorpion*) did extremely well in the contest performance, placing third best overall among all entrants. It achieved this result through its simplicity and dogged perseverance. Many other robots would get stuck and fail for the rest of the round, but *Stupid Scorpion* just kept driving back and forth, and 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.

The Social Message

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 images similar to those promoting the popular American monster truck rallies and demolition shows. 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 1, advertising 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. 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 discourage the creation of more interesting, complex strategies.

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.

For *Robo-Cup*, to reduce the likelihood of unintentional collisions, we designed a default path pattern that would keep the robots 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. 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.

HARDWARE AND SOFTWARE DESIGN

In addition to the design of the contest specifications, 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. 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.

Observability. The students' robots became fairly complex, multi-layered systems which included mechanical, electrical, and software components. Many students had little prior experience working with such systems, and became stymied by the difficulty of debugging the interacting components of their systems. We saw this as a valuable educational lesson, but strove to make the operation of the technology that we provided evident, so that students would be able to deal with their robots' complexity.

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.

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.

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

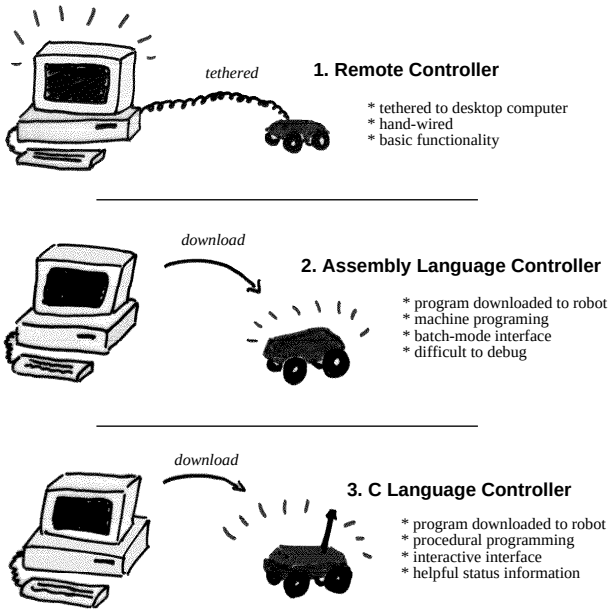

Figure 6: Three stages of robot control hardware.

of an effective educational technology for robotic design—and our understanding of what this technology should be.

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 6 illustrates these three stages of technology design.

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.

Borrowing the terminology of Abelson and Sussman [1], I call the level of functionality provided by our control hardware (for example) a *layer of abstraction*. By this I mean that the control insulates the user from lower-level issues of implementation—the electronic circuit design, for example—rather than suggesting that there is something "abstract" about the hardware. The notion of abstract in this sense is to suggest the underlying technologies that have been insulated from the user's attention.

Beginning with our earliest 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

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 oper-

ated 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.

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. 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.

Observability

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

Use of the LEGO *Technic* system will 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.[2]

[2]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

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. Programming errors tended to be of the "crash-and-burn" variety: programs failed 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 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.

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. 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

and others like it are minor criticisms of a wonderfully designed mechanical building kit.

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.

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 our C language controller system, we incorporated an command line interface. Users were then able to interactively control their robot by typing function calls and compound statements at the command line.

To highlight this aspect of the system, we named the language *Interactive C*, or IC for short. 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, 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.

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 at-

tempted to program his robot in assembly language. 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.

Course Notes. Appropriate documentation played an important role in giving students access to lower levels of the robotic system. 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.

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

CONCLUSION

The technology in use in the course has greatly evolved over the four years. As both a designer of the course technology and a teacher of the course itself, I have had the opportunity to reflect on the effects of different materials on the students' work. The following observations come from my analysis of both the development of our course materials and the experiences of the students in the class.

Design environments and materials should encourage playful design and creative exploration.

Part of the success of the course is due to the ease of use of the robot-building kit. The majority of students, particularly freshmen and sophomores, have not had a great deal of hands-on experience with design and engineering—most of their knowledge is "textbook" knowledge. As such, most students are likely to be hesitant and somewhat fearful when it comes to engaging in a practical, hands-on engineering experience—unless the materials are so friendly and easy to use that the students are not deterred.

Further, if a system actively encourages playful exploration and iterative design, then a whole new avenue of design work becomes possible. For example, the familiar and friendly LEGO Technics kit is adored by the students. Although it is complex and rich, students are anxious to explore and learn by playing with the material.

In contrast, academic design teaching is often predicated on a "top-down" model of the design process, in which needs are analyzed, constraints are determined, and an optimization process used to guide the solution (see for example [2]). While professional designer use these practices, there is growing evidence that this is not the preferred working style for many designers. For example, Turkle and Papert [8] revive Levi-Strauss' term "bricolage" to describe a "bottom-up" method of designing software:

> While hierarchy and abstraction are valued by the structured programmers' planner's heuristic, bricoleur programmers prefer negotiation and rearrangment of their materials.

Donald Schön, the design theorist, studies the work of professional designers from a variety of fields, including engineering. Schön characterizes design as "a conversation with the materials of the situation" [7]:

> 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 backtalk.[3]

In the Robot Design course, students are free to choose the style of design which either suits them best or with which they are most familiar. While some students do take a "top down" approach towards their projects, enough do not so that we as educators should take notice. Most students' projects evolve iteratively and in a piecemeal fashion: they build on top of their previous efforts, sometimes pausing to re-design a previously working mechanism or integrate several previously separate ones. I believe that for these students, the ideas and concepts they are working with are quite unfamiliar, and by giving them the opportunity to "mess around" with the ideas, they are getting a chance to become truly comfortable with the concepts without the pressure of producing a finished work at first.

[3] *The Reflective Practitioner*, pp. 79.

Materials should provide a useful level of abstraction, but not limit the students' explorations.

The robot design kit developed for the students' use is "user-friendly" because it provides a clean level of abstraction. Students do not need to know how a light sensor functions in order to build one and use it effectively. They do not need to know the low-level details of the microprocessor controller in order to write C programs for it. They do not need to know how the servo motor control is implemented in order to use the motor.

These design decisions were conscious as we were developing the students' kit. It was more important that the students have an opportunity to *use* these materials, and get excited about the creative potential of these technologies, than it was necessarily that they understand the low-level details of how the various technologies function.

Yet, at the same time, it was important not to limit the students' explorations. If a student wanted to learn the detail of the interface for a sensor, so that she or he could develop a better sensor, an opportunity should be available. Similarly for any number of topics which were in fact explored by various interested students: new motor control hardware, alternative programming languages, additional digital hardware for more sensors, etc.

In this way, the course materials and also the course philosophy accomodates a wide range of students and their interests. If some students want to concentrate in some areas while ignoring some others, they are encouraged and granted the opportunity to do so.

Students know the difference between authentic learning situations and contrived pedagogical methods.

One of the great strengths of the course is the contest, which is also its culmination. Students take a great deal of pride in readying their robots for this event; many of them literally stay up for two or three consecutive nights in attempts to get their robots in the best possible operating condition.

While a competitive situation is not without its drawbacks, I believe that the contest lends an authenticity to the design work that does not usually occur in academic learning situations. Students are not working for a grade; the course is ungraded, taken for pass-fail credit, if for credit at all. Students are responding to their own satisfaction in engineering, and they are working for the genuine admiration of their peers and teachers: everyone at an engineering school "loves a good hack."

The contest also forces students to confront the limitations of their design. Were it not for the performance situation, few students would realize how unreliable their designs actually are. Most students, while testing their machines, do not realize that one successful performance, which might be contingent on the correct operation of a dozen different actions, does not imply that those dozen actions can be relied upon. Yet many do realize this after their machines have performed in the actual contest, facing several different types of opposing robots.

ACKNOWLEDGMENTS

Wanda M. Gleason provided helpful and timely feedback on the writing of this draft, and also created Figure 6. The Robot Design project itself was the result of a collaboration with Pankaj Oberoi, who provided a steadfast organizational and inspirational role, and Randy Sargent, who co-authored the technology created for the project.

Microsoft Corp., Motorola, Inc., Polaroid Corp., the LEGO Group, Methode Inc., Gates Energy Products Inc., 3M Inc., and Abrams-Gentile Entertainment, Inc. donated much-appreciated materials and funds to support our work.

The MIT Electrical Engineering and Computer Science Department and the MIT Media Laboratory were staunch supporters of this project. I would like to thank Professor Edith Ackermann, my research advisor, for her generosity, and Professor Seymour Papert, head of the Epistemology and Learning Group.

REFERENCES

[1] Harold Abelson and Gerald Jay Sussman. *Structure and Interpretation of Computer Programs*. The MIT Press, Cambridge, Massachusetts, 1985.

[2] Morris Asimow. *Introduction to Design*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1962.

[3] Fred G. Martin. The 6.270 Robot Builder's Guide. Epistemology and Learning Manual 1, MIT Media Laboratory, 20 Ames Street Room E15–315, Cambridge, MA 02139, 1992. Epistemology and Learning Publications.

[4] Fred G. Martin. *Circuits to Control: Learning Engineering by Designing LEGO Robots.* PhD thesis, Massachusetts Institute of Technology, MIT Media Laboratory, 20 Ames Street Room E15–315, Cambridge, MA 02139, 1994.

[5] Seymour Papert. Constructionism: A new opportunity for elementary science education. Proposal to the National Science Foundation. MIT Media Laboratory, 1986.

[6] Henry Petroski. *To Engineer is Human: The Role of Failure in Successful Design*. St. Martin's Press, New York, 1982–1985.

[7] Donald A. Schön. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, Inc., 1982.

[8] Sherry Turkle and Seymour Papert. Epistemological pluralism: Styles and voices within the computer culture. *Signs*, 16(1), 1990.