Chapter 3

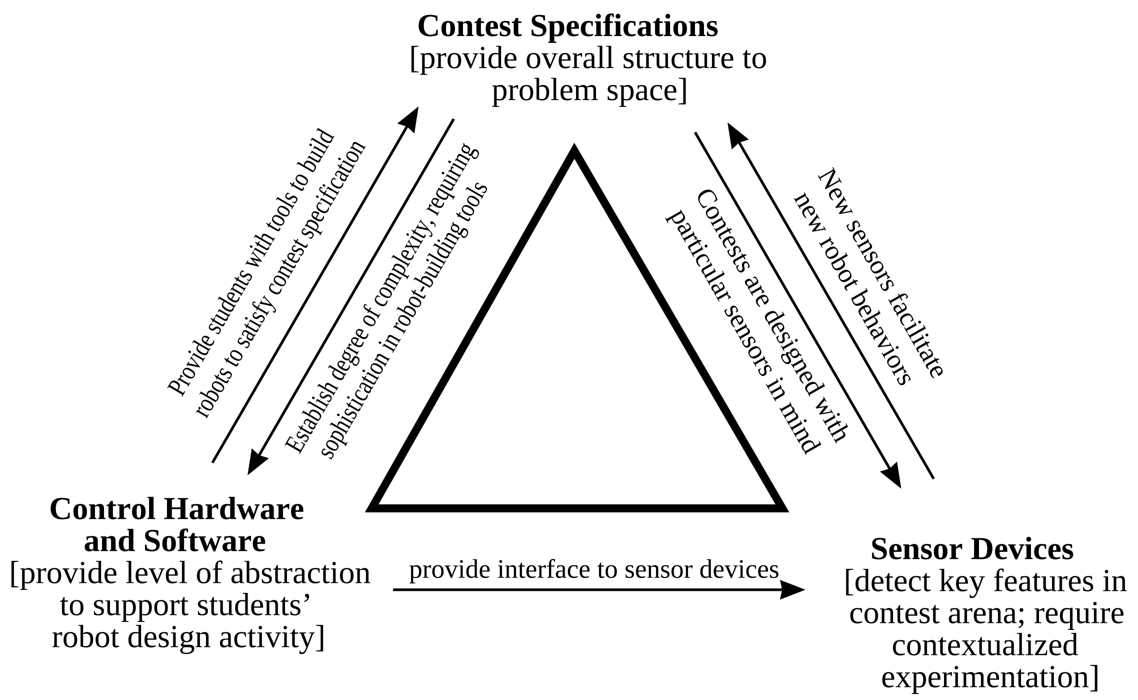
Technology for Learning

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

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

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

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



Contest Specifications
[provide overall structure to problem space]
Provide students with tools to build robots to satisfy contest specification
Establish degree of complexity, requiring sophistication in robot-building tools
Contests are designed with particular sensors in mind
New sensors facilitate new robot behaviors
Control Hardware and Software
[provide level of abstraction to support students' robot design activity]
provide interface to sensor devices
Sensor Devices
[detect key features in contest arena; require contextualized experimentation]

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

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

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

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

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

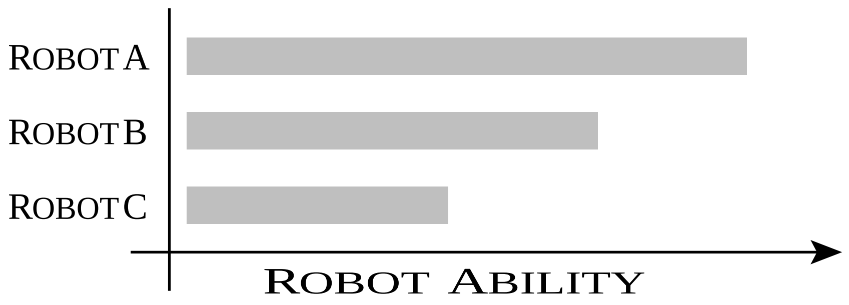
Determining the proper layer of abstraction is not a trivial issue; things should be sufficiently "messy" so that creativity is encouraged rather than stifled. We found that the best balance was struck by providing "glass boxes" rather than "black boxes": abstraction layers that students could peel away and look inside of when they desired.

We wanted to encourage students to design interactively with the materials— to use bottom-up strategies—and we made considerable effort to make our materials easy to use, encourage experimentation, and require a minimum of unnecessary conceptual overhead.

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

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

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



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

Figure 3-2: Visualization of solo strategy relationships

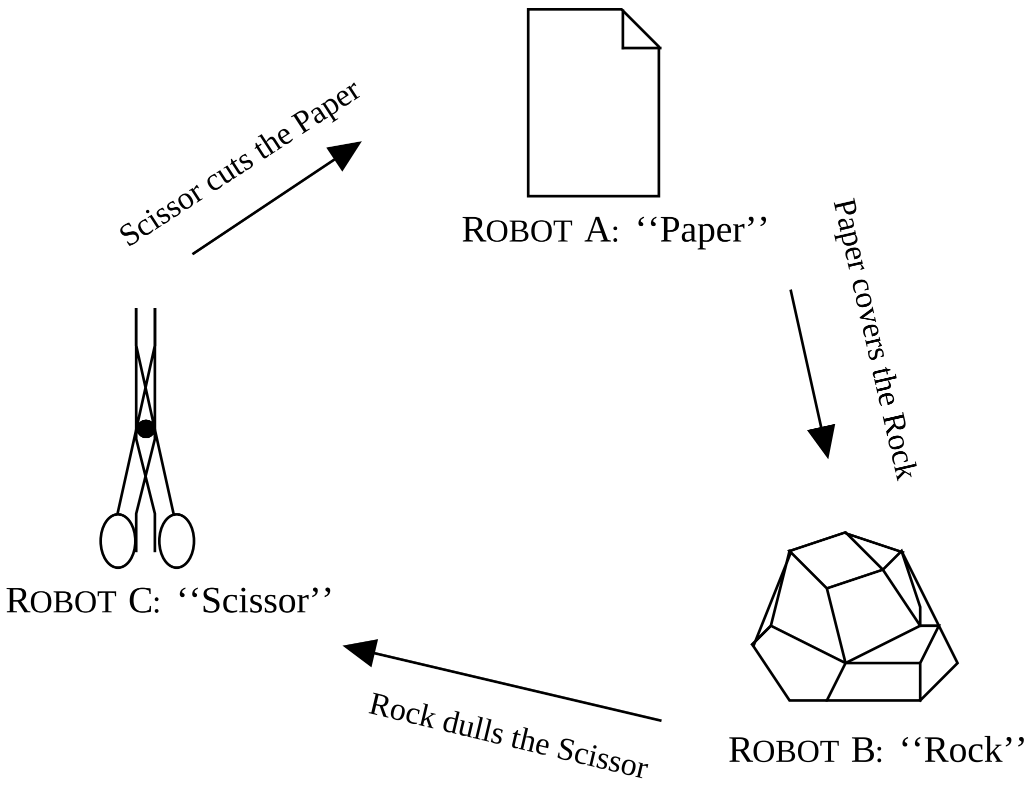
3.1 Contest Design

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

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

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

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



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

Figure 3-3: Visualization of interacting strategy relationships

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

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

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

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

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

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

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

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

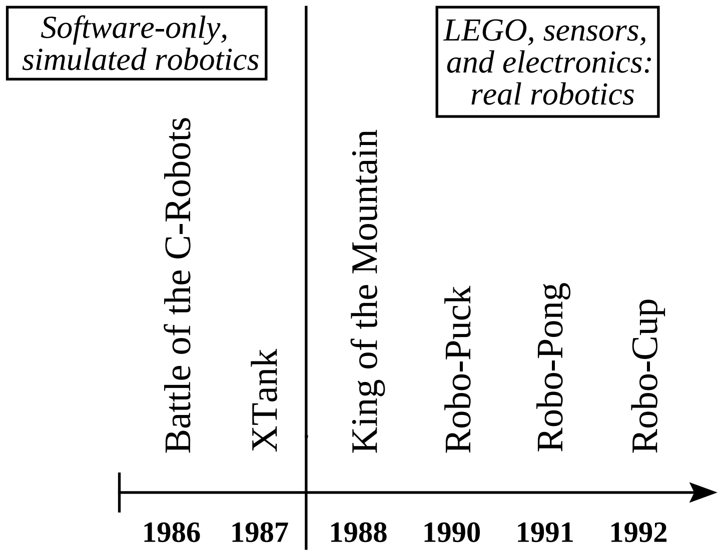


Figure 3-4: Year and name of robot design contests

in 1986, through end of this study, in 1992.[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

[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.

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

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

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

3.1.1 Strategic Diversity

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

Robot Configurability

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

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

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

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

Simplicity versus Specificity

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

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

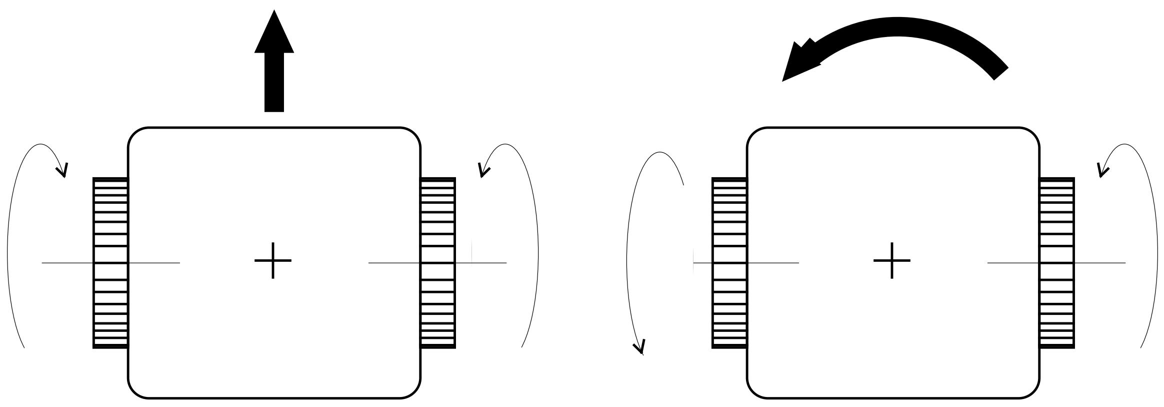

Both wheels spin same direction;
Robot advances.
Wheels spin opposite directions;
Robot pivots in place.

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

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

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

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

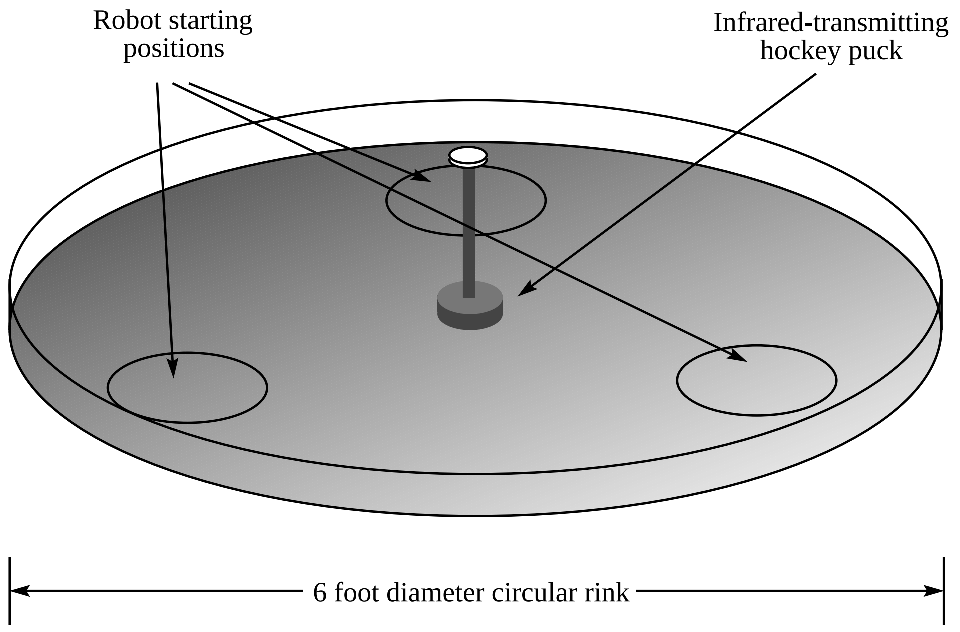

Robot starting positions
Infrared-transmitting hockey puck
6 foot diameter circular rink

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

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

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

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

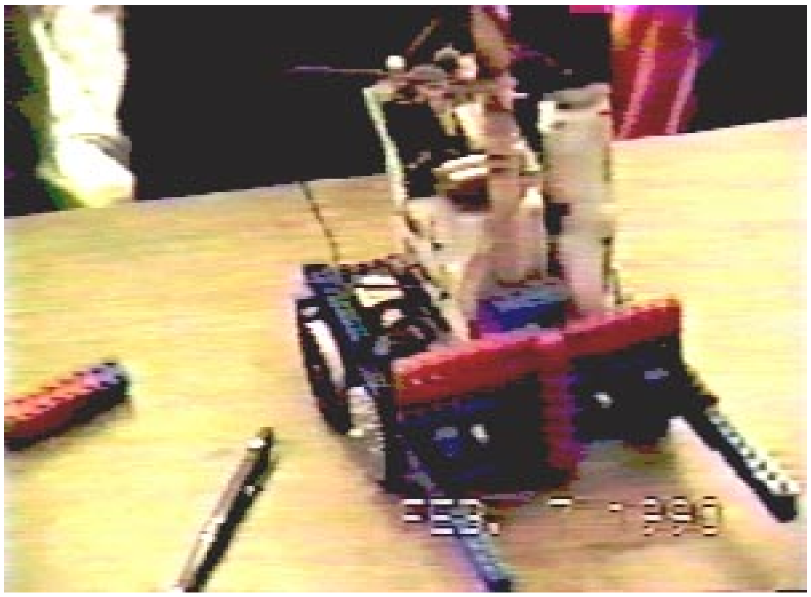


Figure 3-7: *Bertha*, a successful puck-fetching robot from *Robo-Puck*

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

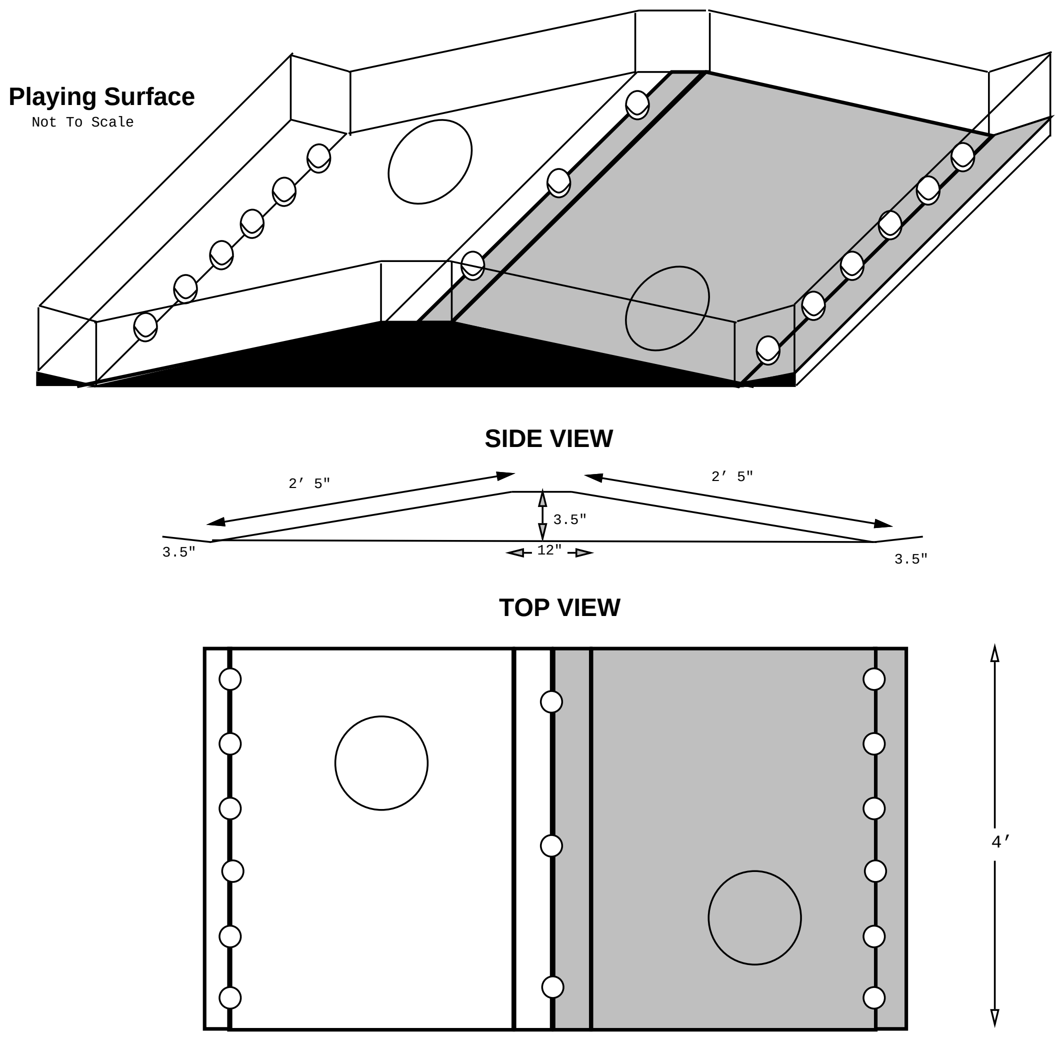
Two teams produced aggressive designs which hoped to overpower their opponents through force. One design used an extremely high gear ratio, so it moved very slowly but presumably with a great deal of power. It failed in that while it could push the other robots a bit, it couldn't force them to give up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

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

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

6.270: The LEGO Robot Design Competition

“ROBOPONG”



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

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

(drawn by Pankaj Oberoi)

Figure 3-8: *Robo-Pong* game design

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

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

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

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

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

[2]David Chen, a Visiting Professor at the MIT Media Lab during 1991, helped categorize and tabulate the robot strategies in the 1991 class.

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

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

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

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

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

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

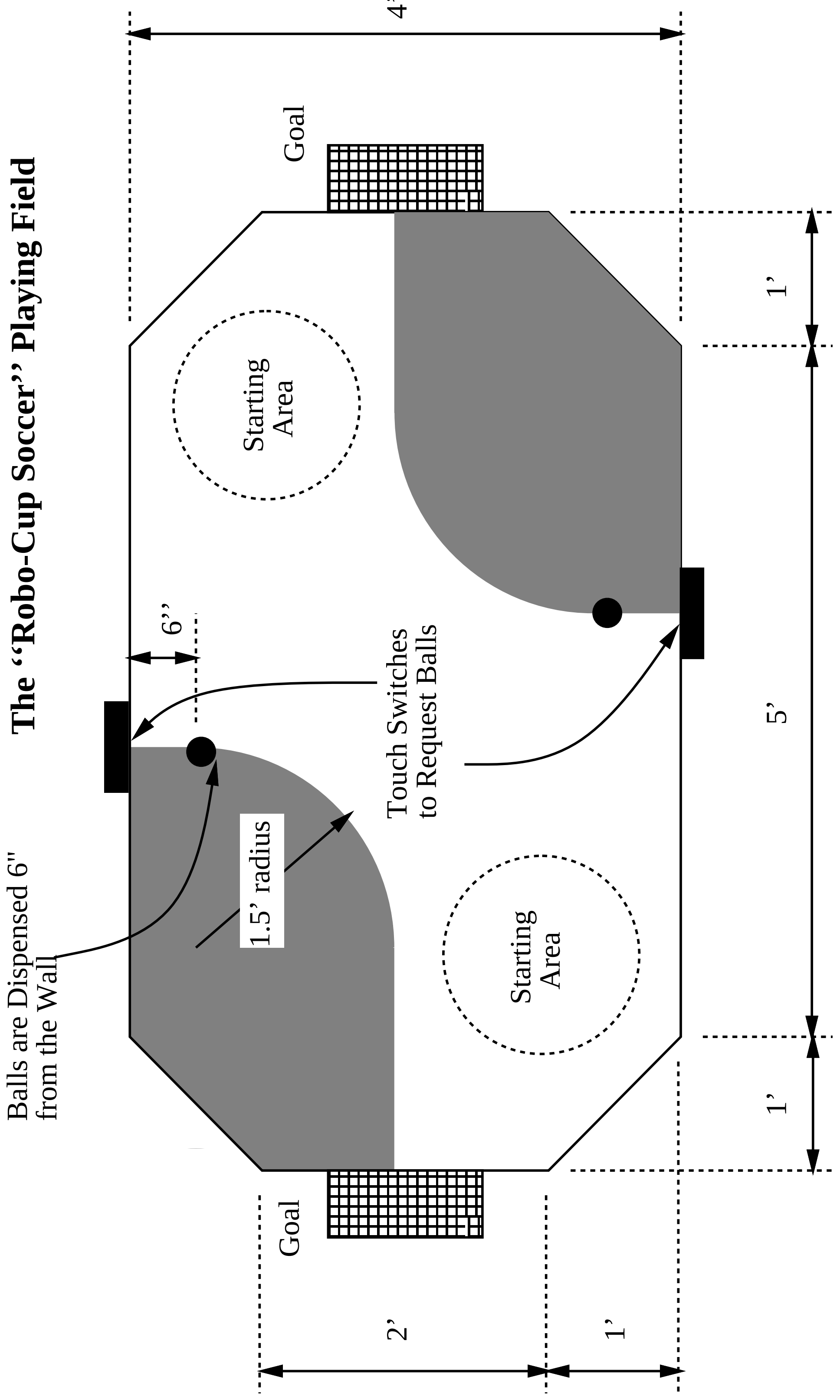

The "Robo-Cup Soccer" Playing Field
4'
Goal
Goal
Starting Area
Starting Area
1'
5'
1'
6"
Touch Switches to Request Balls
1.5' radius
Balls are Dispensed 6" from the Wall
2'
1'

Figure 3-9: *Robo-Cup* contest playing table

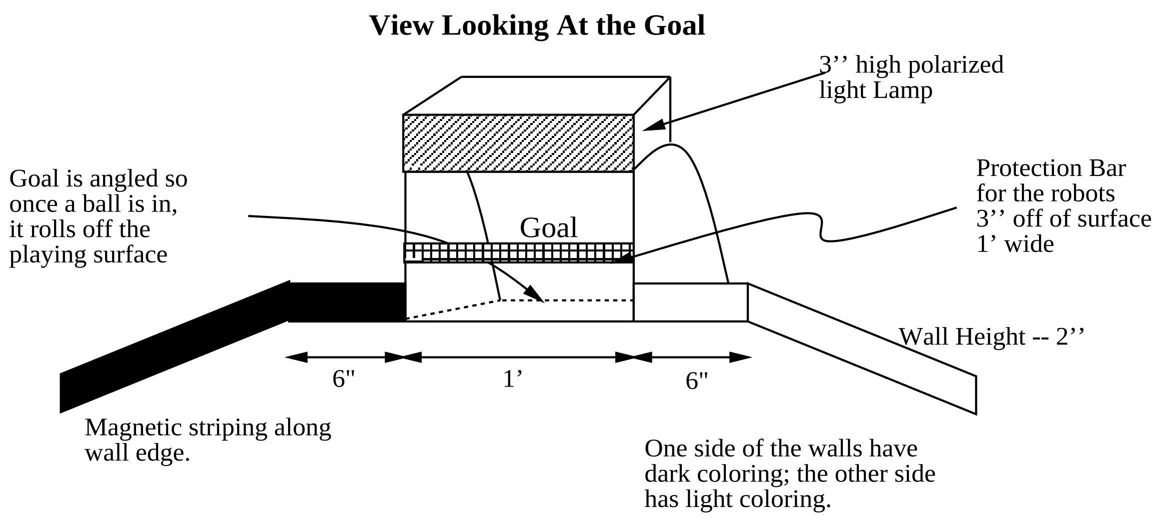

View Looking At the Goal
3'' high polarized light Lamp
Goal is angled so once a ball is in, it rolls off the playing surface
Goal
Protection Bar for the robots 3'' off of surface 1' wide
Wall Height -- 2''
6"
1'
6"
Magnetic striping along wall edge.
One side of the walls have dark coloring; the other side has light coloring.

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

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

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

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

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

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

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

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

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

3.1.2 Challenge Level

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

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

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

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

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

To the genuine surprise of all concerned, the robot did extremely well in the contest performance, placing third best overall among all entrants. It achieved this result through its simplicity and dogged perseverance (the latter a quality that it also shared with its makers)—it just wouldn't give up. Many other robots would get stuck and fail for the rest of the round, but *Stupid Scorpion* just kept driving back and forth. In theory, the robot was just as likely to hit a ball onto its own side of the table rather than onto the opponent's, but somehow it just kept winning.

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

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

- **Higher Minimum Performance Threshold.** In *Robo-Pong*, a robot could win a round (and thereby qualify for competition in the preliminary round) simply by driving uphill and knocking a ball off of the center plateau. In *Robo-Cup*, a robot had to successfully perform a number of competences, including finding the ball feeder, dispensing a ball, locating the goal, and delivering the ball to the goal. Each of these activities was individually as complex as the hill-climbing task that minimally satisfied the contest in *Robo-Pong*.

- **Disqualification for Non-Performing Robots.** Teams whose robots were unable to score a single ball in the preliminary round were given until midnight of the evening preceding the main contest to get their robots working well enough to score at least one ball in a round without interference from an opponent. Otherwise, they would not be allowed to compete in the main contest.

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

3.1.3 The Social Message

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

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

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

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

Battle of the Legos!
Come see *computer-controlled* LEGO robots battle to the death...
ROCK
&
and

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

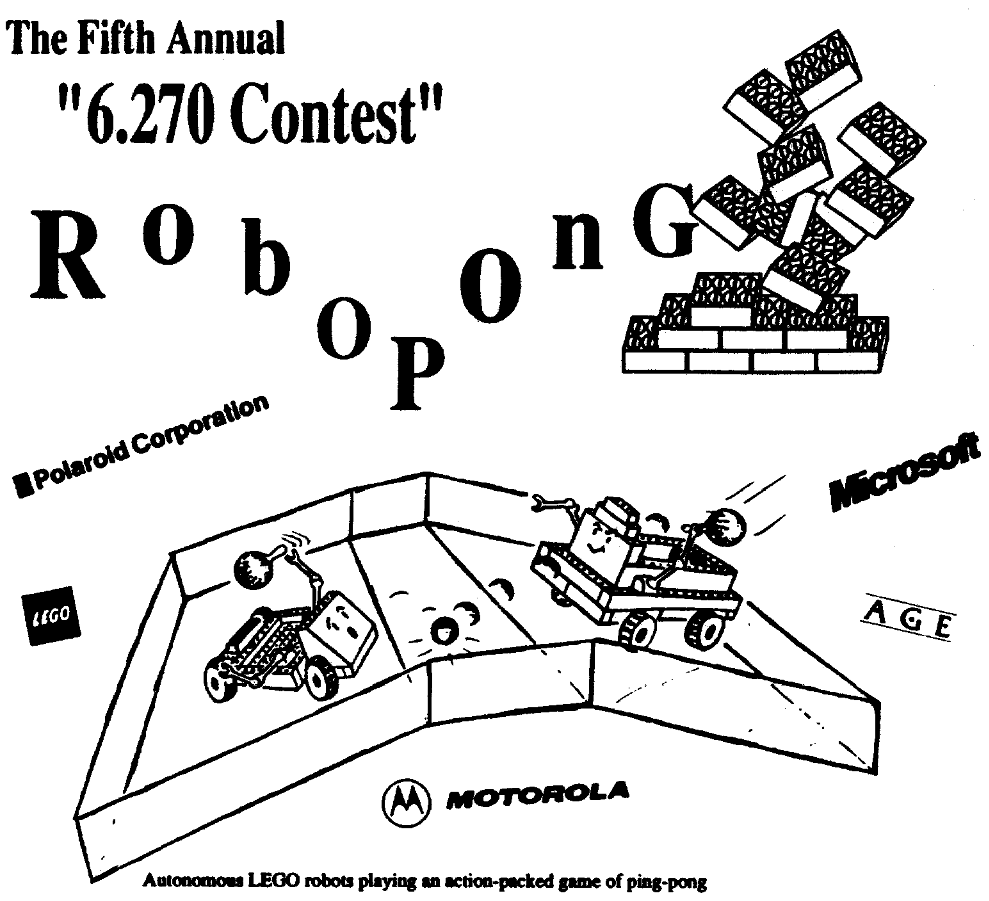


Figure 3-12: Graphic used to promote 1991 *Robo-Pong* contest

discourage the creation of more interesting, complex strategies.

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

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

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

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

3.1.4 Summary

Contest design is challenging because there are many constraints which must be satisfied to create a successful contest. The contest frames the overall problem architecture for the students’ work, and should be open-ended enough to allow for creativity and individual expression, but must be specific enough to focus students on difficult problems. Contests need to create an atmosphere of friendly sport, and must be carefully screened for quick or obvious solutions that would discourage full participation by the students.

3.2 Hardware and Software Design

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

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

- **Level of Abstraction.** Any educational technology hides or isolates the user from certain phenomena while revealing or highlighting others. In developing tools to facilitate the design of robots, we paid special attention to the sort of technological ideas we were exposing. Since a robot is a system comprised of a variety of media—electronics, programming, and mechanics—it was necessary to be clear on which concepts we expected students to master and which others they could simply use.

- **Transparency.** Even if a certain idea is encapsulated by the layer of abstraction, it should be easily accessible to students who are interested. For example, we determined that students should *not* need to have a deep understanding of digital electronics in order to build their robots. But we did not want to prevent or discourage students from exploring this topic as part of their robot-building. Quite the contrary, we hoped to invite them to do so through the design of our materials, while simultaneously taking pains not to intimidate students who might not be interested in this topic.

- **Interactivity.** Central to our project pedagogy was the belief that people learn best by *exploring ideas in a playful manner.* This was the *modus operandi* of our technology development, and a key concern was creating materials that would encourage this behavior in our students.

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

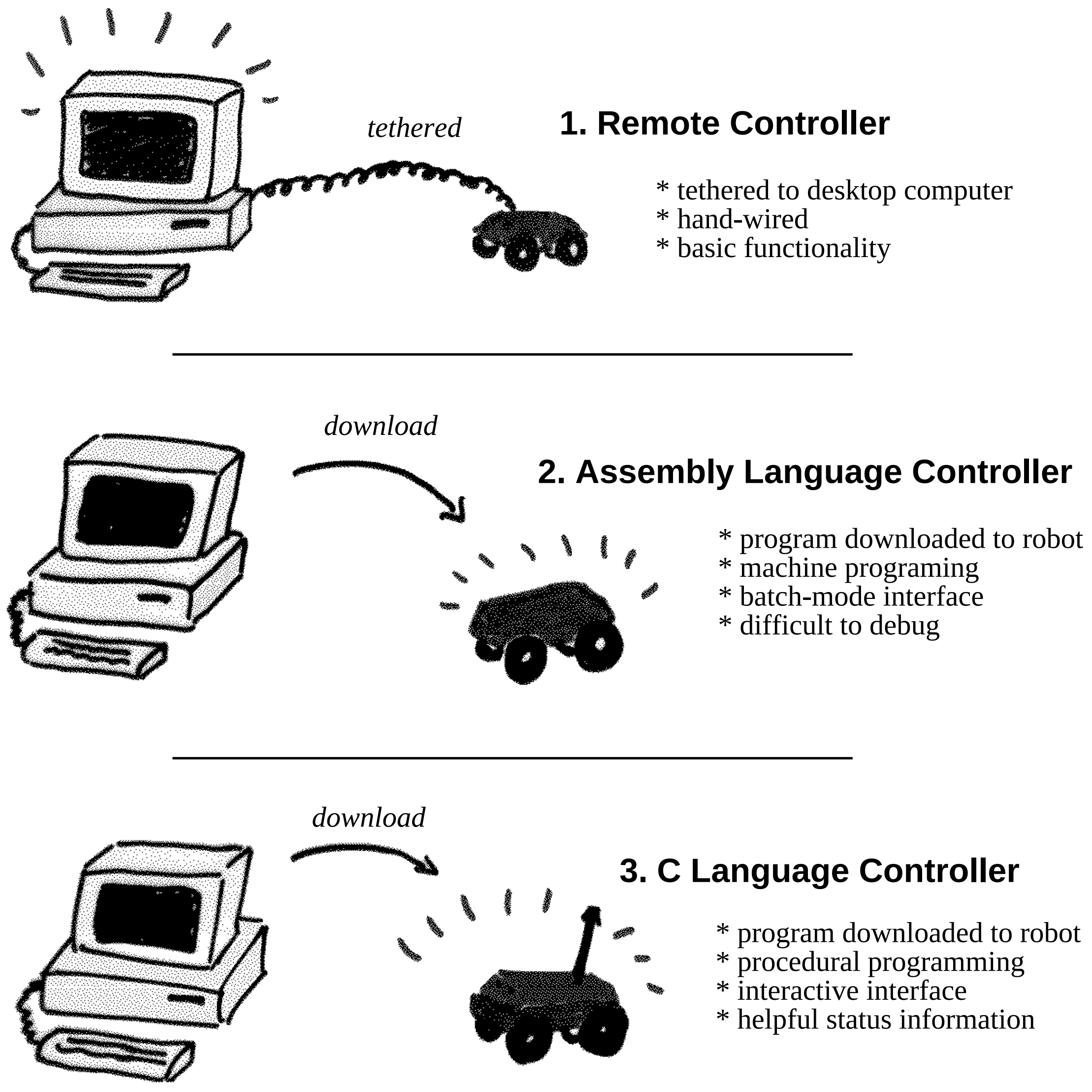
To facilitate the discussion of the core ideas of this section, a brief introduction to the technology is necessary. I attempt to make this discussion as free of technical buzzwords so possible that it may be of value to the readers who are not roboticists by background. I hope that portions of this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

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

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

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

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



(drawn by Wanda M. Gleason)

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

3.2.1 Levels of Abstraction

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

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

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

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

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

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

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

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

Bits and Bytes

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

The students who used the Remote Controller didn’t have to learn the details of the electronic circuit on which the controller was based, but they did have to work at a fairly low level in order to control motors and receive data from sensors. In order to power a motor, a byte with the correct combination of bits set to zero and one would be sent from the host computer to the controller. In order to interpret sensor data, a byte received by the host computer would be checked for a one or a zero in the proper bit position. Thus the remote controller allowed students with little circuit design experience to build robots, but forced them to deal with bit-level manipulations in order to interface with their robot's motors and sensors. We saw this as beneficial; the concepts were well within the grasp of the students, and provided a valuable lesson in interfacing hardware and software.

Machine Registers

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

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

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

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

Other students who had experience in assembly language were successful in implementing their ideas, but still felt hampered and frustrated by the assembly language, mostly for the difficulty in implementing their ideas and the painstaking debugging techniques required.

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

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

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

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

Procedural Programming

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

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

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

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

By providing the higher level tools, students were able to work with a different category of conceptual material. Rather than being concerned with what combination of bits was required to enable a motor, they could focus issues like algorithms for processing sensor data and strategic methods for organizing their robots' behavior.

3.2.2 Observability

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

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

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

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

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

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

of hardware errors:

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

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

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

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

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

Sensor-related Error. Often a sensor does not perform as a student expects. The student might create an algorithm that would perform properly if given the sensor inputs the designer anticipated; the problem arises when the sensor does not perform as expected. This failure mode is particularly tricky for a variety of reasons: (1) students don't like to think that their algorithm is inadequate because the sensor produces anomalous values; (2) sensor data tends to be noisy in a fashion that is difficult to model and for which to compensate; (3) students don't realize that they don't understand the sensor.

To further complicate matters, it is often the case that a given sensor-algorithm *mostly* works—that is, most of the time it gives acceptable performance. This error was commonplace when students used the Assembly Language controller, because it was difficult to get the system to display sensor results in a fashion that would be meaningful to the student.

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

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

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

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

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

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

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

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

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

3.2.3 Interactivity

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

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

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

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

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

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

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

2. Computer displays:

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

   C>
   ```

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

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

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

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

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

   ```
   Returns 5.
   ```

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

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

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

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

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

   to turn off Motor 0.

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

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

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

   The computer displays:

   ```
   Returns 140.
   ```

   Sensor values range from 0 to 255.

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

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

   Computer displays:

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

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

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

   Robot begins executing `testrobot()` procedure.

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

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

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

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

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

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

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

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

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

3.2.4 Transparency

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

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

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

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

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

Board.

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

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

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

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

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

To summarize, a critical concern in the development of our technology was to keep the lower levels of our system open to those students who were interested in them. This transparency made our materials more versatile as a designer's medium, and more pedagogically rich to a student base with varied backgrounds and interests.

3.3 Sensor Design

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

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

3.3.1 Collaborative Learning

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

Noise Problems, Case One

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

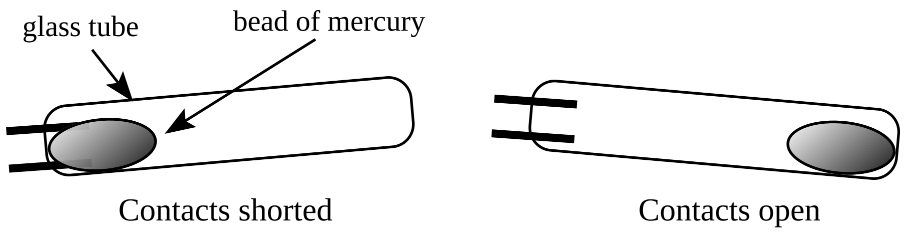


Figure 3-16: Mercury switch

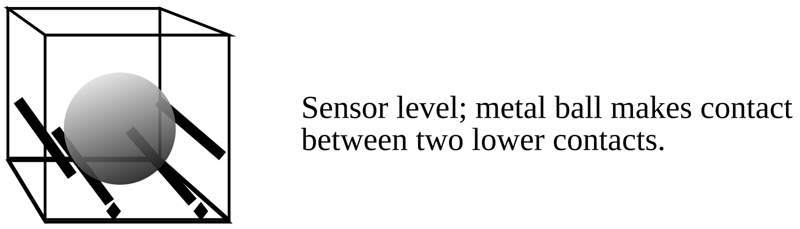

Sensor level; metal ball makes contact between two lower contacts.

Figure 3-17: Rolling ball sensor in level position

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

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

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

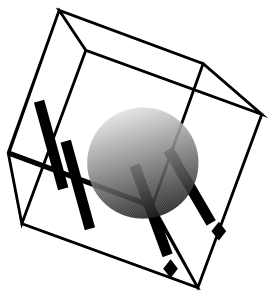

Sensor tilted; metal ball makes contact between right-hand lower contact and right-hand side contact.

Figure 3-18: Rolling ball sensor in tilted position

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

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

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

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

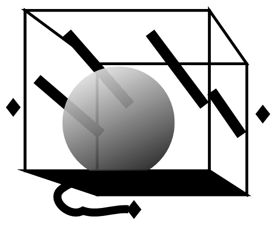

Sensor level; metal ball makes contact with bottom metal plate and left or right side contact.

Figure 3-19: Modified rolling ball sensor

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

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

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

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

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

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

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

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

Noise Problems, Case Two

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

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

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

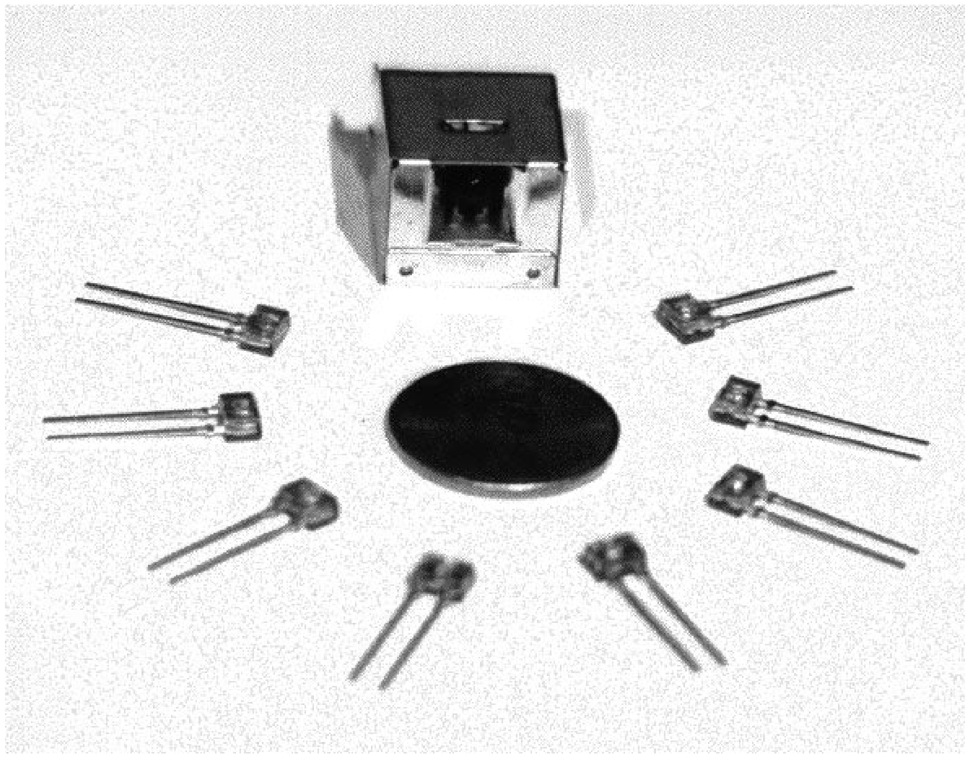


Figure 3-20: One Sharp Electronics no. GP1U52X infrared sensor and eight infrared transmitter LEDs, with a U.S. penny for size reference

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

In general, effective use of the sensor required creativity and experimentation. It was unknown what sort of optical shield would yield the best results, and in general it depended on the nature of the robot's design. Most students developed shields that restricted the sensors' field of view to between fifteen and thirty degrees, but one team in particular had an unusual application. While most robots were simply puck-fetchers, their robot used a rubber-band slingshot to fire a claw toward the puck at high speed. The robot, named *Shotgun*, employed an infrared sensor with a long narrow tube that restricted its field of view to perhaps two or three degrees. The robot needed that sort of accuracy because it only got to fire the claw one time—if it missed, the round was lost. The design used one infrared sensor with this extremely restricted vision and two others with more typical views; the wider-angle sensors were used to initially locate the puck and the narrow one was used to lock on before firing the claw. (*Shotgun* enjoyed several competitive successes but was not reliable enough to ultimately win the contest.)

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

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

Later, after the contest was over, a trivial hardware solution was found that completely eliminated the noise problem. It would have been wonderful had it been known earlier, but despite the frustration that the situation caused to the students and ourselves, the process of working with the noisy sensors was a rich learning experience, ripe with experimentation in hardware and software. Together, we were forced to be creative in their approach to dealing with a serious unanticipated problem in using a key component needed to solve the overall design problem. The answers were not known by anyone involved at the the time when a solution was needed. Certainly, now knowing how to fix the sensors so they worked properly I would not give students sensors without the fix, but it was worthwhile to go through the experimentation process together. (We still do not understand why about 10% of the sensors worked properly without the fix.)

3.3.2 Motivating Understanding

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

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

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

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

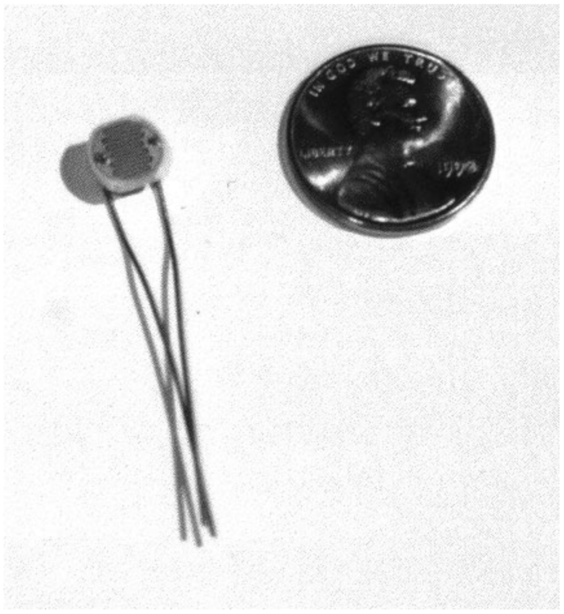


Figure 3-21: Cadmium sulfide photocell

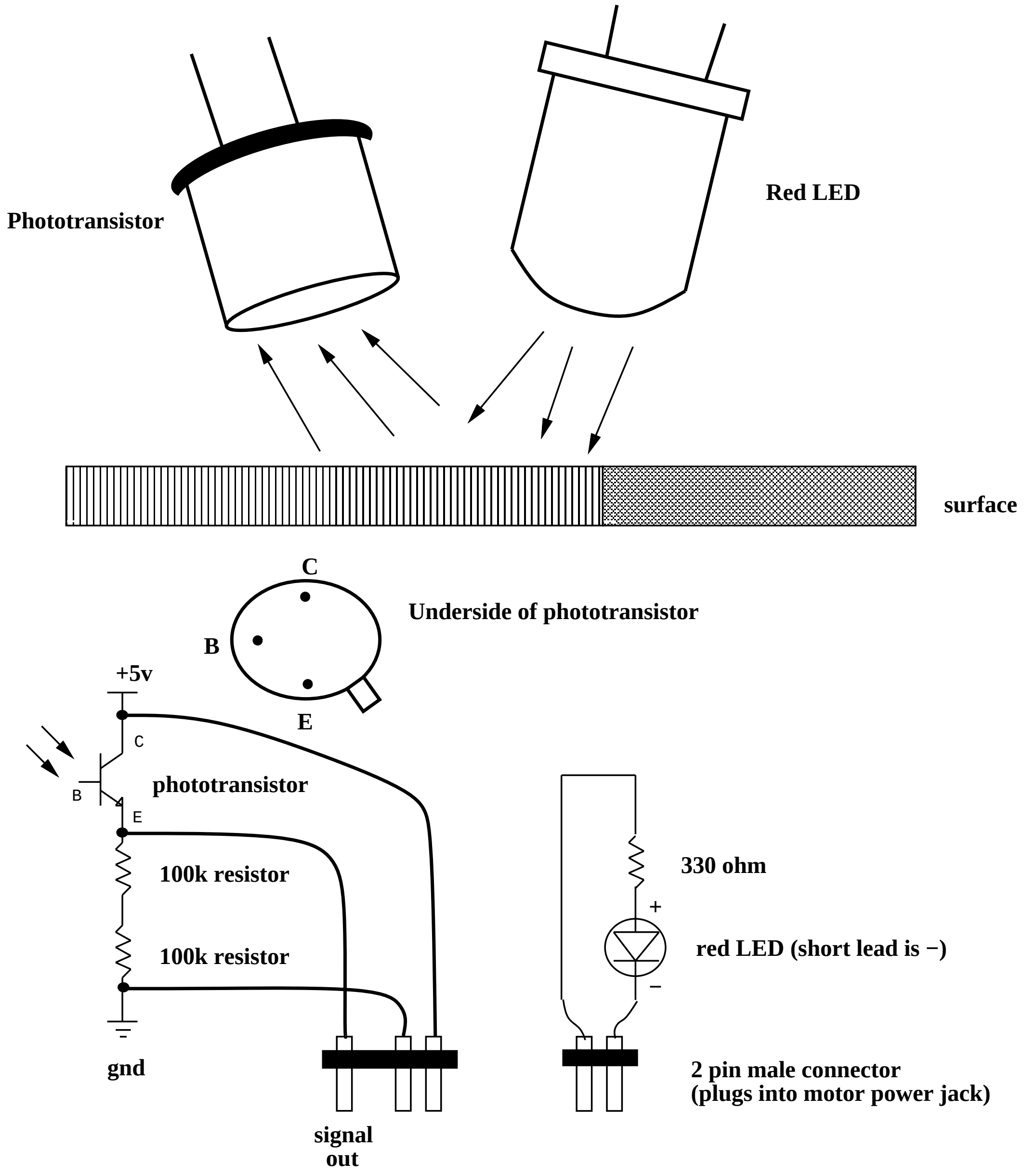
important way for robots to extract information about their location on the table. One half of the table was painted with dark paint while the other half was painted with light paint. Robots could use light sensors to detect which half of the table they were on, and, with appropriate software, drive along the light-dark edge dividing the two halves (the *Robo-Pong* playing table is depicted in Figure 3-8 on page 81).

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

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

Reflectance Sensor

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



Phototransistor
Red LED
surface
C
B
E
Underside of phototransistor
+5v
C
B
E
phototransistor
100k resistor
100k resistor
gnd
signal out
330 ohm
+
red LED (short lead is −)
−
2 pin male connector (plugs into motor power jack)

Output signal is an analog value corresponding to amount of light hitting phototransistor. Signal can plug into ANALOG or COMPARATOR port.

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

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

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

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

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

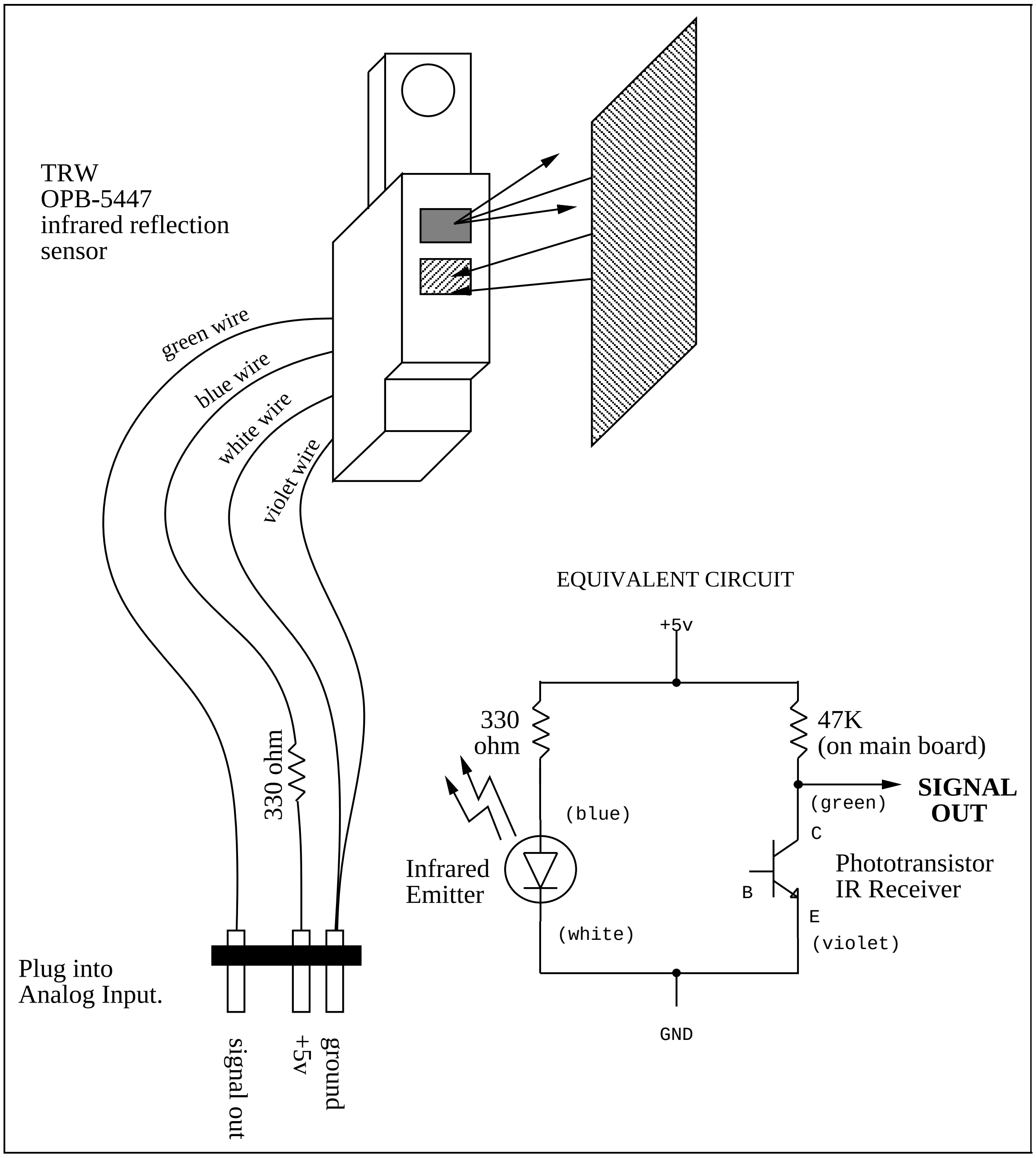


Figure 3-23: Wiring diagram for upgraded reflectance sensor provided to students in *Robo-Cup* contest

3.4 Summary

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

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

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

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

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

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