Intelligent Automation and Soft Computing, Vol. 13, No. X, pp. 1-12, 2007

LITTLE ROBOTS THAT COULD: HOW COLLABORATION IN ROBOTICS LABS LEADS TO STUDENT LEARNING AND TANGIBLE RESULTS

FRED G. MARTIN
Department of Computer Science
University of Massachusetts Lowell
Lowell, Massachusetts, U.S.A.

ABSTRACT—This paper presents a set of learning stories of student work on robotics projects. The stories illustrate key components of student learning, including demonstrations of: (1) an "emergent curriculum" of central ideas in robotics that are embedded in the materials that students work with and the projects they undertake; (2) self-directed learning and student autonomy in which students take true ownership of their projects and carry them far beyond any deliberate specification that faculty may provide (a key ingredient of graduate level work); (3) fluid teamwork, where students make significant contributions to each other's projects without being formal members of those teams; (4) mentorship, in which the faculty participant can work alongside students, guiding their process, modeling problem solving, as well as introducing specific skills and ideas; and (5) effective design, where the robot itself, as a physical object in a shared space, serves as an object for social interaction, collaboration and joint problem-solving. As a whole, the paper presents an exemplar of scholarship in which fine-grained details of student work are situated in their larger educational context to draw out their meaning.

Key Words: Robotics, Design, Learning, Collaboration, Studio, Programming, Physics

1. INTRODUCTION

This introduction takes the assumption that the reader is already familiar robotics education, through personal experience and/or the existing scholarship (e.g., this issue). In this paper, my focus is on close analyses of student work, situated in their educational context, and the interpretation of the nature and value of the corresponding student learning.

The stories presented in this paper should feel quite familiar to readers in the community. I imagine you would readily think of a half-dozen similar ones from the last semester of your own teaching. Nevertheless, I propose that it is crucial to record, discuss, and interpret these learning experiences, which we have fostered in our students.

At first glance, this type of scholarship would seem to be more relevant in the small college context, in which faculty have the opportunity to work quite closely with their students. I propose it is valuable in larger educational settings as well. By understanding how and why robotics is effective in engaging students and promoting learning, we can design better courses and learning experiences at all levels and sizes.

A particular challenge and opportunity of educational robotics is the connection of hands-on design work with the appropriate formal engineering science theory. In educational robotics, ideas are made tangible and concrete in a way that blackboard discussions and paper-based assignments cannot. However, it is often the case that students can get things to work without having a deep theoretical understanding of why they work. This is not altogether bad; in the history of engineering, there are many cases where practical, tacit knowledge preceded formal understanding. In present practice, model-based simulations allow systems to be designed without closed-form theoretical analyses [5].

In many theory-based undergraduate courses, practical problem-solving is introduced explicitly to exercise the course's theoretical content. In other words, when students encounter a typical engineering problem set, they know a priori (from the classroom context) which theory should be brought to bear on the problem. Of course, real engineering is not like this. No one visits you at the job site and whispers in

your ear, "Use finite element analysis to model this system." In the key assumption of the typical undergraduate curriculum, we hope that by introducing students to a large collection of theoretical tools, they will choose the correct one (or ones) for a new task at hand.

Robotics education, in contrast, is the place where students can encounter partially-structured problems and apply the theoretical knowledge that has been delivered to them by their prior academic work. To use the language of noted mathematics educator Richard Lesh, robotics becomes a "model eliciting activity" [10]. This is defined as an activity in which students produce "sharable, manipulatable, modifiable, and reusable conceptual tools … for constructing, describing, explaining, manipulating, predicting, or controlling mathematically significant systems" (ibid, pp. 3).

In Lesh's work, students are asked to create models that explain their understanding of what appear to be, on the surface, traditional word problems as would be given in a mathematics class. However, Lesh's "model eliciting activities" are specially designed to encourage students to draw upon a wide and diverse swath of their existing mathematical knowledge (rather than exercising a particular, focused bit of it). Students of Lesh's activities develop a problem-specific form of mathematics, helping them solve the particular problem at hand.

In the case of robotics, students' robot control program is like a computationally active model that represents their understanding of the control problem [12]. Students' robotic systems are also computationally active; that is, students run their robots and a computer carries out their program. Robot projects naturally lead to collaboration among students because the robot itself is a tangible, shared physical entity. The robot thus acts as an "object to think with" as described by Papert [16]. Students' design ideas are embodied in the robot, which enacts them as its code is run. As an entity whose behavior is readily observable by everyone in the vicinity, students naturally discuss and help debug each other's work.

The style of work we observe in robotics laboratories occurs in other learning environments. In a study of an undergraduate software design project, Kuhn found many features that are shared with the working style that is present in architecture design studios. These include: student work that is "complex and open-ended," design projects that "undergo multiple and rapid iterations," and faculty interventions that include imposing appropriate constraints to help students navigate the open-ended nature of the design problems [9].

When successful, student projects move beyond a classroom assignment and become significant endeavors. As described by Fischer and Scharff, "The challenge for environments supporting self-directed learning is to allow learners to work on authentic problems and tasks of their own choosing, and yet still provide them with learning support contextualized to their chosen problem" [6]. In the work described here, this is done within a framework of collaboration and joint problem solving.

A word about methodology. Some readers may be concerned that the results reported in this paper are not based on a systematic collection of data (e.g., using surveys administered to all students in the course). Instead, the value of the work derives from intimate observations and interpretations of student work and problem-solving. This is a sort of "thick description" of situated activity as established by noted anthropologist Clifford Geertz [8]. The data reported here are provided in its full context so that readers can easily make connections between these observations and their own students and classrooms.

The heart of this paper now follows. There are three narratives of student work in developing robotics projects. In these case studies, we can see these principles in practice. The narratives are followed by a summary discussion.

2. FORMAL ANALYSIS OF STEERING TORQUE

A limitation of using LEGO materials for classroom robotics is that one rarely encounters problems that need formal analysis involving physics. Generally, trial-and-error experimentation is adequate for solving problems—and indeed, can encourage student creativity through highly iterative design processes [11]. In this learning story, I describe how I was able to move students beyond this mode, as we encountered a problem that begged for formal treatment.

In my present version of the semester-long robotics course using LEGO materials, we keep the final third of the semester open for student projects, which may vary widely based on students' interests. In the Fall 2003 course, a group of about 10 students elected to convert a child's ride-on car (the Mattel PowerWheels) into an autonomous robot. It was a significant undertaking, which was only partially completed by the end of that semester.

One sub-system involved adapting a motor to the car's steering system. The car came with a simple steering system where the child turned a steering wheel back and forth. The steering wheel had an axle rod mounted with a double-bend that actuated a steering bar. Rotating the wheel would translate the steering bar to the left and right, causing the wheels to steer.

The student in charge of mechanically actuating the system suggested that we obtain a large-scale, high-torque servo motor and use it to actuate the steering bar. The student's idea is shown in Figure 1. It seemed like a reasonable approach, so we purchased the largest hobby servo we could find, at a cost of about $50.

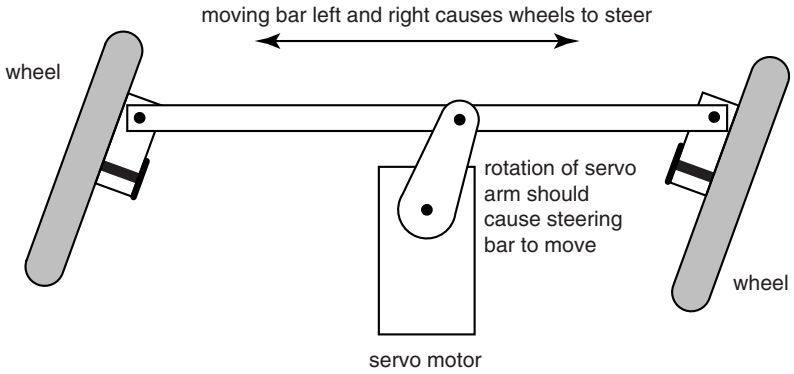

Figure 1. Power Wheels' steering linkage and initial student concept.

The first moment of truth arrived when the student mounted the servo arm to the steering bar and applied a control signal. The motor could move the linkages—but only when the front end of the vehicle was lifted off the ground! When the front end of the car was resting on the ground, the servo would stall and the wheels would not steer.

What to do? At this point, a number of other students in class and I became involved. We all wanted to see the car work, and the steering subsystem was obviously on the critical path.

I began to think out loud, leading students through a discussion of the mechanics involved. All of the students in the class had taken and passed two semester's of college physics—a requirement for our computer science degree—but few, if any, had put it to use in any practical way.

Clearly, we could shorten the servo arm linkage, since this would reduce the torque necessary to translate the steering bar. But this would reduce the amount of travel of the bar, and thus the amount that the wheels steered. In short order, we realized that the problem reduced to: *"How much torque is required to turn the car's steering wheel itself,* and *does our motor have the ability to do it?"* We looked up the specifications of the motor and learned that its torque rating was 300 ounce-inches, or almost 20 lb-inches. The question then was: how much torque would it take to turn the steering wheel?

The problem would have been straightforward if we had a calibrated spring scale in the lab. Since we did not, we invented one on the spot, using materials on hand. First, we found a 16-ounce soda bottle and filled it with tap water. Making use of the fact that one milliliter of water weighs one gram, this provided a known mass of about 500g or 1 lb. This bit of a priori knowledge became the grounding point in our series of measurements and calculations (plus the fact that a container of known volume could be found in the trash). The apparatus is depicted in Figure 2.

Next, we found a box of thick general-purpose rubber bands and hung the soda bottle from it. We measured the stretch of the band due to the mass of the bottle to be approximately 2 inches.

At this stage we had a real-units-based measurement of the performance of the rubber band, so we set off to measure how much force was required to turn the steering wheel of our car. We attached the band to the edge of the wheel and stretched it again, normal to the wheel.

We quickly realized that the band was stretching beyond its linear performance region. We then doubled, trebled, and ultimately quadrupled the rubber bands for our measurement. With four bands, we obtained a stretch of 3 inches when the steering wheel started to turn. Thus, we calculated a force of 1.5 (based on the stretch amount) times 4 (based on four bands) of the soda bottle mass, yielding 6×1 lb or 6 lbs to turn the wheel. The steering wheel had a 5-inch radius, so the entire experiment determined that 30 lb-inches of torque would be required to turn it. Figure 3 illustrates the experiment.

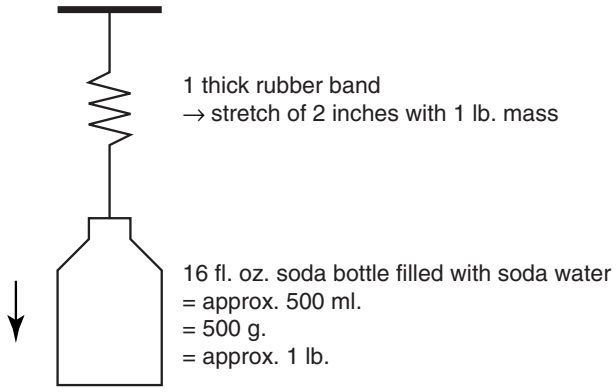

Figure 2. Spring scale built with rubber bands and a soda bottle.

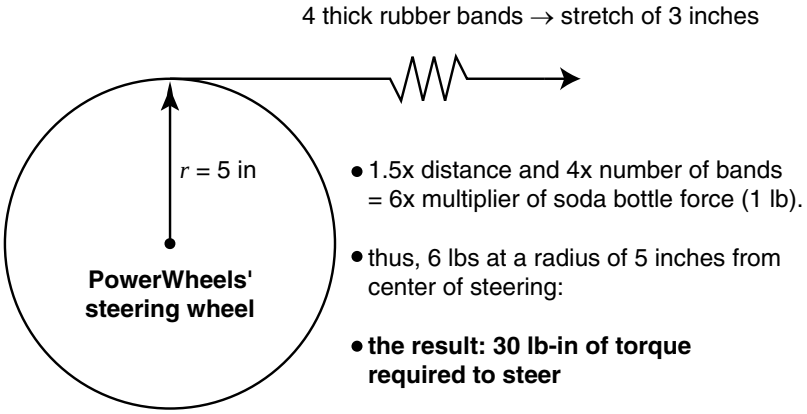

Figure 3. Measuring the force required to rotate steering wheel.

As mentioned, the hobby servo motor we had procured had a torque rating of 300 ounce-inches, or about 20 lb-inches. *Therefore, a mechanical advantage of at least 1.5:1 would be required in order for the motor to be able to turn the steering.*

We then designed a gear reduction system, as shown in Figure 4. A small pinion gear was mounted on the servo motor (shown mounted on the vertical aluminum bracket). A large gear (three times the size of the pinion) replaced the steering wheel. The gears were drawn using an open-source gear generation web site [1] and manufactured from 1/2" aluminum stock at a nearby contract manufacturing shop. The design gave a 3× torque advantage over the servo motor's intrinsic capability, and thus provided a 2× safety margin over our experimental measurements and calculations. After the whole analysis, design, and manufacturing process, we assembled the physical gears to the chassis and motor. We were rewarded when the servo motor indeed possessed the power to actuate the steering!

This discussion does not suggest that the students came up with this on their own. As was narrated, I led the whole process, with input and suggestions from any of a group of 5 or 6 students who had elected to participate. At this stage of the semester, class meetings were being held in the lab, with all class time allotted for joint project work. The problem analysis, apparatus design, measurements, and conclusion occurred over the course of about an hour, during one of these sessions. Students subsequently manufactured and installed the gears in the described solution.

Summarizing, students were able to follow the analysis of a contextualized problem that mattered, and had the satisfaction of designing a solution and seeing it work. Students commented that this was the first time they had used the physics they had learned for something practical. The exercise of constructing instruments from first principles, and combining several disparate bits of physics knowledge (mass of water, Hooke's Law, and a basic statics analysis) was an exciting process of connecting ideas that had previously seemed separate and inert.

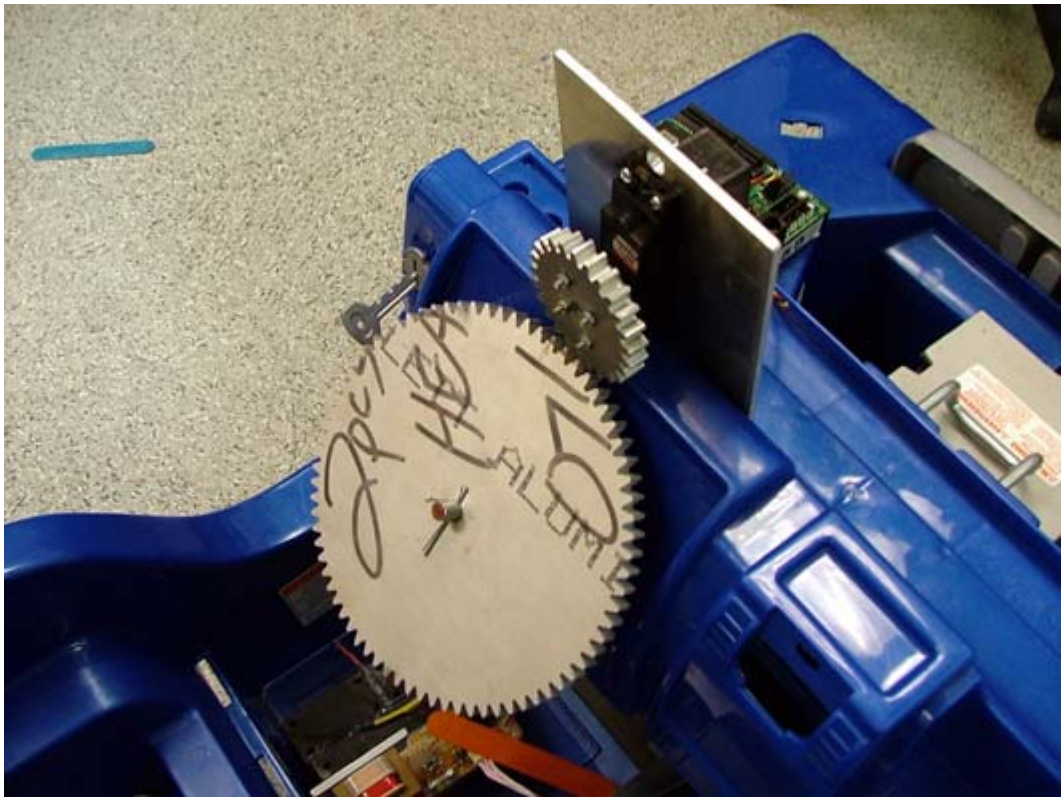

Figure 4. Resulting steering system for Power Wheels robot.

3. LIGHT-SEEKING IS HARDER THAN EXPECTED

In most algorithms that computer science students learn, data are either perfectly known or already modeled. So it is a surprise when an algorithm that should be simple turns out otherwise. In this example, the details are based on a 1-sensor light-seeking algorithm, but in many robotics applications, one's initial model of sensor data turns out to be inadequate. Thus, the lesson here is general.

The example is taken from an intensive robotics design workshop I conducted at the Universidad Autónoma de Yucatán in Mérida, Mexico. The Department of Mathematics had created an engineering-oriented undergraduate program in computer science named "Licenciatura en Ingeniería en Computación" [15]. Students take a rapid prototyping class during their freshman or sophomore year. As part of this class, I conducted a week-long robot design course for about 20 students during the Spring 2005 semester, using LEGO Technic, the Handy Board, and Interactive C [13].

I gave the students a simplified version of the Case-Western Egg Hunt competition, which involves transporting plastic eggs to a goal that is marked with a light source [2]. A natural sub-problem of the contest is to build a robot that can perform light-seeking. Most students had two-wheel, "wheel chair" style robots, and I encouraged them to work with one light sensor, as shown in Figure 5.

For the sake of expediency, I provided students with a sample algorithm for performing light seeking using a robot with this configuration. The algorithm was:

1. Turn around (for at least a complete circle), capturing the strongest light reading while doing so.
2. Turn until that value is attained again, and stop.
3. Drive forward.

The Interactive C code describing this algorithm is as follows:

```
void main() {
  find_strongest_light();
  turn_to_strongest_light();
  go_forward();
}
```

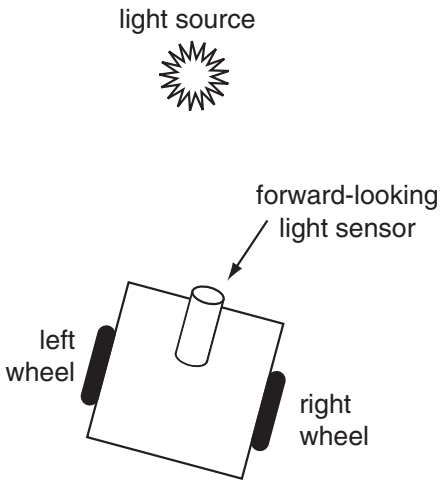

Figure 5. Two-Wheeled Robot with One Light Sensor

(Please note that this is only a partial solution, as one would need iterate over these three steps in order to obtain complete light-seeking behavior.)

Since many of the students were novice programmers, I further gave them example code for finding the strongest light, since this code made use of some timing features that would not be obvious.

Here is the code. Please note that stronger light causes *smaller* sensor readings:

```
int min;

void find_strongest_light() {
  float now = seconds();   // used to time a period of turning
  int current;             // capture a sensor reading
  min = 255;               // initialize min to high value

  //make the robot start turning
  while (seconds() - now < 5.) {  // turn for 5 seconds
    current = analog(2);   // read photocell on port 2
    if (current < min) min= current;  // capture sensor value if
                                      // it's less than best previous
  }
}
```

In the discussion that followed, one student pointed out a subtlety in the details of the algorithm to capture the smallest reading. A more naive way of writing the loop would be:

```
  while (seconds() - now < 5.) {  // turn for 5 seconds
    if (analog(2) < min) min = analog(2);
  }
```

This implementation has the (not so obvious) bug that the sensor reading can change between the point where it is discovered to be less than the previous minimum and the point where it is subsequently captured. I had hoped to simplify students' work by giving them the "more correct" algorithm (and not explaining this case), but at least one of the students had experienced the problem before, and pointed out that my code was already improved.

As the lecture/discussion was wrapping, I gave students specific process guidance to finishing the algorithm I had provide. I told them that they should ascertain that the `find_strongest_light()` routine was working before beginning the `turn_to_strongest_light()` piece. In other words, they should first implement

```
void main() {
  find_strongest_light();
  stop_robot();
  print_minimum_reading();
}
```

```
//  turn_to_strongest_light();
//  go_forward();
//}
```

After students had this working—or maybe before?—they set out to write their `turn_to_strongest_light()` routine. Half of the eight groups had code with an identical bug, equivalent to the following:

```
void turn_to_strongest_light() {
  //make the robot start turning
  while (1) {
    if (analog(2) == min) break;  // if current reading equals
                                  // previous minimum, break
  }
  // make robot stop turning
}
```

The conceptual error lies in the conditional test. These students were checking if the current light sensor reading was *exactly equal* to the previously captured reading. In actuality, it is possible for consecutive sensor readings (taken as the robot is rotating) to straddle the previously captured "best" reading. In practice, this would cause the robot to keep spinning and spinning, even though it could be registering sensor readings that were stronger than the previously captured reading.

This is a classic error that students first working with real systems make. Even experienced programmers are used to designing algorithms that work in a perfect world, and the idea of sensors with sloppy, unreliable values is foreign. So, in this case, the routine should be re-written as:

```
void turn_to_strongest_light() {
  //make the robot start turning
  while (1) {
    if (analog(2) <= min) break;  // if reading equals or bests
                                  // previous minimum, break
  }
  // make robot stop turning
}
```

At this point, I thought we were done debugging the light-seeking procedure, and that we could all move on to other things. But I was mistaken. As work on the robots proceeded, one team discovered a yet more subtle way even this improved algorithm can fail.

This team was the strongest and most experienced group of students. A member of the team was the student who had pointed out my optimization at the beginning of the conversation. They noticed their robot's performance was unreliable, though it was one of the better-constructed robots in the room. They realized the light-seeking algorithm would fail when the robot *was started directly facing the light*.

Ultimately, they figured out what was happening. If the robot started off directly toward the light, the algorithm would capture an extra-strong sensor reading that resulted from the robot *standing still at the very beginning of the algorithm's execution*. In that case, the algorithm would capture a reading that would not be attainable when the robot was moving!

When the group first described the problem to me, I thought, "OK, the simple solution is just don't start your robot when it's facing the light." But this group had higher aspirations. They solved it by first starting their robot moving, and delaying the onset of the algorithm until it was turning at full velocity—a much more robust solution.

This example and the progression of our understanding of light-seeking illustrates the complex ways in which code interacts with the world when it is embedded in a mobile robot. The value of the robotics approach is to provide a context where these lessons are both meaningful and fairly easily observable. The central lesson of this example is that it is difficult to model the real world, particularly when dealing with dynamic components. Of course, there are much more sophisticated versions of this notion, but this lesson can be meaningfully experienced even with a simple mobile robot.

It is worth noting that this issue is applicable well beyond robotics. For example, in operating systems design, the atomic test-and-set operation is a crucial concept. The issues in test-and-set are analogous to those in the sample-sensor/capture-sensor problem.

Finally, I will also point out that students do not often get the opportunity to directly experience this issue in the computer science core curriculum. Most algorithm design problems involve known data

structures that are not subject to corruption. Computer science students are primarily introduced to systems that are provably correct from a mathematical standpoint.

For any of us, it is a valuable and humbling experience to discover that the real world does not easily lend itself to systematic modeling.

4. EVOLUTION OF THE CONTROL ARCHITECTURE FOR A MULTI-STUDENT ROBOT

This section presents the evolutionary design of the control architecture for the Power Wheels robot discussed earlier in the context of the steering mechanics. The overall robot is shown in Figure 6. As of this writing, work on the robot has taken place over three semesters with three different groups of students (with a few individual students overlapped across semesters). A primary lesson in this project has been the story of how the design of the robot adapted to become compatible with the work styles of the students who developed it. In particular, we emerged with a modular design that allowed individual students (or small teams of 2 to 3 students) to have ownership of a particular subsystem (say, the motor control system, or the sonar sensor system) but have their work be easily shared with others on the project team.

Figure 6. The MCP Robot.

Broadly, the project went through three stages of work, corresponding to three semesters of student activity. In the first semester of work (Fall 2003), an individual Handy Board separately controlled each of the robot's subsystems. The Handy Boards communicated with each other using a custom master-slave protocol called "Cricket Bus" [14]. One Handy Board acted as the master controller and issued commands over the Cricket Bus to the other slave Handy Boards. The robot got its name, "MCP," from this initial design—**M**ultiple Handy Board **C**ontrol **P**rogram. This design is illustrated in Figure 7.

Teams of 1 to 3 students implemented each of the subsystems, but in practice, we only got two Handy Boards interfaced with each other. The Handy Board that operated the sonar sensors acted as the master, and issued commands to the Handy Board that operated the steering and drive motors.

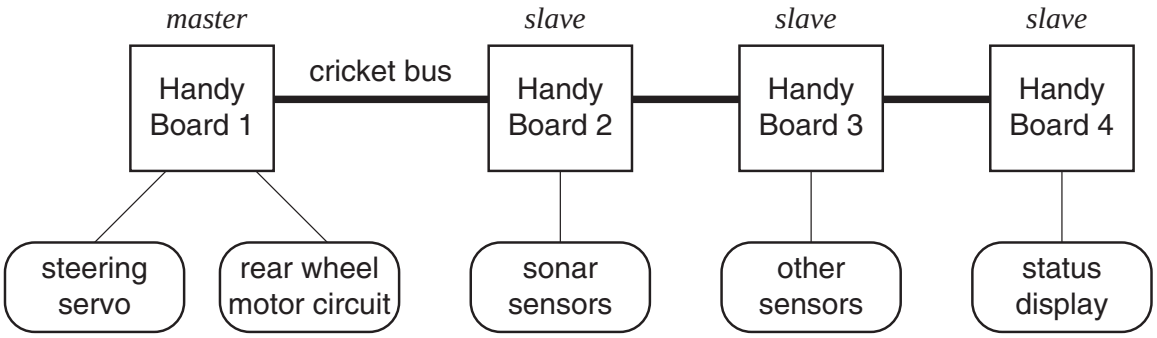

Figure 7. MCP Robot Control Architecture, First Semester.

There were several challenges in getting the robot to work with this approach. The robot had a complex "launch sequence" of turning on Handy Boards in a particular sequence and getting them to run their code. Each student team used a different computer in the lab to develop code for their subsystem, so students could not easily view each other's code, and it was hard to know which version of the code on any given desktop was the "correct" version. Also, each Handy Board needed to have its battery charged separately. Overall, the system was brittle.

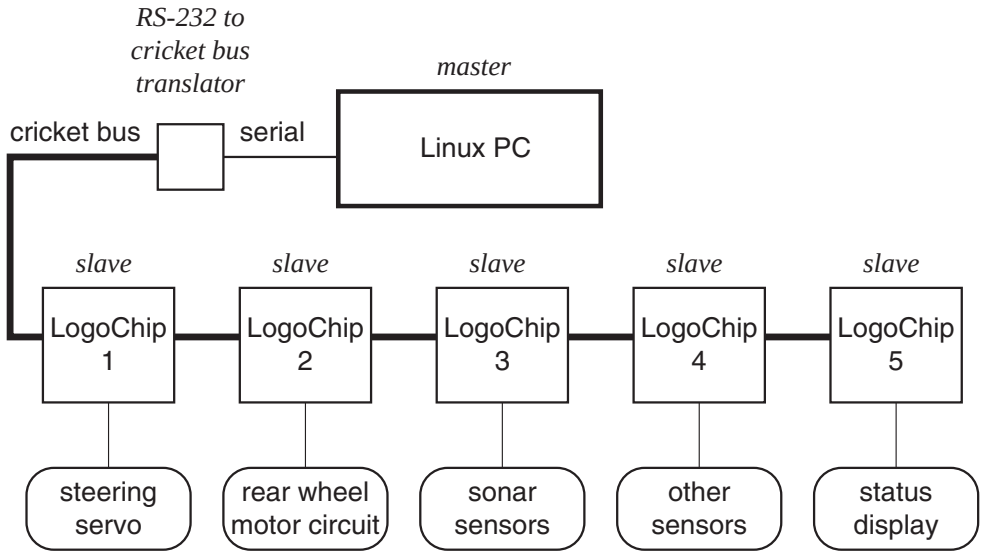

Figure 8. MCP Robot Control Architecture, Second Semester.

In the second semester of work (Fall 2004) I started a new group of students on the project. Based on the previous work, I encouraged students to re-design the Handy Board-based subsystems to employ a simpler, single-chip microprocessor technology called the "LogoChip" [3]. The idea was to remove the complexity associated with having multiple Handy Boards, each with its own program that had to be loaded separately. The LogoChip device had native drivers for the Cricket Bus and used persistent internal memory (so it could not lose its program because of loss of power). So the plan was to have each student team develop and debug a reliable subsystem that would respond to commands over the Cricket Bus. Finally, we planned to add a Linux PC as the master control computer, which would talk over the Cricket Bus using an RS-232-to-Cricket Bus translator. This design is illustrated in Figure 8.

The LogoChip/Cricket Bus design did not make as much progress as I had hoped. When the semester ended, the project teams had individually gotten their subsystems to work, but only in bench testing. They had not gotten to the stage of programming the Cricket Bus communications code, so the separate projects were not even ready to be integrated.

In retrospect, I did not provide students with a method for readily debugging Cricket Bus communications code. Effectively, each team would have needed two LogoChip systems—one for their own subsystem, and one to act as the master so they could debug their communications code.

This became evident as worked moved to the third semester (Spring 2005). A new student picked up the LogoChip/rear wheel drive subsystem, and advanced it to the point where he developed the communications code. But rather than developing it using the Cricket Bus, he wrote a communications protocol that worked over standard RS-232 serial, and tested it using his own development PC to generate control commands.

Separately, another student was working adapting a scrolling LED sign to be a large output display for the robot. The sign had a serial interface, but we wanted any process running on the Linux PC to be able to send a message to the sign. The student then developed a UDP-based client-server system for sending messages to the sign. A sign server program opened the serial port and communicated at the low level with the sign. Also, the server listened on a particular network port for incoming messages. The student wrote a separate client program that would generate a UDP packet to the server, containing the message to be displayed. The server would then receive the packet and serially transmit the text to the sign.

One other development figured crucially into our architectural redesign. A student in our lab had developed a robotics interface board called "SerialSense," which made it easy to control motors and read a variety of sensors using a serial line. Along with the hardware, the student developed a C++ class library that made the board simple to use from a development standpoint [4].

At some point in the semester, it just became obvious that we should connect everything to the robot using serial lines and the SerialSense board. We got the Linux PC mounted into the robot, and acquired a USB-to-8-serial port expander. Some devices connected directly to the robot using serial (the LED sign and a global positioning system [GPS] unit), while others were interfaced through SerialSense boards (Figure 9).

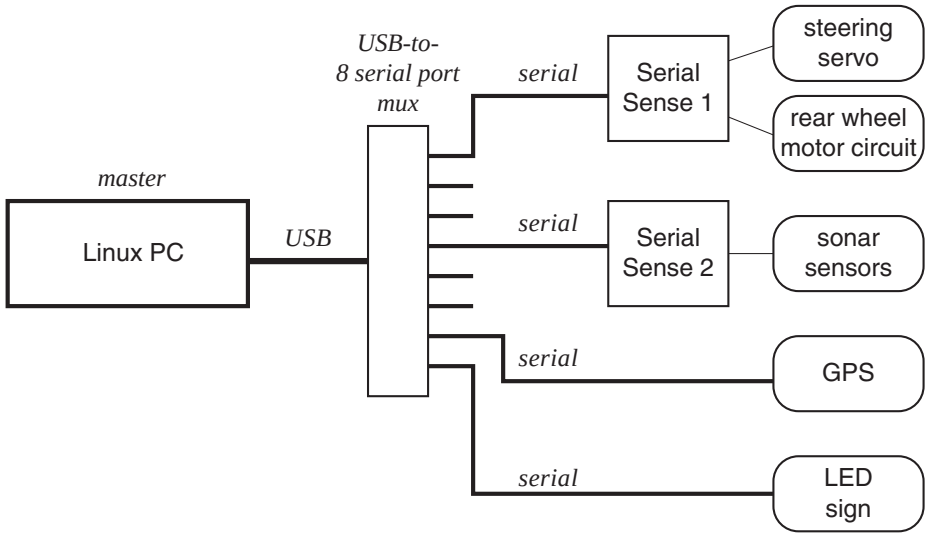

Figure 9. MCP Robot Control Architecture, Third Semester.

From a development standpoint, the approach was transformative. After getting their hardware designs working, each student team could then set out to develop its software drivers, effectively "owning" the whole subsystem. At one point in the semester, there was one group working on the steering, another student working on the drive wheels, and a third team working on the sonar system. Each group had its own SerialSense board, plugged into their own serial port on the robot's Linux computer. All groups could be simultaneously logged in to the Linux machine, developing code in parallel. Further, it became trivial to share code and learn from each other's work—all the code was literally on the same machine. The steering and drive wheel subsystems were later merged onto one SerialSense board. In an instance of the value of code sharing, the steering and main drive subsystems, which were initially developed by different groups, were easily merged when needed.

Inspired by the approach of the LED sign project, we employed local communications to integrate software subsystems. As with the sign server, the steering and rear wheel drive software was written as a server, and accepted movement commands over a local UDP port. The sonar subsystem created a block of shared memory, into which it continuously wrote sensor readings; other Unix processes attach this shared memory segment and read the sonar values. The GPS subsystem was based on an open-source server that uses the same technique. This strategy allowed each team's code to run as a separate process that did not need to be compiled into a monolithic control program.

5. DISCUSSION

In an essay that described his motivations in creating the MIT Mechanical Engineering undergraduate design competition, Woodie Flowers noted that the design problem should be different every semester, so that it is a genuine process of discovery for all involved [7]. Neither students nor faculty know which solutions are likely to be best, or precisely what problems will be encountered. Of course, faculty have an advantage over students in terms of experience, and formal knowledge, but if the specific problem formulation is new to faculty as well, there will be an edge of authenticity—including the possibility of failure—that is not present when faculty run the same challenge over successive semesters.

This authenticity permeates the work discussed in this paper. In each of the three examples, I had no particular prior knowledge that would have allowed me to "short circuit" the process and arrive directly at an "answer." Students knew that I was just as genuinely searching for understanding as were they, and we respected each other because of it. In particular, drawing on the narratives that have been presented:

- In the development of the steering system, students initially tried an experimental approach, and when this failed, we invented an apparatus for performing standards-based measurements, and applied a combination of knowledge from basic physics to perform the analysis. After the analysis, we developed a new design, which was then tested as operational.
- In the light-seeking study, students implemented light-seeking algorithms, and along the way, discovered various unexpected interactions between their algorithms and the real-world.
- In the evolution of the robot control architecture, a series of overall control architectures were developed, leading to a design that allowed students to easily modularize their own projects and learn from each other's work.

Additional themes are common across all of these projects. Work and learning occurred as we interacted with ideas-in-progress. Rather than leading with a formal analysis, we began with our initial understandings, made an implementation based on this, and then iterated. As mentioned, this is the studio design style of projects that "undergo multiple and rapid iterations." Further, analysis was brought into play as needed; as we discovered problems that were amenable to formal techniques, they were applied. Papert and Turkle have referred to this as "logic on tap, not on top" [17].

Another key theme is how students share work and learn from each other. This is perhaps strongest in the third example of the robot control architecture. The primary failings of the early designs were not so much technical as they were in how they limited sharing of work. The crucial innovation was the Linux PC mounted into the robot, which allowed multiple students to simultaneously log-in, perform work, and examine each other's work. Further, the modularization of hardware, firmware, and Linux-based driver software allowed a student or team to own a whole sub-system, yet readily share it with others.

My role as facilitator, instructor, mentor, and collaborator was also crucial in each of these case studies. Depending on the situation, I presented new material, worked alongside students, debugged problems, or steered their thinking in particular directions. This is a significantly more complex role than the one required in a lecture style classroom (and one that I personally enjoy immensely).

I will close by noting the central role of a laboratory environment. In all cases, students learned by seeing each other's work as they individually or jointly got things to work. But the lab is more than this; it is a shared social space in which students (and faculty) get to know each other as people. Casual conversations lead to trust, serendipity, and the reward of building something that you care about with people who you care about.

ACKNOWLEDGMENTS

Most importantly, I would like to thank the students whose work I have discussed in this paper. It has been a pleasure to work with each and all of them.

Three anonymous reviewers and the editors of this issue (David Ahlgren and Igor Verner) provided critical and valuable feedback on earlier drafts of this paper. I greatly appreciate their contributions.

I also would like to thank my department and the University of Massachusetts Lowell for providing funding for the equipment and materials used by students in my lab, and Alberto Muñoz, who brought me to his university in Mérida, Mexico for the workshop with his students.

REFERENCES

1. C. S. Ananian, PCB Mill web site, http://sinfor.lcs.mit.edu:8180/pcbmill/index.html.
2. R. D. Beer, H. J. Chiel, and R. F. Drushel, "Using autonomous robotics to teach science and engineering," *Communications of the ACM*, volume 42, number 6, 1999.
3. R. Berg, B. Mikhak, and B. Silverman, LogoChip, http://www.wellesley.edu/Physics/Rberg/-logochip/
4. A. Chanler, SerialSense, http://www.cs.uml.edu/~achanler/robotics/serialsense/, 2005.
5. E. Ferguson, *Engineering and the Mind's Eye*, MIT Press, 1992.
6. G. Fischer and E. Scharff, "Learning technologies in support of self-directed learning," *Journal of Interactive Media in Education*, volume 98, number 4, October 1998.
7. W. C. Flowers, "On Engineering Students' Creativity and Academia," Proceedings, 1987 ASEE Annual Conference, Reno, Nev., June, pp. 227–231.
8. C. Geertz, *The Interpretation of Cultures.* Basic Books, 1973.
9. S. Kuhn, "Learning from the architecture studio: Implications for project-based pedagogy," *International Journal of Engineering Education*, volume 17, numbers 4–5, pp. 349–352, 2001.
10. R. Lesh and H. Doerr (editors), *Beyond Constructivism: Models and Modeling Perspectives on Mathematics Problem Solving, Learning, and Teaching*, Lawrence Erlbaum Associates, 2003.
11. F. Martin, "Circuits to Control: Learning Engineering by Designing LEGO Robots," unpublished PhD dissertation, MIT Media Laboratory, 1994.
12. F. Martin, M. Hjalmarson, and P. Wankat. "When the model is a program." To appear in volume entitled *Foundations for the Future in Mathematics Education*, Richard Lesh, editor, Lawrence Erlbaum, 2006.
13. F. Martin, *Robotic Explorations: A Hands-On Introduction to Engineering.* Prentice-Hall, 2001.
14. F. Martin, B. Mihkak, and B. Silverman,, "MetaCricket: A designer's kit for making computational devices," *IBM Systems Journal*, volume 39, number 3–4, 2000.
15. L. A. Muñoz and A. Espinosa, "Ingeniería en Computación: entre constructivismo e investigación científica," III Congreso Internacional de Informática y Computación ANIEI, 20–22 Octubre, Tépic, Nayarit, México, 2004.
16. S. Papert, *Mindstorms: Children, Computers, and Powerful Ideas*, Basic Books, 1980.
17. S. Turkle and S. Papert, "Epistemological pluralism and the revaluation of the concrete," *Journal of Mathematical Behavior*, volume 11, number 1, March 1992.

ABOUT THE AUTHOR

F. Martin is an Assistant Professor in the Department of Computer Science at University of Massachusetts Lowell. Martin directs the Engaging Computing Lab, which develops technology that enables children, teachers, engineers, and artists to design interactive, embodied computational systems. He was co-founder of the MIT Autonomous Robot Design Competition and designer of the "Handy Board" microcontroller, inspiring similar courses based on this technology at universities and colleges worldwide. Martin's work on educational robotics for children led to the launch of the commercial LEGO Mindstorms Robotics Invention system in 1998. Based on his educational robotics materials, Martin has led teacher professional development workshops with K–12 teachers across the USA and in Mexico, Brazil, Thailand, Ireland, Spain, Germany, and Costa Rica. In 1995, Martin co-founded Gleason Research, which distributes his educational technology to schools, colleges, universities, and hobbyists.