Launching into AI's *October Sky* with Robotics and Lisp

Frank Klassner

■ Robotics projects coupled with agent-oriented trends in artificial intelligence education have the potential to make introductory AI courses at liberal arts schools the gateway for a large new generation of AI practitioners. However, this vision's achievement requires programming libraries and low-cost platforms that are readily accessible to undergraduates and easily maintainable by instructors at sites with few dedicated resources. This article presents and evaluates one contribution toward implementing this vision: the RCXLisp library. The library was designed to support programming of the Lego Mindstorms platform in AI courses with the goal of using introductory robotics to motivate undergraduates' understanding of AI concepts within the agent-design paradigm. The library's evaluation reflects four years of student feedback on its use in a liberal-arts AI course whose audience covers a wide variety of majors. To help establish a context for judging RCXLisp's effectiveness this article also provides a sketch of the Mindstorms-based laboratory in which the library is used.

In the movie *October Sky,* a group of high school students become inspired to learn how to build high-altitude rockets after watching Sputnik glide across the night sky soon after it was launched in October 1957. Today AI should be poised to capture students' interest and imaginations in the same way that movie showed one application in physics and astronomy capturing them in the 1950s and 1960s. Look at what today's undergraduates are seeing and hearing about AI in popular culture. Besides the popularity of the AI-inspired fiction in the movies *I, Robot* and *A.I.,* consider the highly publicized success of the Mars rovers *Spirit* and *Opportunity,* the ESPN coverage of the DARPA autonomous vehicle Grand Challenge, the lust among gamers after cleverer computer opponents, and the prevalence of word processor speech-recognition systems. Students are not just hearing about AI applications—they are experiencing them more directly than did the students in *October Sky* gazing up at Sputnik's starlike dot. Today's college and high school students can evaluate AI applications firsthand (for example, in games, robotic vacuum cleaners, and intelligent search engines). More importantly for AI, the immediacy of their experience often makes them feel they could replicate or even improve the applications' capabilities—if only they understood the AI theory behind the application.

And this situation definitely is enticing students into trying out introductory AI courses at liberal arts colleges. The difficulty for instructors at such schools is retaining these students' interest in the field after their first exposure to formal AI. In many smaller schools' computer science departments there is at most one faculty member with AI training, usually with few dedicated resources. Instructors from such schools have attested at AAAI and SIGCSE sym-

posia that they must tread carefully between a breadth-first survey style that downplays full system implementation experience and a depth-first concentration on subfields that might not directly relate to popular applications without exorbitant software or hardware investments. A heavily unbalanced course, in either direction, can leave undergraduates with unrealistic (or worse—nonexistent) views of how real-world AI systems can be designed and implemented to meet real-world performance goals.

Fortunately, three trends are helping to maintain the balance that AI education needs for inspiring students from smaller schools. The first is the growth of the intelligent agent design framework (Russell and Norvig 2003) that emphasizes study of the limits and capabilities of different AI techniques in producing actions appropriate to an agent's environment. Because it stresses identification of the weaknesses as well as the strengths of these techniques, this framework also naturally encourages exploration of how to integrate two or more AI techniques (as might be covered in a more breadth-first oriented course) into hybrid systems so as to mitigate their individual weaknesses.

The second trend is the increasing availability of low-cost robotics platforms[1] that can play the same motivating role in AI education that model rockets and toy telescopes did for astronomy and physics education in the past century. Pure software-oriented projects such as gaming agents or simulations can indeed motivate students, but there is no denying the increased registration levels seen in elective AI courses the first time a new robot-development component is announced for them.

The third trend is vital to the success of the second. It is the growing body of open-source software systems and programming tools for robot kits that allows students with limited resources to move quickly from learning about AI techniques in the abstract to actually building artifacts with them (Yuasa 2003).[2] It is not enough to give students a robot platform that requires a whole semester to learn how to program. After installing the platform's programming tools, students should be able to get past "Hello Robot" trials quickly in order to grapple with moderately complex programs with confidence-boosting results. From a small-college instructors' perspective, the most effective elements in this software collection allow them to extend the depth of their courses' coverage in various ways. Those instructors who want to encourage students' deeper understanding of a topic through attention to implementation efficiencies can supply partial source code for AI systems (like planners or search algorithms) for students to complete. Those who want to encourage students' deeper understanding of a topic through attention on "knob-turning" experiments (Cohen 1995) can supply completed source code sets from the same corpus.

All three trends have strongly influenced the AI curriculum at Villanova University. In the fall of 1997 the introduction to AI course CSC 4500 was reorganized around the concept of agent-oriented design. This approach was augmented in the fall of 1999, offering with team-based Lego Mindstorms (Klassner 2002) projects in order to have students explore the relationships among hardware, environment, and software organization in agent design. Since the spring 2001 course we have used the RCXLisp library developed at Villanova for programming Mindstorms in Common Lisp. This article describes the RCXLisp library with respect to these trends, focusing on its development, its uses in CSC 4500, and student feedback on both the course and its library. To help the reader evaluate RCXLisp's effectiveness, this article also provides a sketch of the Mindstorms-based laboratory in which the library is used.

Choosing Mindstorms and Lisp

The CSC 4500 course is an elective with no formal programming prerequisites. Computer science majors typically take the course in their fourth year, by which time most of these students have taken a programming languages course that introduces them to Lisp. The course is also open to third- and fourth-year cognitive science concentrators and computer engineering majors, who generally have no programming experience in Lisp and as little as one semester of introductory programming in Java from their first year. The cognitive science students hail from a variety of majors: biology, political science, and physics, to name a few.

The course uses Common Lisp in its programming projects for two reasons. The first is to leverage Lisp's support for rapid prototyping against the large body of open-source AI Lisp systems. The second is to reinforce computer science majors' Lisp experience from their programming languages course, thereby offering students another option besides C/C++ for meeting the department goal that graduates be familiar with at least one language beyond Java.

Once the decision was made in 1997 to organize CSC 4500 around the agent paradigm, assignments using robotic agent simulations

were tried but did not generate significant enthusiasm among students. Based on student feedback and positive reports on robots' use in the computer science education literature (Beer, Chiel, and Drushel 1999; Kumar and Meeden 1998; Mazur and Kuhrt 1997; and Stein 1996), the decision was made to try using physical robots in the course. The choice of robot platform was based on four criteria: programming environment, cost, flexibility, and "student comfort level." In 1999 there was no low-cost robot kit that supported Common Lisp programming, so this criterion played a negligible role in the decision. A Mindstorms kit with 750 construction pieces, sensors, and programmable hardware cost approximately US$200. This was one quarter the cost of some Handy Board–based robot kits and one tenth the cost of an ActivMedia-based robot kit—two of the other platforms considered. In the time since this evaluation was conducted Lego has released a "RoboChallenge" kit for US$150 that has fewer construction pieces (250) for courses where a sophisticated robot chassis is unnecessary. In terms of flexibility the Mindstorms kit supports the creation of effectors such as arms, grippers, legs, tread casings, and so on, often not available in other basic kits. The Handy Board platform, however, was similarly extensible. The final decision hinged on the "student comfort level" criterion. Many students had played with Lego as children and felt more familiar with those kits. The Mindstorms platform's reusable snap-together sensors and effectors allowed computer science majors to focus more quickly on programming issues. Finally, compared with an uncushioned Handy Board, Mindstorms was judged to be more durable in the face of inevitable drops and bumps in a computer lab setting.

At the time of Mindstorms' adoption, there was no Common Lisp environment for Mindstorms, and LeJOS—a popular Java environment for Mindstorms—was still under development and not complete enough for classroom use. There was a Scheme compiler targeted to Mindstorms' firmware, but it was a Unix-based utility not integrated into any Lisp programming environment. This situation led to the decision to use Not Quite C (NQC) for team-based robotics projects and Common Lisp for separate course projects in search, planning, and machine learning. NQC was judged to have a syntax that would be recognizable by most students coming into CSC 4500, yet would hide certain complexities like pointers and dynamic allocation from nonmajors.

Student surveys, however, indicated that the burden of working with two different languages distracted students from learning the AI concepts of the course. We also found the language had little support for coordinating teams of robots, though this problem is due more to NQC's dependence on Lego's firmware than to any design flaws in NQC itself.

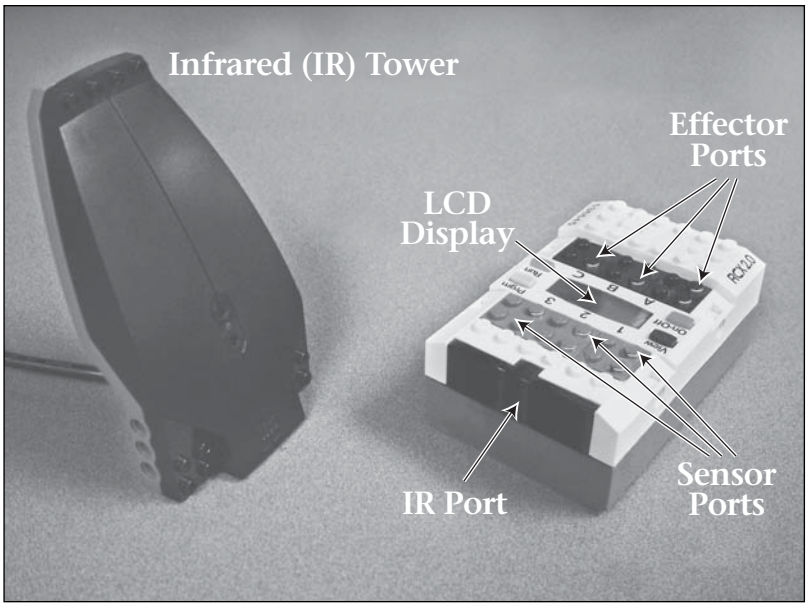

Figure 1. The Lego Mindstorms Infrared Transceiver Tower (Left) and the RCX Unit (Right).

Extending Mindstorms for a Lisp Environment

Against this historical background the decision was made to develop the Common Lisp library RCXLisp for programming the Mindstorms RCX microcontroller. The RCX is the programmable yellow brick at the heart of the Lego Mindstorms kit, shown on the right in figure 1.

The RCX has a 16-megahertz central processing unit (CPU) (the Hitachi H8/3292 microcontroller) and 32 kilobytes of random-access memory (RAM), and it houses an infrared (IR) transmitter/receiver for sending and receiving data and commands from a desktop PC or from other RCXs. The IR transceiver has a range of 15–25 feet, depending on lighting conditions. The RCX is powered by six AA batteries. The platform supports both serial-port and universal serial bus (USB) IR towers for broadcasting messages and software to the RCX from a desktop (a USB IR tower is shown on the left in figure 1). RCX units have three input (sensor) ports, three effector ports, and a five-character liquid crystal display (LCD) display. Lego markets touch, light, temperature, and axle-rotation sensors that can be connected to the RCX's input ports; third-party vendors offer magnetic

Control an RCX from a Common Lisp program on a desktop computer.
Create and compile RCXLisp programs for downloading to RCXs on t he fly," from within Common Lisp desktop environments.
Control more than one RCX from a single MindStorms IR Tower simultaneously.
Set up networks of RCXs that can communicate with each other in a targeted manner.

Table 1. RCXLisp Features That Support Integration of Common-Lisp AI Systems with the Mindstorms RCX.

compass sensors, ultrasonic distance sensors, and infrared distance sensors. Motors, light-emitting diodes (LEDs), and infrared emitters can be attached to the effector ports. Lego also offers a computer-tethered video camera for use with the RCX.

The 32 kilobytes of nonupgradeable RAM on the RCX represents a significant limitation for AI robotic agent projects involving schedulers, planners, or neural networks with large footprints. However, this can be mitigated by judicious use of the system's firmware.

The Lego-supplied firmware models a primitive virtual machine with support for threading and 16-bit integer arithmetic. There is no call stack; programmers must watch for accidental unbounded recursion problems. There is also no dynamic allocation support. A key feature of the firmware is that it supports two styles of program control: on-board programming and remote-control programming. In the first style the robot's behavior depends only on the program loaded into its memory. In the second style a control program on a desktop computer broadcasts individual instructions to the robot for immediate execution.

Thus, it could be possible for large AI systems (with accompanying needs for call stack and dynamic allocation support) to be used with Mindstorms as long as they are modified to control RCX robots remotely from a desktop computer. Unfortunately, Lego firmware uses a broadcast protocol for sending control messages to RCXs. It does not support targeted message passing. It therefore is not possible to have several teams of students in the same lab control their own robots without interference. It is also not possible for a single desktop application to coordinate several robots of a team without a mechanism to address the robots' RCXs individually.

To address this communication shortcoming, a firmware version named Mnet (for Mindstorms networked) was developed at Villanova University. Mnet replaces Lego's standard firmware. It supports directed IR communications by extending the Lego protocol with source and target fields and by allowing each RCX and host desktop to set a 1-byte ID value for itself. Mnet firmware also allows an RCX to restrict the RCXs from which it will accept messages. To support compatibility with other programming environments targeted to the RCX, Mnet recognizes the same byte-code set as the standard Lego firmware.

Mnet does not support garbage collection or call stacks. This decision was made to speed up firmware development and to keep the firmware footprint reasonably small. Lest the lack of garbage collection seem too limiting for RCXLisp, it should be noted that LeJOS, too, does not currently implement a garbage collector in an effort to keep the JVM footprint as small as possible (Cohen 1995). Our RCX-based lisp system, RCXLisp, developed atop this Mnet firmware.

RCXLisp Language Design

The first goal of RCXLisp was to help AI students start quickly with designing basic software for controlling individual robots and robot teams. It achieves this by augmenting the Common Lisp language with an application program interface (API) of primitive functions that represent a programmer's, rather than an engineer's, view of a robot and its communication abilities. It adheres as closely as the Mnet firmware permits to the Common Lisp specification of the second edition of Steele's *Common Lisp: The Language* (1990).

The second goal of RCXLisp was to support integration of Mindstorms with sophisticated AI systems. Since such integration will naturally produce distributed systems, this goal included the ability to coordinate processes on the RCX with processes within a Lisp desktop environment. Table 1 summarizes the features that RCXLisp provides in support of this goal.

The RCXLisp libraries are currently supported in the LispWorks environments on both Windows (98 through XP) and Mac OS X. It is also available for use on Digitool's MCL for the Mac platform, but currently only for Mac OS 9.

A version for Allegro (Franz) is under development.

RCXLisp has both a remote and local component. The first is remote RCXLisp, a collection of macros, variables, and functions for remotely controlling RCX units from a desktop. The second is local RCXLisp, a subset of Common Lisp that can be cross-compiled to run on both Lego and Mnet firmware for on-board control of the unit.

The next two subsections detail these two parts of RCXLisp, with the aim of showing the tight integration among the Lisp environment, the RCX, and the desktop control processes the library makes possible. The example code supports the claim that RCXLisp facilitates linking AI systems with Mindstorms robots. The discussion occasionally makes comparisons between RCXLisp functions and Common Lisp functions of the same name. For clarity's sake all RCXLisp functions will appear in italics.

Remote RCXLisp

The remote RCXLisp library provides *with-open-com-port* and *with-open-rcx-stream* macros to set up communication environments for contacting RCX units. These macros are modeled closely on Common Lisp's with-open-stream macro. *With-open-com-port* specifies the communication port (serial or USB) for contacting an RCX, and *with-open-rcx-stream* defines the data stream through a port for a specific RCX unit. The macros offer users several arguments to modify low-level port characteristics such as data transmission rate, timeout interval, and retry limit. This flexibility is important especially when ambient lighting and other environmental conditions change. The code in figure 2 shows how these macros are used and provides an example of the functions that can be invoked to control an RCX from within the macros' communication environment.

In keeping with RCXLisp's design goal of helping programmers focus on programming rather than lower-level communication issues, the language contains a *tell-rcx* macro that combines the two port and stream macros' argument lists and builds in flexible default values for port settings loadable from a site configuration file. This level of abstraction helps beginning Lispers new to macros and stream management.

Ordinarily one would type the functions in figure 2 into an editor and then load them into a desktop environment for invocation at the Lisp listener prompt as (testing). The full-speed-ahead example contains the following remote RCXLisp functions for controlling an RCX: *set-effector-state* (for controlling motors), *set-sensor-state* (for configuring input types), and *sensor* (for accessing the current reading from a sensor input). The testing function uses *alivep* to determine whether the RCX is in range and responding. It is important to note for later discussion that the language includes a *var* function that asynchronously queries for the value stored in one of the RCX's 32 globally-accessible registers.

All remote RCXLisp functions take an optional stream argument (for example, "r" in full-speed-ahead). One can forego repetitious typing of the stream argument, with the *using-rcx* macro as in figure 3 to define a default stream for enclosed RCXLisp functions. This macro is closely modeled on Common Lisp's using-slots macro for object-oriented programming. It also serves to define a symbolic environment in which the desktop acts as a director telling some particular RCX what to do next.

Programs that are intended to be executed on an RCX are first compiled within the RCXLisp desktop environment to Mnet byte code and then downloaded through the IR tower to the RCX. This is accomplished through the *rcx-compile-and-download* utility function in remote RCXLisp. The Mnet firmware (and standard Lego firmware) is also loaded from the desktop Lisp environment with the *download-firmware* utility function in remote RCXLisp.

It is important to note that these functions are native to the Lisp environment. That is, no non-Lisp mechanism is needed for these actions. Both LeJOS[3] and Wick, Klipsch, and Wagner's Lego/Scheme compiler[4] require a separate command-line system program to download compiled code or firmware into the RCX. The compiler in *rcx-compile-and-download* is implemented in Lisp. It is this feature that allows RCXLisp to interface cleanly with other AI systems written in Common Lisp. The *rcx-compile-and-download* utility supports on-the-fly compilation of both dynamically generated RCXLisp forms (by a planner, for example) and static forms in prewritten files for execution on a robot.

XS (Yuasa 2003) is a Scheme interpreter for Mindstorms developed after RCXLisp. It too is supported by native libraries for a Scheme desktop environment. The difference between remote RCXLisp and XS is that XS sets up the desktop as a listener and the RCX as the evaluator, while remote RCXLisp evaluates all forms on the desktop and sends byte codes to the RCX. XS enjoys the advantages over RCXLisp of a garbage collector and full call stack support on the RCX but requires higher bandwidth in

```
(DEFUN full-speed-ahead (r s dir)
"This will make the rcx in stream R go at speed S
in direction DIR until touch sensor on its '2' port returns 1."
(LET ((result 0))
   (set-effector-state '(:A :B :C) :power :off r)
            ;in case things are in an inconsistent state,
            ;turn everything off first
     (set-effector-state '(:A :C) :speed s  r)
     (set-effector-state '(:A :C) :direction dir  r)
              ; dir is eq  to  :forward, :backward, or :toggle
              ; no motion will occur until the
              ; next call to set-effector-state
     (set-sensor-state 2 :type :touch :mode :boolean  r)
     (set-effector-state '(:A :C) :power :on r)
     (LOOP ;this loop will repeat forever until sensor 2 returns a 1
        (SETF result (sensor 2 r))
        (WHEN (AND (NUMBERP result)
                        ;needed to keep = from causing error if
                        ;sensor function returns nil due to timeout.
                       (= result 1))
           (RETURN)))
     (set-effector-state '(:A :C) :power :float r))))

(DEFUN testing ()
(with-open-com-port (port :LEGO-USB-TOWER)
  (with-open-rcx-stream (rcx10 port :timeout-interval 80 :rcx-unit 10)
    ; increase/decrease  serial timeout value of 80 ms depending on
    ;environmental factors like ambient  light.
    (WHEN (alivep rcx10)
       (full-speed-ahead rcx10  5 :forward)))))
```

Figure 2. Sample Remote RCXLisp Code Illustrating Sensor-Based Motor Control.

communication since remote control is achieved through the transmission of complete ASCII expressions to the RCX for evaluation by the on-board XS interpreter.

Local RCXLisp

Local RCXLisp is a subset of Common Lisp with a few nonstandard additions that can be compiled and downloaded to run autonomously on an RCX. As with remote RCXLisp, the design goal was to follow Steele's standard (Steele 1990) as closely as possible and to maintain compatibility with both Lego firmware and the Mnet extended firmware. Yet because neither firmware supports indirect addressing or call stacks, local RCXLisp still lacks some of the functional nature of Common Lisp.

For consistency with remote RCXLisp, and to simplify the transfer of desktop Lisp code to the RCX, local RCXLisp implements all of the RCX control functions in remote RCXLisp. In the local version, however, control functions like *set-sensor-state* do not have an optional final stream argument. It is assumed that the code will only be executed on the RCX unit itself. If an RCX needs to control the behavior of another RCX unit, it can do so through integer-valued messages for the other RCX to interpret rather than by outright code transfer.

Figure 4 shows how the "full-speed-ahead" function in figure 2 would be expressed in local

```
(DEFUN full-speed-ahead  (r s dir)
"This will make the rcx in stream R go at speed S in direction DIR until touch
sensor on its '2' port returns 1."
(LET ((result 0))
   (using-rcx  r
      (set-effector-state '(:A :B :C) :power :off)
      (set-effector-state '(:A :C) :speed s )
      (set-effector-state '(:A :C) :direction dir )
      (set-sensor-state 2 :type :touch :mode :boolean)
      (set-effector-state '(:A :C) :power :on )
      (LOOP
         (SETF result (sensor 2 ))
         (WHEN (AND (NUMBERP result)
                    (= result 1))
               (RETURN)))
      (set-effector-state '(:A :C) :power :float ))))
```

Figure 3. Cleaning up Remote RCSLisp Code with the using-rcx Macro.

RCXLisp for execution on board a robot. Note that threads do not accept input arguments in RCXLisp yet, so parameters must be made into global variables. Figure 5 demonstrates how defregister can set variables' values from a desktop program.

Local RCXLisp supports analogs to a core subset of Common Lisp control expressions, Boolean operators, and data types, along with 16-bit signed integer arithmetic including the most commonly used operators. For example, just as Common Lisp allows one to use comparison invocations such as "(< 2 *x* 6)" to test for when the value of *x* is between 2 and 6, so too does local RCXLisp. It provides a limited version of the Common Lisp RANDOM function, allowing on-board experimentation with probabilistic control algorithms.

Local RCXLisp does not support floating-point arithmetic. The ability to define new keywords is also lacking, but constants may be declared with *defconstant* and global variables can be declared with *defvar. Setq* provides value-assignment, but there is currently no analog in local RCXLisp to the Common Lisp SETF macro.

Since general function calls are not supported, local RCXLisp does not have an analog to the Common Lisp DEFUN form. In an effort to support some kind of code abstraction, the language design borrows inspiration from NQC's emphasis on macros for code abstraction and includes a *defmacro* form that follows the complete semantics of Common Lisp's DEFMACRO form. RCXLisp also borrows from Brooks's much earlier L language (Brooks 1993) a desire for simplicity and memory conservation that is necessary for squeezing as much programming as possible into the small memories available on most inexpensive robot platforms even today.

The language provides two special-purpose forms that are neither in the remote RCXLisp language nor the Common Lisp language. The first form is *defthread,* which is used to define threads to run on an RCX unit. Calling this form nonstandard, however, is less of an indictment of local RCXLisp than of the Common Lisp specification itself since as of 2005 little progress has been made in formalizing threading in the language! The second form, *defregister,* allows a programmer to tie a symbolic variable name to a given register so that a remote RCXLisp program on a desktop using *var* to query a register can be guaranteed to access the intended variable value on an RCX.

Figure 5 summarizes and illustrates many of the features just described. The program beeps whenever the RCX is carried too close to a reflective object. This code makes the RCX IR port work together with a light sensor attached at port 1 in order to implement a simple proximity sensor. The signaler thread repeatedly sends out an arbitrary integer message through the IR port. When the front of the RCX gets close to an obstacle, the IR port's signal reflects

```
(defvar *dir* :forward)
(defvar *s* 5)
(defthread (full-speed-ahead :primary t)  ( )
    (set-effector-state '(:A :B :C) :power :off)
    (set-effector-state '(:A :C) :speed *s*)
     (set-effector-state '(:A :C) :direction *dir*)
     (set-sensor-state 2 :type :touch :mode :boolean)
     (set-effector-state '(:A :C) :power :on)
     (loop
        (when (= (sensor 2) 1)
             (return)))
     (set-effector-state '(:A :C) :power :float))
```

Figure 4. Local RCXLisp Code That Performs the Same Actions as "full-speed-ahead" in Figure 3.

```
(defconstant *receiver* 1)
(defregister 4 *LIMIT* 16)

(defthread (signaller) ()
  (loop
   (send-message 78)
   (sleep 25) ;; leave IR port silent for short time
              ;; in case desktop is sending message.
  ))

(defthread  (alpha :primary t)  ( )
  (let  ((diff1 0)  (diff2 0))
   (set-sensor-state *receiver* :type :light :mode :raw)
   (setq diff1 (abs (- (sensor *receiver*) (sensor *receiver*))))
   (setq diff2 (abs (- (sensor *receiver*) (sensor *receiver*))))
   (start-rcx-thread signaller)
   (loop
     (when (>= (abs (- diff1 diff2)) *LIMIT*)
        (play-tone 500 1))
     (setq diff1 diff2)
     (setq diff2 (abs (- (sensor *receiver*)
                         (sensor *receiver*)))))))
```

Figure 5. Multithreaded Local RCXLisp Sample Code.

back, and the light sensor picks this echo up. As the reflections increase in intensity, the light sensor's value jumps more wildly. The value of *LIMIT* may need to be adjusted for environmental conditions or the light sensor's sensitivity. It is declared as a register variable so that a remote RCXLisp program can modify it on the fly. Figure 6 shows the remote RCXLisp code for achieving this. The *:raw* mode setting for the light sensor means that the sensor returns uninterpreted 10-bit sensor values.

Setting up a Mindstorms Robotics Lab

The high school student protagonists in *October Sky* required more than just reliable fuel and construction materials to achieve their goal of a high-altitude rocket; they needed a testing field. Besides a control system like RCXLisp, AI students need appropriate facilities to design and test their robot agents. The CSC 4500 course has a 15 by 30 foot computer lab dedicated for its projects during a semester, but, as will be seen from the following description, the room's setup can be duplicated without too much overhead even at schools where space is at a premium and labs must be shared among courses.

The course uses team-based active-learning robotics projects. Students are grouped in threes or fours, with each team having at least one nonmajor. Accordingly, the lab has four PCs—one per team—for programming robots. Each PC has an IR tower suspended from the ceiling not only so that its signal will be in range of the group's RCXs on the floor but also so that the likelihood of someone accidentally blocking the IR signal is reduced. The room's lighting is under the complete control of students, since sometimes bright sunlight or fluorescent lighting can interfere with the IR signals used in the distributed control systems supported by RCXLisp. For some projects, it is useful to have a navigational grid on the floor, so a temporary grid of electrical tape marking out the 1–square-foot tiles on a portion of the room is put down at the beginning of each semester CSC 4500 is offered. Figure 7 gives a picture of this setup's implementation. Note the IR towers at upper right suspended from the ceiling and facing down for maximum range for remote control projects. The entire room is 15 by 30 feet and accommodates four teams of four students.

Kits are prepared for the teams to use on all projects throughout the semester. Table 2 lists the equipment in a typical kit.

The robotics laboratory also has a few extra pieces of equipment: two Hitechnic ultrasonic distance sensors, three Lego vision command cameras, and ten Techno-stuff limit switches for special projects as they arise. Hitechnic[5] is a contractor firm that manufactures Lego-compatible sensors. Techno-stuff's[6] limit switches allow one to link a touch sensor directly to a motor's power connection. In this way the touch sensor can detect when a motor-driven arm, wheel, and so on, has reached some limit of movement and stop the motor without being attached to one of the RCX's three input

ports. Thus the limit switch is an excellent resource for increasing the RCX's sensory bandwidth.

Because of the course's emphasis on programming teams of agents, the cost for one team's kit (with three RCXs) is approximately US$900, and Lego now has piece sets available that could reduce the cost to US$750. Curricula focusing on design of one robot per student team could use one Mindstorms set (with one extra light sensor, one extra touch sensor, one extra motor, and one axle-rotation sensor) per student team for approximately US$300.

```
;;;;Example 1
(tell-rcx (38 :port :LEGO-USB-TOWER :retries 3)
     (SETF (var 4) 19))

;;;;Example 2
(with-open-com-port (p :LEGO-USB-TOWER)
   (with-open-rcx-stream (s p :rcx-unit 38 :retries 3)
     (using-rcx s
        (SETF (var 4) 19))))
```

Figure 6. Remote RCXLisp Code Allowing a Desktop to Update the Contents of Register 4 on RCX with Unit ID Number of 38.

3 Mindstorms Robotic Invention Systems packages
3 Lego light sensors beyond the three in the Mindstorms Systems packages
3 Lego touch sensors beyond the six in the Mindstorms Systems packages
3 Lego motors beyond the six in the Mindstorms Systems packages
2 Lego axle-rotation sensors
1 Hitechnic magnetic compass sensor
18 rechargeable batteries
1 large lockable toolbox to hold all of the above as well as a partially completed robot (for an idea of size, see the box in the middle of the floor or the black box on the grid in figure 7).

Table 2. Equipment in a CSC 4500 Kit.

Course Project Descriptions

Since 2000, RCXLisp has been used to implement the following projects in the CSC 4500 course's laboratory. Typically three or four of the projects are assigned and completed in a given semester. Some are used to help students develop skills for the course's final robot competition (see the "Capture the Balls Contest" section). Although the majority are symbolic AI projects, there is nothing in the RCXLisp library that would prevent it from supporting numeric-level AI projects such as neural networks.

Simple-Reflex Robot Design (and RCXLisp Orientation)

This 10-day project is always assigned to show students how robots with simple stimulus-response rules and no model of the environment can achieve effective behaviors as well as demonstrate emergent behavior. Students design a robot based on a tread-wheeled "Pathfinder" model described in the Mindstorms user manual. Students start with this basic design in order to reduce time spent on distracting mechanical engineering issues, but they are free to mount sensors on the chassis as needed.

Students first build a robot that uses a compass sensor to maintain a given bearing: team 1 goes north, team 2 goes south, and so on. They next add code to monitor, using data from mounted light sensors, whether the robot was about to roll over a dark tile on the floor. In such cases the robot has to back up and/or turn to avoid the obstacle for a brief time, after which it resumes its bearing. In programming efforts where the threads for the two behaviors are well-balanced (that is, avoidance turns are kept short and forward motion on the required bearing is steadily maintained), students observe the emergent behavior of a robot performing edge-following along dark tiles. Students implement this project first using local RCXLisp and then remote RCXLisp to gain an understanding of the reaction times for tethered and untethered operation.

Robot Odometry

This two-week project's goal is to help students understand the major factors that can introduce error into a robot's internal representation of its current location in the world—an important issue in any navigation process. It also introduces them to the importance of maintaining a representation of state—in particular, the robot's position.

Each team designs and builds a robot that will measure the perimeter of a convex black shape on a light background on the floor. These shapes are printed on reusable laminated sheets (see figure 8). The reported measurement (over 200 cm) should be accurate to within plus or minus 3 cm and should be obtained within 90

Figure 7. Robotics Lab at Villanova.

seconds. The project permits use of dead reckoning (for example teams used one rotation sensor to measure the distance being traversed and a second one to estimate when the robot's orientation had completed one rotation around its body's axis) and landmark-based navigation (for example, some teams used a second RCX as a beacon to tell the robot when it had reached the point at which it had started) techniques. Although all teams generally succeed in this project, many find the time limit challenging in light of the accuracy constraint.

Eight-puzzle Solver

This two-week project has the goal of showing students that data abstractions that facilitate search can produce solution representations that are not immediately translated into control programs for hardware.

The project has two stages. In the first stage students develop a knowledge representation and Common Lisp search program to solve the eight-puzzle on a desktop. The team first tries a search formulation that defines four operators—"move up," "move down," "move left," and "move right"—for each piece in the puzzle. They find that the formulation can lead to a search algorithm with a branching factor of 32. The team then develops a set of four operators that involve conceptually moving the "space" up, down, left, or right, rather than 32 operators for moving each of the numbered tiles up, down, left, or right. The students observe that this design decision dramatically reduces the branching factor of the search algorithm (4 versus 32), leading to a faster execution time for the game-solver.

In the second stage students write a remote RCXLisp program for a Mindstorms robotic arm mechanism that moves pieces in an eight-puzzle according to the solution developed by stage 1's programming. In this stage students find that the search space reformulation trick

ultimately costs them in the complexity of their second program as it spends time translating the "move space up/down/left/right" operator list to a list of "move tile at (2,2) to (2,1)" commands.

Search-Based Path Planning

In this three-week project students design a Common Lisp search program to solve the navigation problem of having a robot traverse a field of obstacles whose corners are defined by points in a plane while trying to minimize distance traveled between its start point and its goal. Students then use remote RCXLisp to interface the search algorithm with a prebuilt robot and have the robot follow the prescribed path. This project has been used many times in AI courses with graphical simulations. But when students see even the best-designed robots incur too much location uncertainty using simple geometric dead reckoning, they gain a better appreciation for the need to augment planning with sensor-based localization and navigation techniques.

Decision-Tree Control

In this project students run a prebuilt robot with light sensors and touch bumpers on a random walk through a simple maze, recording sensor readings when an obstacle is encountered. Although the Mindstorms RCX has only three input ports, it is possible to multiplex up to three touch sensors or three light sensors on a single port. Thus a total of nine sensors can be arrayed on the robot. Students record the robot's correct action (move forward, turn left, turn right, or back up) for each situation (vector of all sensors' readings). They then develop a decision tree that tries to capture the essential sensor readings for guiding the robot through the maze. The decision tree is incorporated into either a remote RCXLisp program or a local RCXLisp program for evaluation and refinement.

Robotic Agent Planning

In this individual independent-study project, a student used a planner (Carver and Lesser 1993) to control a robot through remote RCXLisp. Some plan primitives were precompiled, and these local RCXLisp programs were loaded into the robot for execution as each primitive action completed. Other plan primitives included remote RCXLisp queries about the current state of the robot. This project has considered "Wumpus World" scenarios thus far, but could be formalized for a more intricate environment to provide students a chance to investigate both reactive and incremental planning.

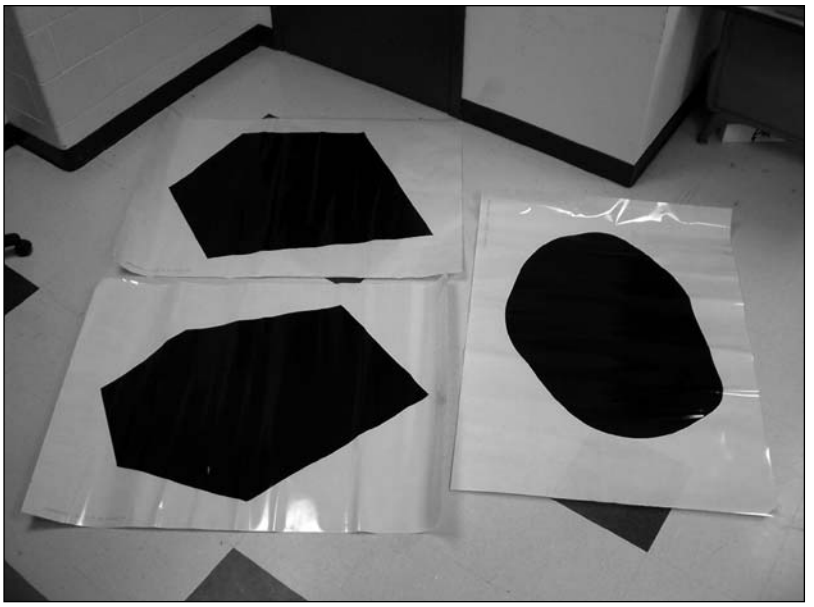

Figure 8. Example Shape Sheets for the "Robot Odometry" Laboratory Project.

Capture-the-Balls Contest

This project serves as a capstone that ties together skills students have developed through the course. It also generates an exciting finish to the semester. The contest is based on the paper by Beer, Chiel, and Drushel (1999), with some adaptations to fit the RCX's capabilities. Each student team designs and builds two robots, each no larger than 1 cubic foot, to compete against other teams' robots in a 20-minute contest. Contestants play in a walled arena that has ranged from 64 square feet to 160 square feet in area (depending on what classrooms are available each year). When large classrooms are available for the contest (that is, those that permit a playing field larger than 120 square feet), student teams are allowed to field teams of three robots.

Each robot team has a 1-square-foot dark-colored goal area. All other portions of the playing field are light-colored. Black, white, and yellow Ping-Pong balls are scattered throughout the field. Robots gather balls and bring them back to their goal for points. Each yellow ball in a team's goal earns five points; white ones add one point; black balls subtract a point. A 1-foot-square grid of half-inch-thick black tape lines the playing field. Figure 9 shows one such contest layout. To aid in navigation, the walls' corners have RCXs mounted to bathe the immediate area in IR signals uniquely identifying the region. Robots are permitted to scatter or steal balls already in opposing teams' goal areas. They are also able to

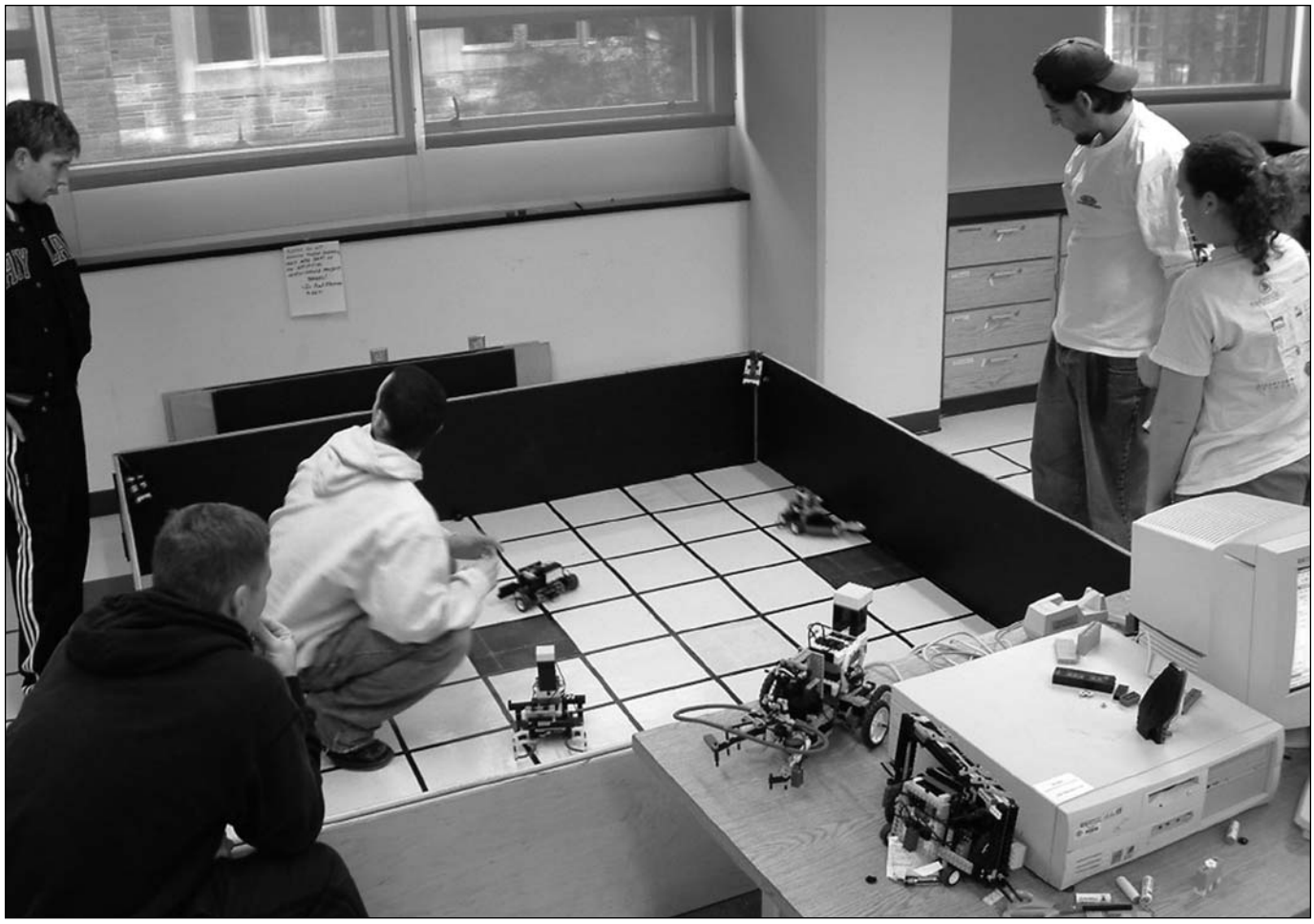

Figure 9. 2003 Capture-the-Balls Competition Setup.

place balls in opposing teams' goal areas. Robots are permitted to block opposing robots' movements physically. However, robots are not allowed to "park" closer than one foot from a goal area, and robots are not permitted to use behaviors that clearly would damage other robots (for example, ramming or piercing). Competitions allow students to dangle IR towers over the playing field for remote-control approaches.

Teams used both local and remote RCXLisp in the project. Student teams are encouraged to explore a variety of navigation techniques and play strategies. Navigation strategies employed in past competitions have included landmark-based navigation using the grid lines, hill-climbing based on the strength of infrared beacons hung over a goal square, and, most recently, sonar-based particle filtering that used the playing field's walls to aid in position estimation.

Student teams have developed many play strategies over the years. Some have incorporated time-dependent approaches. For example, as the contest progresses, robots generally push balls toward the walls of the field; some teams therefore designed ball-scooping robots with search patterns whose coverage regions expanded probabilistically toward the walls whenever too few "scoops" occur. Some teams have tried with varying degrees of success to combine elaborate planners on their control desktop with very limited planners on board their robots for recovering from breakdowns in remote-control IR communication that happen when robots' RCXs lose line-of-sight contact with the control desktop's IR tower.

When three-robot teams were permitted (only twice in the last five years), some student teams have attempted to use distributed-agent approaches whereby one of the robots is designed as an "information gatherer" that periodically traverses the playing field to find where concentrations of balls are located and

reports this information over IR transmissions to "ball gatherer" robots that react dynamically to the news.

Because of the public attention the contest draws, students become invested in the competition and try their best in designing their robots and control code. However, there was a tendency during the first three competitions for student teams to focus less on programming general AI techniques into their contest entries and to rely more on "contest engineering" approaches limited to reflexive agent designs. To reduce this tendency, since last year students are required to include in their final report a description of how their design uses at least one AI technique discussed in class or adapted from Internet research.

Student Experience Reports

The addition of robots with RCXLisp to CSC 4500 has had several noticeable impacts on students' quality of experience in the course.

Before 1999 the CSC 4500 course was offered once every two years, with enrollments rarely reaching the level of 10 students. Since the introduction of robots, demand for the course has necessitated annual offerings, with enrollments typically over 15 students. The only time this level was not achieved was in a semester when there were five electives scheduled (usually there are only four electives scheduled) and the course was scheduled as an evening course—a time period avoided by most students. Under those circumstances, the total enrollment was 8. Thus, one can argue that although a robotics element in AI courses is usually a strong interest generator, there are some factors that it will not overcome! Four course graduates have gone on to pursue graduate work in artificial intelligence areas (two Ph.D. candidates and two Master's degree candidates, one of whom is applying this year for admission to doctoral work in AI) since the robotic element was added. While it can be debated whether those students would have gone on to graduate AI work even without exposure to robotics projects, it is interesting to note that prior to the robot projects' addition no course graduates had pursued graduate AI work.

The robotics component has definitely added to department camaraderie. The Capture the Balls Contest has been widely publicized in the department, and each year some course graduates attend the following year's competition to regale current participants with triumphs and tragedies from the past.

Although the RCX transceiver often picks up IR signals reflected from walls, students have

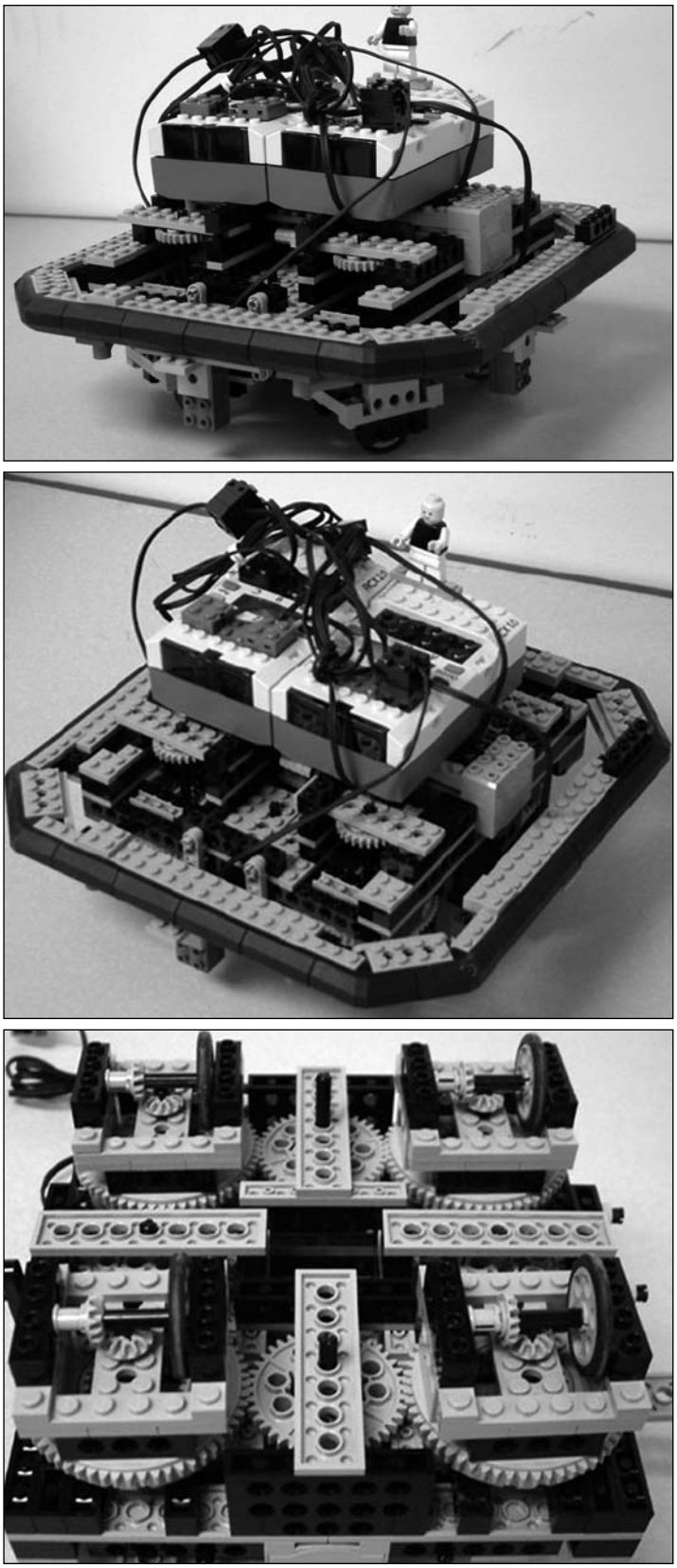

Figure 10. Synchro-Drive Chassis Design That Allows RCXs Mounted on Top to Face the Same Direction at All Times While the Lower Wheels Turn on Their Own Axes.

sometimes been frustrated in maintaining wireless control of a robot when it turns its transceiver away from its control desktop PC's IR tower. To mitigate this problem (and the problem of having nonengineering students spend too much time on complex chassis design) the course supplies teams with a prebuilt synchro drive chassis based on the Lego Mindstorms Syncro Drive[7] (see figure 10). This chassis design can keep an RCX mounted on it pointed in the same direction since the base does not turn as the wheel casings below it turn. Thus, the chassis can keep its RCX units aimed at an IR communication tower as the robot maneuvers on the floor.

Regarding RCXLisp itself, several cognitive science concentrators and engineers have commented on how they felt they could get past coding details faster in RCXLisp and Common Lisp than in Java because of those languages' lack of typing. The prominent use of RCXLisp in our department's AI course has had the effect of raising students' interest in understanding how to program in Lisp in our department's programming languages course.

Students have also commented on the immediacy of working with RCXs using the Common Lisp Listener: simply by invoking a function in the Listener, an RCX can be made to respond. This is definitely an example of satisfaction with being able to get past the "hello robot" stage of programming quickly.

The library's integration with low-level system functions such as infrared USB communication helps students get past the uninteresting details of port communication and instead concentrate on the AI-oriented problems of environmental noise, sensor sensitivity, and environmental nondeterminism. For most students, the library and the robot projects gave them their first experience with coordinating responses to real-time data sources (sensors) whose values could change "even when the program isn't watching," as one student put it.

The library's support for "on-the-fly" program generation and download helps students appreciate the power of Common Lisp's lack of differentiation between code and data. Since Common Lisp function declarations are themselves merely formatted linked lists, students can generate plans as linked lists within a planner and then download the same data to the RCX as an immediately executable form.

Students have reported that debugging local RCXLisp programs involves more effort than debugging remote RCXLisp programs within Common Lisp environments. They usually resort to adding commands to a remote RCXLisp program to play musical tones or to light up LEDs in order to debug program control flow problems.

Another recurring challenge students have noted is that the library does not provide built-in support for synchronized message passing either between threads on the RCX or between RCX threads and desktop threads. At the start of each course offering, computer science majors are encouraged to develop their own macros and functions for their team for these purposes so as to reinforce the process synchronization skills they typically learn in the second semester of their sophomore year. There is no Common Lisp standard for process coordination. Nevertheless, in an effort to make RCXLisp more useful in a wider array of computing curricula, the next version of RCXLisp will include simple ad hoc synchronous message-passing macros for local RCXLisp threads. However, construction of a general mechanism for synchronizing desktop access to RCX registers on top of RCXLisp's *var* function will require greater care. Any high-level synchronization constructs added to remote RCXLisp should be expected to interface well with the threading constructs in whichever Common Lisp environments they are used.

Conclusions and Future Work

RCXLisp is the first open-source Common Lisp library for Mindstorms that supports both remote control and onboard programming of robots as well as targeted communication between multiple robots and command-center desktops.

The growing role of intelligent agent design as an organizing principle for the study of intelligent artifacts, the increasing availability of low-cost robotics platforms, and the growing number of accessible robot software libraries for undergraduates are important trends in AI education. Each trend has had a significant positive impact on student experience in Villanova's CSC 4500 AI course because of the adoption of RCXLisp.

The projects described in this article demonstrate that students in the AI course have been able to use RCXLisp projects to explore the design of example agents from every one of the classes described in Russell and Norvig's intelligent agent taxonomy (Russell and Norvig 2003), with the exception of the utility-based agent class. This exception is due to the late coverage of the topic in the course. Our department has recently added a course on machine learning. With the movement of that material from the AI course to the new course, we expect to be able to start coverage of utility-based agents early enough in the AI course to permit work on utility-based robotic agents.

Combined with the low cost and adaptability of Mindstorms, RCXLisp made it possible to add robotics-inspired projects to our AI course cost-effectively, without having to distract students with many mechanical or computer engineering issues. Student feedback, enrollment levels, and graduates' continued interest in AI over the last four years of using RCXLisp and Mindstorms in our course lend support to this claim. The RCXLisp library's integration with Common Lisp enabled students to adapt Lisp AI software from the Internet and from Russell and Norvig's sample code corpus to robot applications with little difficulty.

RCXLisp was first developed to address needs of an AI course, but has since been subsumed into a larger initiative—Thinking Outside the (Desktop) Box—to support the use of Mindstorms in combination with other programming languages across the full computer science curriculum (Klassner and Anderson 2003).[8]

Students' imaginations are not the only place where RCXLisp and Mind-

storms have had an October sky effect. RCXLisp's design team continues to make improvements to the package. In addition to the future work described in the previous section, an important goal is to add call stack support and memory management support to the Mnet firmware. This would extend the Lisp functionality of the on-board language for RCXs. A related possibility would be to eliminate firmware and target the H8 processor directly, as the C++ BrickOS library for Mindstorms does.

Another future project is to integrate the Lego vision command camera into the RCXLisp library. This would provide a low-cost tool for exploring machine vision problems. It would also provide an additional data channel for student teams to use in the capture the balls contest in coordinating robots' coverage of playing field areas.

The RCXLisp library already has generic serial API functions, making it a useful basis for Common Lisp solutions for interfacing with serial devices. However the library's USB interface functions currently only work specifically with Mindstorms USB IR towers. Work is underway to extend Common Lisp streams to support generic USB devices so that not only the vision command camera but also other USB-based sensors and devices can be used in RCXLisp (Jacobs, Jorgage, and Klassner 2005).

In January, 2006, Lego announced a new version of Mindstorms, NXT. Based on released information about the new platform, I expect that RCXLisp will be portable to NXT.

Acknowledgments

I am grateful to Andrew Chang for his graduate independent study work that developed the Mnet firmware, my graduate students Drew Jacobs and Eric Clark for their work in the design of reliable USB communication libraries for RCXLisp, and my undergraduate research assistant Christopher Continanza for his work in the design of robust Lego chassis for the AI course projects. I also thank the reviewers and editors for comments that improved this article.

Lego Mindstorms and RCX are trademarks of the Lego Group, which does not sponsor, authorize, endorse, or support any of the third-party work cited in this article. The author has no financial relationship with the Lego Group except for a discount purchase plan for Mindstorms equipment for training seminars run under NSF Grant No. 0306096.

This material is based upon work supported by the National Science Foundation under Grant No. 0088884 and Grant No. 0306096. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Notes

1. Such as ActivMedia Robotics (www.activrobots.com) and Lego Mindstorms, www.legoMindstorms.com/. See also F. Martin, The MIT HandyBoard Project, December 2005, http://lcs.www.media.mit.edu/groups/el/Projects/handy-board.

2. See www.noga.de/legOS; lejos.sourceforge.com; Not Quite C (NQC), bricxcc.sourceforge.net/nqc; and the LEGO/ Scheme compiler, Oct. 2005 (www.indiana.edu/~legobots/legoscheme/), by A. Wick, K. Klipsch, and M. Wagner.

3. See lejos.sourceforge.com.

4. See www.indiana.edu/~legobots/legoscheme/.

5. www.hitechnic.com.

6. www.techno-stuff.com.

7. See www.visi.com/~dc/synchro.

8. Installation files for RCXLisp and handouts for the assignments described in this article are available for download from the web site for this project at robotics.csc.villanova.edu.

References

Beer, R. D., Chiel, H. J., and Drushel, R. F. 1999. Using Autonomous Robotics to Teach Science and Engineering. *Communications of the ACM* 42(6)(June): 85–92.

Brooks, R. A. L. 1993. A Subset of Common Lisp. Technical Report, Massachusetts Institute of Technology Artificial Intelligence Laboratory, Cambridge, MA.

Carver, N., and Lesser, V. 1993. A Planner for the Control of Problem Solving Systems. *IEEE Transactions on Systems, Man, and Cybernetics,* Special Issue on Planning, Scheduling, and Control 23(6): 1519–1536.

Cohen, P. 1995. *Empirical Methods for Artificial Intelligence.* Cambridge, MA: The MIT Press.

Jacobs, D.; Jorgage, B.; and Klassner, F. 2005. A Common Lisp USB Communication Library. Paper presented at the 2005 International Lisp Conference, Stanford, CA, June 19–22.

Klassner, F., and Anderson, S. 2003. Lego Mindstorms: Not Just for K–12 Anymore. *IEEE Transactions on Robotics and Automation Magazine* 19(3): 12–18.

Klassner, F. 2002. A Case Study of Lego Mindstorms'™ Suitability for Artificial Intelligence and Robotics Courses at the College Level. In *Proceedings of the Thirty-Third Special Interest Group on Computer Science Education Technical Symposium on Computer Science Education*, 8–12. New York: Association for Computing Machinery.

Kumar, D., and Meeden, L. 1998. A Robot Laboratory for Teaching Artificial Intelligence. In *Proceedings of the Twenty-Ninth Special Interest Group on Computer Science Education Technical Symposium on Computer Science Education*. New York: Association for Computing Machinery.

Mazur, N., and Kuhrt, M. 1997. Teaching Programming Concepts Using a Robot Simulation. *Journal of Computing in Small Colleges* 12(5): 4–11.

Russell, S., and Norvig, P. 2003. A*rtificial Intelligence: A Modern Approach,* 2nd ed. Englewood Cliffs, NJ: Prentice Hall.

Steele, G. 1990. *Common Lisp: The Language,* 2nd ed. Woburn, MA: Digital Press.

Stein, L. 1996. Interactive Programming: Revolutionizing Introductory Computer Science. *ACM Computing Surveys* 28(4es)(December): 103.

Yuasa, T. 2003. XS: Lisp on Lego Mindstorms. Paper presented at the 2003 International Lisp Conference, New York, October 12–15.

Frank Klassner earned B.S. degrees in computer science and in electronics engineering from the University of Scranton. He earned his M.S. and Ph.D. in computer science from the University of Massachusetts at Amherst. He is an associate professor in Villanova University's Department of Computing Sciences. In addition to AI and robotics, his interests include adaptive signal processing and Lisp modernization for the twenty-first century. He can be reached at Frank.Klassner@villanova.edu.

We invite you to participate in the Fifteenth Annual AAAI Mobile Robot Competition and Exhibition

Sponsored by the American Association for Artificial Intelligence, the Competition brings together teams from universities, colleges, and research laboratories to compete and to demonstrate cutting edge, state of the art research in robotics and artificial intelligence.

The 2006 AAAI Mobile Robot Competition and Exhibition will be held in Boston, Massachusetts, as part of AAAI-06, from July 16–20, 2006.

The program will include the Robot Challenge, the Open Interaction Task, the Scavenger Hunt, the Robot Exhibition, and the Mobile Robot Workshop.

Registration and information is available at palantir.swarthmore.edu/aaai06/.

AAAI-06 conference details: www.aaai.org/Conferences/ AAAI/aaai06.php