A Case Study of LEGO Mindstorms'™ Suitability for Artificial Intelligence and Robotics Courses at the College Level

Frank Klassner
Department of Computing Sciences
Villanova University
Villanova, PA 19085
Frank.Klassner@villanova.edu

Abstract

This paper examines LEGO Mindstorms'™ suitability as a hardware platform for integrating robotics into an Artificial Intelligence course organized around the agent paradigm popularized by Russell and Norvig. This evaluation discusses how kits and projects based on Mindstorms supported students' exploration of the issues behind the design of agents from three classes in Russell and Norvig's intelligent agent taxonomy. The paper's investigation also examines several popularly-perceived limitations of the Mindstorms package for college-level robotics projects and shows that most of these "limitations" are **not** serious impediments to Mindstorms' use, while certain other of these "limitations" do indeed present challenges to the platform's use.

1 Introduction

Recent publications show that robotics-oriented problems can be used successfully as a pedagogical tool for grounding abstract topics in certain computer science courses (e.g. introductory programming [7,10]) In particular, Kumar and Meeden showed [5] that it was feasible to use a combination of the HandyBoard [6] and LEGO building blocks as the basis for an effective robotic laboratory for artificial intelligence (AI) course topics at undergraduate liberal arts institutions. Since the late 1990's, several undergraduate computer science and computer engineering programs [3] have started robotics labs built around the HandyBoard/LEGO combination or the ActivMedia platform [1].

In 1996 LEGO Inc released the Mindstorms robot development and programming kit for middleschool students. Though related to the HandyBoard's design, the Mindstorms platform has not been explored as a viable platform for college-level projects until quite recently [4].

SIGCSE'02, February 27- March 3, 2002, Covington, Kentucky, USA.

Review of recent SIGCSE and ITiCSE literature also shows a lack of study of Mindstorm's suitability for supporting AI pedagogy. Popular justifications for this situation have included the kit's small number of sensors and effectors, a lack of sophisticated programming environment, and a lack of support for common college-level programming languages such as Java and C.

These observations did indeed represent serious obstacles to Mindstorms' use in AI courses at the time of the platform's release. However, during the last four years an active Mindstorms hobbyist community has arisen on the Internet and has produced a large, informal body of work (e.g. developed programming environments and experimented with sensor-management techniques at the platform's bytecode programming level) to ameliorate some of these and other perceived shortcomings.

In addition to this development, two practical reasons recommend that the Mindstorms platform be re-examined for viability in college-level artificial intelligence projects:

- A single Mindstorms kit costs approximately $200 and thus is one-quarter the cost of a HandyBoard-based robot kit and one-tenth the cost of an ActivMedia-based robot kit.
- The Mindstorms kit is a suite of reusable snap-together sensors (touch, rotation, light, temperature), effectors (motors, lights, infrared emitters) building blocks, and a programmable control unit that can serve as the basis for a wide variety of robotics projects. Platforms of equivalent price to Mindstorms often require soldering and other one-shot construction methods that make their reuse across course offerings economically impractical.

For these reasons, this paper reports on a two-track investigation into the Mindstorms platform's suitability for integrating robotics into two semester offerings of the Villanova University AI course organized around the agent paradigm popularized by Russell and Norvig [11]. The first track examines the sophistication of the robotics projects that the course kits supported. It also examines several popularly-perceived limitations of the Mindstorms package for college-level robotics projects and shows that many of

these "limitations" are not serious impediments to Mindstorms' use, while certain other of these "limitations" do indeed still present challenges to the platform's use. The second investigation track examines the pedagogic outcomes observed from student evaluations.

As background for interpreting the projects and Mindstorms' evaluations, this paper first describes the students who participated in the CSC 4500 offerings, the course's relationship to other courses in the Villanova computer science program, and the LEGO equipment used by the students. The paper then describes the LEGO projects and student performance. The last two sections of the paper discuss the observed strengths and limitations of the Mindstorms platform and course projects and propose guidelines for using the platform in other environments

2 Student Backgrounds and Course Context

The elective AI course at Villanova (CSC 4500) has no formal programming prerequisites. Computer science majors typically take CSC 4500 in their fourth year, by which time most computer science majors have taken a Programming Languages course that briefly introduces them to Lisp, the language most often used in CSC 4500.

The course is also open to cognitive science minors and computer engineers, who generally have no programming experience in Lisp and at most one semester of introductory programming in Java.

The course offerings examined in this paper were held in Fall 1999 and Spring 2001.

3 Mindstorms Equipment

From the outset, robotics projects in CSC 4500 were intended to be team-based active-learning projects. Each team's kit for constructing LEGO robots contained the following hardware:

(a) 3 Mindstorms Robotic Invention Systems packages

(b) 3 more light sensors beyond the three in (a)

(c) 3 more touch sensors beyond the six in (a)

(d) 3 more motors beyond the six in (a)

(e) 2 rotation sensors

(f) 24 rechargeable batteries

(g) 1 large lockable toolbox to hold all of the above as well as a partially-completed robot.

A search was conducted for a Common Lisp programming environment for Mindstorms, but none could be found for the Fall 1999 offering. Each team in that offering therefore used the NQC (Not Quite C) programming environment developed by David Baum [2] to program their robots. The language has C syntax, but lacks the following C features: pointers; recursive procedural invocation, and procedures with return values or parameters. NQC compensates for the lack of procedure support with a macro definition facility for "inline procedures."

NQC was selected for use in CSC 4500 in spite of its missing C features because it did provide a high-level Mindstorms API that students could quickly learn, allowing them to develop control programs that were much more sophisticated than those achievable through the LEGO-supplied GUI programming environment. NQC also provided students with a more powerful API for using the Mindstorms hardware's multithreading capabilities.

The Spring 2001 offering also used NQC as the primary programming tool, although a primitive Lisp interpreter for remotely controlling Mindstorms robots was used in one project (see project VI in Section 4).

4 Course Projects

In both course offerings students worked on a number of team projects, some based on Mindstorms and NQC coding, others based on Lisp coding. The following Mindstorms-based projects were developed for the course:

I. Simple-Reflex Robot Design. This 1-week project's goal was to show students how robots with simple stimulus-response rules and a very limited model of the environment (the second-simplest design in Russell and Norvig's agent design taxonomy) could achieve effective behaviors.

This project asked students to design a robot based on a tread-wheeled "Pathfinder" model described in LEGO's user manual. Students were required to start with this basic design in order to reduce time spent on distracting mechanical engineering issues, but they were encouraged to mount sensors as needed.

Students first built a robot that ran a random walk, ignoring environmental stimuli. The walk consisted of traveling forward for a random number of seconds, then randomly turning 30 degrees left or right, and repeating these two actions forever. We then ran these first-stage robots through a short obstacle course with narrow passages, and timed their progress.

Students next added one or more threaded tasks to monitor either (a) whether, via feedback from the Mindstorms' built-in infrared sensor or mounted light sensors, a robot is too close to a wall, or (b) whether, via feedback from mounted light or touch sensors a robot was wedged in a corner. The monitor task was to override the default random walk by turning 30 degrees away from the observed stimulus (either left or right, as necessary). We then allowed these second-stage robots to run in the same obstacle course, and students observed a significant (though not as large as they generally expected!) improvement.

II. Sensor Accuracy and Functional Simulation. This project's goal was to show students how (in)sensitive each of their sensors were to various stimuli, and to discover how some sensors can simulate other sensors' capabilities.

The first part of the project required each team of students to expose their sensors (touch, light, infrared, and rotation) to as wide a variety of stimuli as they could generate to learn the range of meaningful response values the sensors could generate. They were also asked to explore what responses were generated when two or more sensors were connected to the same input port. The experiences students gained in this project were invaluable in helping them decide how to choose and arrange sensors on robots for the more complicated projects described later.

The second part of the project asked teams to design simple robots that used only touch, infrared, and/or light sensors to duplicate the accuracy and sensitivity of rotation sensors. This duplication effort was intended to familiarize students with the concept of functional emulation – a key concept in understanding how abstract AI tools like planners and search algorithms can (partially) emulate each other.

III. Robot Odometry. This 2-week project's goal was to help students understand the major factors that can introduce error into a robot's internal representation of where it believes it currently located in the world – an important issue in any navigation process.

Each team was required to design and build a robot that would measure the perimeter of a black shape on a light background on the floor. The reported measurement (over 160 cm) had to be accurate to within +/- 5 cm, and had to be obtained within 6 minutes from the time the robot was started. The project allowed use of dead-reckoning and landmark-based navigation techniques. Although all teams succeeded in this project, all were surprised at how short the 6-minute time limit soon appeared in light of the accuracy constraint.

IV. Capture-the-Balls Contest. This 1-month project's goal was to help students tie the skills they developed in projects I-III with search-state-based and hill-climbing-based agent designs (the third level of Russell and Norvig's taxonomy).

Each team was required to design and build a robot no larger than 1 cubic foot. The robot's task was to play in a 45-minute contest (based on [3], but adapted to fit Mindstorms' capabilities) against other teams' robots. Contestants had to play in a 20x9 sq. ft. walled playing area in which each team had a 1 sq. ft. nest area, colored dark purple. All other portions of the playing field were light-yellow colored. Scattered throughout the field were black, white, and yellow ping pong balls. For each ball that was in a team's nest at the end of the contest, the following points were awarded: white +1, yellow +5, black –1. The playing field was marked with a black 1x1-foot grid whose lines were 1 cm wide.

Teams were encouraged to try a wide variety of game strategies, some of which required landmark-based navigation via the grid, others of which required state-space hill-climbing, and still others of which relied on probabilistic observations about the environment. Teams were also encouraged to make their use of strategies time-dependent: as the contest progressed, robots could switch strategies based on their current state (e.g. estimated score, position on field). Since robots were permitted to "attack" other nests and scatter or steal balls, there was a very wide variety of approaches that a team could explore, minimizing the risk of teams unintentionally duplicating their efforts.

V. Team-based Capture-the-Balls. This 1-month project was the same as project IV, except that each student team was allowed to field up to three robots as a team. This project's pedagogic goal was to help students understand the issues behind the design of cooperative, communicating agents.

VI. Robotic 8-Puzzle Solver. This 2-week project had the goal of showing students that knowledge representations (data abstractions) that speed up search-based problem solvers can produce solution representations that are not easily translated into control programs for hardware.

The project had two stages. In the first stage students had to develop a knowledge representation and Lisp search program to solve the 8-Puzzle. The team developed a set of four operators that involved 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 observed that this design decision dramatically reduced the branch factor of the search tree (4 vs. 32), leading to a faster execution time for the game-solver.

The second stage required students to write a second Lisp program that invoked functions in an ad hoc library developed by the instructor to send remote-control messages to a Mindstorms robotic arm mechanism (designed by the instructor – construction details available upon request) to move pieces in an 8-Puzzle according to the solution developed by stage 1's programming. It was in this stage that students discovered that the search space reformulation trick ultimately cost them in the complexity of the translation their second program had to perform on the "move space" operator list to "move tile at (2,2) to (2,1)" types of commands.

5 Course Organization

The Fall 1999 offering had 17 students: 15 computer science majors, 1 physics major, and 1 computer engineering major. Projects I-V described in Section 4 were assigned to 6 teams (three teams of 3 and two teams of 4).

The Spring 2001 offering had only 4 students: 3 computer science majors, and 1 cognitive science minor (psychology major). The reduced class size led the instructor to decide to make all four students work on a single team whose Project Manager position rotated among the members for each project. In this offering the team worked on Project VI and III (in that order). The small size in the second offering is attributable to two factors: our department offered five elective courses that semester instead of the usual four

(leading to a dilution of students across a larger number of courses) and the Spring 2001 CSC 4500 offering was scheduled as an evening course (which historically have depressed enrollments at Villanova). To address potential reader concerns that the course's workload or design unduly discouraged students from signing up, it should be noted that as of November 2001 there are 24 students (computer science, astronomy, mechanical engineering, biology, and computer engineering majors) enrolled for the Spring 2002 offering of the course (one of 4 CSC electives available, in a late-morning time-slot).

6 Student and Project Evaluation

As seen in students' performance on examination problems, the robot projects did give them a strong understanding of the concepts highlighted in the project goals in Section 4. In timed exams students were asked to describe how (hardware or software) agents in all of the Russel and Norvig taxonomy (including those not explicitly explored in robot projects) might be realistically implemented for various environments. This type of abstract question was generally poorly answered in CSC 4500 offerings prior to Fall 1999. In the two robot-oriented offerings students scored 20% higher.

Based on other exam questions, students in the robotic-based CSC 4500 offerings were able to demonstrate their understanding of the concept of "emergent behavior" better with very concrete robotic examples than those in earlier offerings. To be objective, however, it must be noted that robotics students were more susceptible to making the mistake of believing that "emergent behavior" only refers to behavior that arises from errors in the interaction between pieces of a complex system rather than *any* behavior (positive or negative) that arises from interactions between pieces of a complex system.

Although it is not a stated goal of the AI course, the multitasking programs that students wrote in the robotics-based offerings gave them much more confidence in working with multithreading issues, according to course exit surveys (10 of 21).

All student survey respondents, while generally viewing the robotics problems as positive influences on their learning, felt that the AI course should be expanded to a 4-credit (4-hour/week) course to better reflect the workload.

A large number (16 of 21) of student survey respondents also indicated that their impression of the difficulty of sensing the environment and correctly interpreting sensors' environmental measurements was mixed: half felt the problem was more difficult because the LEGO sensors were inaccurate, while the other half felt that the problems they had encountered were just indicative of the nature of the general sensor interpretation problem.

Along the same lines of "hardware evaluation," 12 of the 21 survey respondents felt that more planning should be done by the instructor to reduce the time computer science students spend on design of robot chassis. When asked to rate their agreement with the statement "The design of hardware for an intelligent agent is every bit (if not more) important as the design of software for an intelligent agent," at the beginning of the robotics courses, the respondents' average was 5.7 (1 being strongly disagree, 10 being strongly agree). After the course this same question received an average agreement level of 7.1.

Based on these observations from the (admittedly small) set of students in the two course offerings, I argue that the robotics projects were a more positive than negative influence on students' learning of AI concepts, and a more positive than negative influence on students' appreciation of the issues behind design of "hardware+software" agents.

7 Mindstorms Evaluation

While one may be able to conclude that this report's project evaluations lend some more support to the claim that robotics-oriented problems can be used successfully as a pedagogical tool for grounding abstract topics in computer science courses (at least, in artificial intelligence), several questions must be addressed before one can claim that LEGO Mindstorms is a good platform for supporting projects of enough sophistication to produce these benefits.

A commonly-cited criticism of Mindstorms for use in college courses is that LEGO's GUI-based programming support for the platform is too grade-school oriented. While true, I assert that the existence of programming environments such as NQC and RCX-Ada [4] counter this problem reasonably well. In the summer since the Spring 2001 course offering, open-source programming environments for the Java [9], C++ [8], and Lisp [12] languages have been developed for the Mindstorms platform. These languages should make Mindstorms more easily integratable into any programming-oriented computer science course as an alternative platform for exploring programming problems.

Another common criticism of the Mindstorms kit is that it does not allow one to build sophisticated robots. Indeed, it might seem that the fact that this paper's own robotics toolbox (Section 3) needed the contents of three individual Mindstorms kit is evidence in favor of this claim. I hasten to clarify this problem by observing that the only project that required the material in more than one kit was Project V – Team-based Capture-the-Balls. It was, in fact, anticipation of this project that led to the expansion of the basic team toolbox.

I do accept the criticism that the Mindstorms kit does not contain an adequate number of sensors to solve some interesting problems. In particular, the rotation sensor (aka "angle sensor") is crucial to any robust solutions to the robotic navigation problem, yet it is not included in the basic Mindstorms kit. I highly recommend that anyone contemplating use of Mindstorms for college robotics projects add this sensor to their development budget!

While it is also true that the Mindstorms' platform only supports three input ports, this does not limit the platform's sensory capability as much as it would seem. In my experience and my students' experience, it is possible to "stack" more than one touch or light sensor on the same port and to interpret the port's measurements appropriately. That is, by configuring (through an NQC program) a port to be in "raw mode," one will obtain an integer value between 0 and 1024 that can be analyzed through repeated experimentation to determine when one, two, or no sensors on the same port have been activated. In some situations that I will discuss in my lecture, one can even determine which particular sensor in a stack has been activated.

When one is restricted to using the firmware supplied by LEGO in a Mindstorms robot, I believe from my experiences that it is a valid criticism that control programs are somewhat limited in size and capability. Primarily, this is due to the firmware's lack of support for a procedure call stack and dynamic memory allocation. However, now that tools such as LeJOS (a native Java Virtual Machine for Mindstorms) [9] and LegOS (a C++ kernel for Mindstorms) [8] exist, this criticism has less vigor.

It is still a valid criticism as well to claim that the Mindstorms' 32KB onboard memory represents a serious restriction for artificial intelligence instructors (like myself) who want to give students experience in working with real-time schedulers and planners situated within robots. One possible solution to this problem would involve designing a framework that offloads most of the computation in these systems to a desktop, and then periodically issues remote-control commands to the Mindstorms robot.

A major criticism I do agree with is that the platform (and, for now, even open-source enhancements) does not support useful wireless protocols over its built-in infrared port. All Mindstorms programming environments that depend on the standard LEGO firmware are limited to using a broadcast protocol that does not support targeted message-passing among robot teams or between a central control desktop and a remotely-controlled set of robots. This was a serious problem for students working on Project V.

A personal criticism my students and I have in regards to using the Mindstorms platform in my Lisp-based AI course was that because there was no Common Lisp programming environment for the platform, my students had the added overhead of learning NQC in addition to Lisp. However, with new developments in Wick et al.'s Lego/Scheme compiler [12] and my own future work in a Lisp Mindstorms remote-control library, I expect this criticism to disappear soon.

8 Conclusions and Future Work

Based on the sophistication of the projects in Section 4 and the observations in Section 6 and Section 7, I claim that the LEGO Mindstorms platform has grown since its introduction to become a cost-effective platform worthy of consideration by any small college as a complementary programming environment to tradtional PC programming.

As more work is done in the open source community to ameliorate the unaddressed criticisms in Section 7, the Mindstorms platform's pedagogical viability will only improve. To this end, I plan to work on extending the basic LEGO firmware to include support for targeted message-passing and to develop a system for distributing computation within a robotic network.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 0088884. 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 NSF.

References

[1] ActivMedia Robotics, http://www.activrobots.com

[2] Baum, D. Not Quite C (NQC), June 2001, http://www.enteract.com/~dbaum/nqc/index.html

[3] Beer, R. D., Chiel, H. J., and Drushel, R. F. "Using autonomous robotics to teach science and engineering," *Communications of the ACM*, Vol. 42 (6) June 1999, pp. 85 - 92.

[4] Fagin, B., Merkle, L., and Eggers, T. "Teaching basic computer science concepts with robotics." In *Proceedings of the 32nd SIGCSE Technocal Symposium on Computer Science Education (2001).*

[5] Kumar. D., and Meeden, L. A robot laboratory for teaching artificial intelligence. In *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education (1998).*

[6] Martin, F., The MIT HandyBoard Project, 2000. http://lcs.www.media.mit.edu/groups/el/Projects/handy-board/, Sept. 6, 2001.

[7] Mazur, N. and Kuhrt, M. "Teaching programming concepts using a robot simulation," Journal of Computing in Small Colleges, Vol. 12, (5) 1997, pp. 4-11.

[8] Noga, Markus. LegOS, June 2001, http://www.noga.de/legOS

[9] Solorzano, Jose. LejOS, May 2001, http://lejos.sourceforge.com

[10] Stein, L. A., Interactive programming: revolutionizing introductory computer science," ACM Computing Surveys 28A(4), December 1996.

[11] Russell, S. and Norvig, P. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[12] Wick, A., Klipsch, K., and Wagner, M. LEGO/Scheme compiler, v. 0.52, (June 2001) http://www.cs.indiana.edu/~mtwagner/legoscheme/