Mobile Robot Programming in Education

Jean-François Lalonde, Christopher P. Bartley, and Illah Nourbakhsh
The Robotics Institute
Carnegie Mellon University
Pittsburgh, USA
{jlalonde, cbartley, illah}@ri.cmu.edu

***Abstract*— The *Mobile Robot Programming Laboratory* course has been taught at Carnegie Mellon University for the past twelve years. It is a problem-driven class designed for students with little or no experience with robots. In this paper, we first present the current status of the class, and show how it improves the education and training of students in a robotics curriculum by giving them a hands-on experience with a real robot. We show that, in addition to core subjects such as perception, action and cognition, students also have the opportunity to learn advanced topics such as reinforcement learning and multi-robot coordination. We then discuss the evolution of the class under general categories: hardware and programming environment, team experiments, and assignments. We present important lessons learned in each category, and how they affect the learning experience of participating students. We conclude by discussing future opportunities.**

I. Introduction

Robots have played an important role in education for many years. Whether being used as tools to support teaching or as the primary objective, their presence is stimulating for students. As shown in [1], the current uses of robotics in education are now very broad and diverse.

In this paper, we describe the *Mobile Robot Programming Laboratory* course that has been taught for the last twelve years at Carnegie Mellon and earlier at Stanford University [2]. It is offered both to undergraduate and graduate students. In this class, teams of students with little or no prior experience with robots must program a mobile robot to overcome a variety of challenges of increasing difficulty. As we shall see, mobile robots represent an excellent opportunity for students to get hands-on experience with real problems and are a unique tool for learning.

The aim of the paper is two-fold. First, we present the current status of the course and show that it contributes significantly to the students' education not only in robotics, but also under more general educational themes. Second, we take advantage of its long history and present the evolution of the class and lessons learned since its creation. By doing so, we hope to improve the learning experience of students in future versions of the class, and also help groups from other universities that have a similar curriculum by sharing this long-term experience. We also support our claims with educational data, taken from a survey filled by $n = 37$ students from the last two editions of the class at Carnegie Mellon, as well as with collected students' typical comments.

Similar courses are now being offered at numerous universities. However, the *Mobile Robot Programming Laboratory* course differs from many in two key areas. First, it emphasizes robot *programming* whereas many other classes also cover robot morphology and construction. For example, the courses discussed in [3], [4], [5] use commercially-available parts (LEGO bricks), but require the students to assemble the robot. Similarly, some other courses, such as those discussed in [6], [7], use custom kits which the students must first build. In contrast, the *Mobile Robot Programming Laboratory* course focuses solely on robot programming and therefore uses a preassembled, commercially-available platform. Thus the students can begin programming the robot immediately and hardware-related concerns are kept to a minimum. The second major difference concerns hands-on experience. Courses such as the one described in [8] employ research robots which are very often in high demand and whose availability is very limited. Courses such as [9], [10] use simulated or remotely-controlled robots. However, the *Mobile Robot Programming Laboratory* course strives to provide a large amount of hands-on experience under real-world constraints by providing each team with a robot to which they have virtually unlimited access.

The rest of the paper is organized as follows. In Section II we describe the course logistics. The following section presents its current status, and provides more details about the technical concepts learned. Finally, Section IV presents the evolution of the class.

II. Course outline

We first present the current status of the course by describing the logistics, presenting the mobile robotic platform used, and by giving a brief outline of the different topics covered.

A. Logistics

Mobile Robot Programming Laboratory is a one-semester course that uses a problem-driven syllabus to give students hands-on experience with mobile robot programming. The class is kept to a manageable target size of 30 students, with the support of 2 teaching assistants. It is designed for students with little or no experience with robots, and it is available to senior undergraduates in Computer Science or new graduates in Robotics.

Students first form teams of three that are kept intact throughout the semester and immediately begin addressing weekly assignments that usually build upon the previous one. Each week, teams are evaluated based on their robot's ability to meet the assignment requirements, and all team members are given the same grade. The last four weeks are dedicated to preparing for a final mobile robot competition between the teams (see Section IV-C).

Also part of the syllabus is a weekly lecture that aims to enrich students' theoretical knowledge of important concepts in robotics. Part of the lecture time is reserved for group discussion, where students are invited to discuss the previous week's challenge and present the solutions they used.

B. Robotic platform

The current robotic platform in use is the Nomad Scout differential-drive robot mounted by a Dell laptop computer running the Java 1.4.2 runtime environment under Windows XP (see Figure 1). The robots are also equipped with a low-cost USB webcam, for basic vision. Communication between robot and computer is achieved via a serial interface, and Java code making native calls via the Java Native Interface (JNI) to low-level C code, necessary for implementing basic motor control and sensor reading is provided to the students. The laptops are also equipped with IEEE 802.11b wireless adapters, necessary for the last part of the class (see next section).

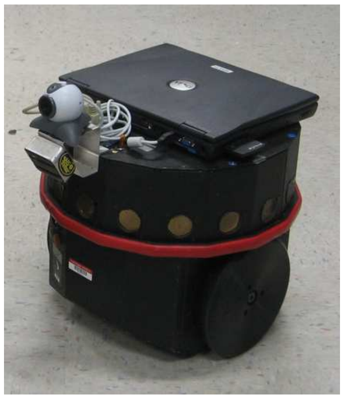


Fig. 1. Hardware environment in use for the 2004 edition of *Mobile Robot Programming Laboratory*: Nomad Scout with Dell laptop equipped with a PCMCIA wireless card and a Logitech USB webcam. The camera is fixed on the magnet support and facing forward. The circles on the side of the robot are its sonars.

C. Curriculum

We present an updated curriculum overview in Figure 3 from the originally introduced version in [2]. In summary, new concepts are introduced each week during lectures. Topics include feedback control, reactive control, sensors (sonars and camera), localization, planning, robot architecture, and multi-robot collaboration. In addition, we also provide a detailled review of the main fundamental concepts covered, and we introduce advanced concepts explored by students, in Section III.

As shown in Figure 3, the goal of the assignments is to build a system capable of taking part in the final competition by playing a *game*. The students create a team of two mobile robots to compete with other robot teams in a maze. The robots have to pick up gold pieces scattered throughout the maze, and whichever team returns them the fastest wins the game. The annual competition has now become a popular event in the university community.

Fig. 2. Students in preparation for the 2004 *Mobile Robot Programming Laboratory* contest.

III. TECHNICAL CONCEPTS

We now review robotics concepts covered in the course, and show that the "learning-by-doing" experience acquired by the students is an essential part of a robotics-oriented curriculum. Furthermore, the hands-on experience with a real robot helps them develop problem-solving aptitudes and understand real-world constraints.

A. Fundamental concepts

We divide the fundamental robotics concepts into three broad categories: perception, action, and cognition.

1) Perception: As mentioned previously, the robots are equipped with two different kinds of sensors: 16 sonars and one camera. The sonars are used for localization in the maze world. Students have to deal with the inherent uncertainty in sensing, learn about the physics governing the sensors used, and implement filtering methods to get high-accuracy localization despite noisy raw data.

The camera is used only to detect the presence of gold in front of the robot (blue cardboard attached to a metal base). Color detection algorithms range from simple RGB thresholding to more robust HSV-space automatic segmentation. These more complex methods proved more successful at dealing with the different lighting conditions of the final contest location.

2) Action: The main action-related challenge is to build a robot that can move in a maze very accurately. Therefore, after exploring low-level controls with PI and PID controllers, the teams are asked to build an abstract action, *go-to-next-node*, that allows the robot to move from one maze node to another without accumulating rotational or translational error. This seemingly simple request turns out to be very challenging because of wheel encoder error accumulation and noisy sonars.

For example, this year's successful teams developed line fitting methods based on filtered sonar readings. When detected, the walls on either side of the robot were used to infer a center line (see Figure 4). The robot then followed this center line using a simple PI controller. The total distance travelled by

1 Introduction
  1.1 Initiation to robot programming
2 Feedback and Reactive Control
  2.1 Precise rotational and translational movement
  2.2 Reactive programming
3 Sensor Interpretation and Corridor Following
  3.1 *What-do-I-see*: Abstract perception command
  3.2 Corridor following
4 Robust abstract action
  4.1 *Go-to-next-node*: Basic movement in maze
5 Executing plans
  5.1 Introduction to *Mazeworld*
  5.2 Sequential and conditional plans
  5.3 Universal plans
  5.4 State-set tracking automata
6 Planning
  6.1 Sequential planning with certainty
  6.2 Sequential planning with uncertainty
  6.3 Conditional planning
7 Architectures for Interleaving Planning and Execution
  7.1 Assumptive programming architecture
  7.2 Interleaved planning architecture
8 Single-Robot Game Playing
  8.1 Introduction to *Gameworld*
9 Cooperation: Two Robot Game
  9.1 Communication protocol
10 Final contest
  10.1 Competition and collaboration

Fig. 3. Brief outline of the *Mobile Robot Programming Laboratory* class, as it was taught in Fall 2004.

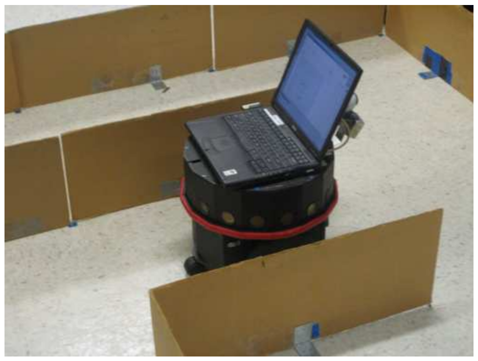
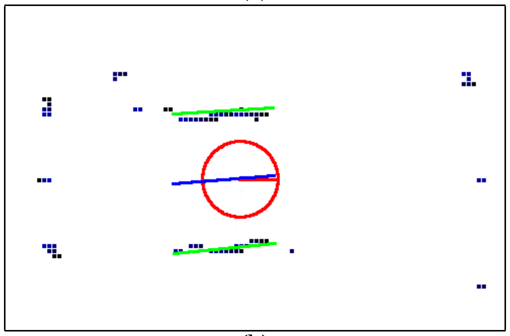

(a)

(b)

Fig. 4. Filtering of noisy sonar data. (a) Robot navigating in a maze. (b) Corresponding graphical interface showing the robot in red, the sonar data accumulated over time as the black dots. The blue center line is inferred from the two detected walls (green lines).

the robot was locally integrated to allow accurate displacement from one node to another.

3) Cognition: Because the robots are constrained to evolve in a structured, discrete environment, the low-level navigation is sufficiently facilitated and the course can graduate to AI-level programming assignments after the first third of the semester. Planning assignments begin with simply executing a hand-computed conditional plan that depends on the starting location of the robot in the maze. This quickly evolves to a fully-autonomous system that automatically interleaves conditional plan generation and execution. Basic AI tree-searching algorithms (depth-first, breadth-first, AND-OR, etc) are explored and implemented in a real-world context.

To facilitate the development process, many of this year's teams developed a simulator that could reliably replace the physical robot. They could then test their planning strategies off-line, without the need to be physically present in the lab. This example shows that students learned to develop testing tools for their application and reduce the time spent debugging.

B. Advanced concepts

In addition to the three fundamental concepts presented in Section III-A, students also explore more advanced robotics-related concepts that are often not directly part of the curriculum.

1) Robot observability: An important aspect of mobile robot programming learned by the students in this class is the psychology of diagnosing a misbehaving robot. To facilitate this process, students develop tools that improve *robot observability* [11], or the degree to which an outside observer can identify the evolution of the internal state of a robot. This can be achieved through the use of audio feedback using sound clips or a speech synthesizer: the robot would simply "speak" its actions or states. Data logging and visual interfaces are also very useful.

For example, a visual interface such as the one shown in Figure 4-(b) helped one team determine why their robot consistently stopped in the middle of an empty corridor. Because of sonar interference due to certain pathological configuration of the maze, a false short reading on the front sonar generated a false positive detection of an obstacle. Without the help of such a tool, diagnosing this problem would have been much more difficult. In fact, all of the teams surveyed reported the use of at least one form of interface. 86% of students felt that the interface was a very useful debugging tool.

Since the computers are connected to the robot using a serial port, students are often obliged to program directly on the robot to accelerate the debugging process. It has been observed that this actually helps them understand the robot better, because they are constantly watching its every move, thus improving *robot observability*. Although remote control of the robots was also available, students would prefer to "*sit down on the floor and code manually without any remote fashion*".

2) Property mapping: Throughout the semester, students learned how to map a set of robot percepts to outputs. This is achieved in a progression, where they first map low-level sensor inputs to motor output for position control and obstacle avoidance. They then repeat the same process with higher-level sensors and effectors (e.g. wall detectors and abstract actions such as *go-to-next-node*).

Even though it was not required, the team of which the first and second authors were members implemented a framework based on the suggestions of *property mapping*, a simple and language-independent approach proposed in [11] to map robot percepts to outputs. They discovered that it improved the observability of their robot. They realized that this framework was indeed very helpful when trying to determine the reason certain problems arose in particular situations. With the help of a data-logger, they could precisely determine the evolution of the internal state of the robot, and the reason why it made these decisions. As a result, debugging was much faster and more intuitive.

3) Reinforcement learning: Although not in the original curriculum, that same team also explored reinforcement learning by implementing a simple version of the well-known Q-learning algorithm [12]. The robot automatically learned the mapping from its percepts (obstacle on the left, right, front or no obstacle) to an output (wheel velocities). The learning process was implemented directly on the robot, and negative rewards were given when the robot came too close to a wall. In the end, a robot using the learned parameters would behave comparably to one that used carefully hand-tuned thresholds. As in the previous section, this example shows that the class allows enough flexibility for students to explore and learn about interesting new subjects, even if they are not part of the curriculum. This freedom adds to the student's sense of achievement and enjoyment of the course.

4) Multi-robot coordination: During the last part of the class, the teams are asked to build a two-robot team that competes against another team in the same maze (see Section II-A). It is therefore critical that a robust communication protocol be developed in order to synchronize the two robots towards a common goal. This section introduces students to evaluate different kind of control architectures, and deadlock and collision avoidance schemes. Because the robot population is homogeneous, egalitarian architectures were successful, although master-slave protocols were also tested.

It is now clear that the *Mobile Robot Programming Laboratory* class is a great educational experience for students to learn not only about robotics and software engineering, but also about problem-solving in general. The challenges they face require imagination, teamwork, observing their robot using different interfaces, and creatively elaborating solutions. Because mobile robots evolve in the real world, students have to learn to cope with a constantly changing environment, noisy sensors, and real-time constraints. Moreover, the structure of the class also gives them occasions to further explore subjects that arouse their curiosity, even if they are not explicitly in the curriculum. In the next section, we will look at the evolution of the class and the educational lessons learned.

IV. Evolution

The *Mobile Robot Programming Laboratory* class was first taught twelve years ago, and has since been offered yearly. The evolution of the class will now be presented under four different categories: the hardware environment, the programming environment, the final challenge (the game), and the teamwork experience. Finally, future directions will be discussed.

A. Hardware environment

The hardware on which the students work has a great influence on their learning experience. Throughout the years, different robotic platforms were used (see Table I). We now review the main platforms and their influence on the class.

The first models used were the Nomadic Technologies robots serial numbers 1 and 2. These are three-wheeled, synchro-drive robots, which means that the robots always face in the same direction and the wheels can turn independently of the robot's body. The next generation of robots, the Nomad 150, had the same motion system. However, it could also orient its sensor turret independently, which gave it an additional degree of freedom. This allowed a lot of variability in the solutions proposed by the students, even for simple assignments. For example, the common solution for a maze-wandering robot is to have the robot execute straight lines in corridors and stop before turning around each corner. However, the synchro-drive also allowed the implementation of a continously moving robot that would turn without stopping, which resulted in much faster performances. Higher number of degrees of freedom in the physical system can thus favor student's creativity by allowing a larger number of valid solutions to the same problem.

Figure 1 illustrates the current robotic platform: the Nomad Scout robot. It is driven by a two-wheeled differential-drive, and is therefore more restricted than its predecessors. As a consequence, since its introduction, the number of different solutions proposed by students has decreased significantly. The "continous robot" implementation, although still feasible, is much harder and less popular among students. However, this model was kept because of its smaller size, which in turn allows for much larger mazes and different challenges (see Section IV-C).

B. Programming environment

The programming environment also has a critical importance on the class. As illustrated in Table II, the programming language and environment underwent many changes.

LISP on Macintosh was the programming language originally used for this class. Its functional nature made it perfect for the application. Moreover, because each function can be executed independently from a command-line debugger, it was the ideal diagnostic tool to test fragments of code, and make sure the basic functions were working properly. This allowed students to identify and correct bugs very rapidly. Unfortunately, LISP was later replaced by C because there existed no reliable LISP programming environment on Windows at that time.

With C, programming suddenly became much harder, and the instructors witnessed an explosion of bugs in students'

Year	Robotic platform	Description	Sensors
1-2	Nomadic Technologies robots serial numbers 1 and 2	3-wheels synchro-drive	Infrared
3-6	Nomad 150	3-wheels synchro-drive, independent sensor turret	Sonars
7-11	Nomad Scout	Differential-drive, smaller size	Sonars

TABLE I

EVOLUTION OF THE ROBOTIC PLATFORM.

code. However, most of these bugs were not “robot-related”, but were instead due to bad memory and pointer management. The use of C++ tried to address that problem, but the learning curve for novice programmers then became too high, and there simply was not enough time for students to learn the language appropriately. Therefore, students would spend the semester struggling with programming language problems, and were not fulfilling the educational objectives of the class.

The advent of Java solved much of the problems cited above. Students already possessed good knowledge of it: 86% of surveyed students reported having prior Java programming experience ranging from intermediate to expert. Memory-management problems were rarely an issue due to Java’s garbage collector. The availability of easy-to-use graphical interfaces also made it the ideal tool for debugging fragments of code. Finally, the presence of many well-documented packages in the Java Software Development Kit (JDK) provide users with useful basic building blocks. It was observed that a lightweight Integrated Development Environment (IDE) equipped with a simple text editor with an easy interface to the compiler was preferable over big and slow IDEs. The latter are too bulky and memory-consuming for this kind of application and development process (often by trial and error).

Year	Programming environment
1-2	LISP on Macintosh
3-4	LISP on Windows
5-7	C and C++ on Windows
8-11	Java on Windows

TABLE II

EVOLUTION OF THE PROGRAMMING ENVIRONMENT.

Also related to the programming environment is the amount of code provided to the teams. Throughout the years, this has been kept to a minimum. Examples of given code are: a sample project to get students started; a maze and game editor, and a Java class that allows easy connection to the communication server (for multi-robot coordination). This way, students have to write their own code, which gives them complete ownership over the system they develop. It increases their responsibility: in case of a bug, they alone can fix it. In the end, the final product is complex, and students feel proud that they developed it entirely. 97% of surveyed students felt that they were provided the right amount of code throughout the semester.

In summary, the following programming environment aspects have a direct influence on the student’s learning experience. The students should already be familiar with the programming language in use for the class, because its learning curve might be too high, and students simply do not have enough time to learn it. Moreover, the language should be chosen such that students are faced with *robot-related* and not *language-related* bugs. Finally, fast debugging tools should be readily available.

C. Final challenge

The final challenge given to the students depended primarily on the available hardware. Initially, robots were competing one-on-one in a shared maze, and the goal was to reach a particular position in the maze first. Common strategies consisted of trying to block the opponent while progressing towards the goal. However, because the robots were equipped with infrared sensors, they would be “blinded” as soon as they moved within each other’s line of sight.

With the advent of RF devices for communication, the next challenge was to build a team of two robots which would compete against an opposing team in a shared maze. This proved to be very hard for the students, because the robots could not know the location of their opponents, and thus frequent collisions occured.

Each team of robots was then put in separate mazes, and the quality of games improved dramatically. The challenge changed: robots now had to pick up “gold” pieces scattered in the maze. At first, they had to ask a human to pick the gold, but they were soon equipped with magnets and had to perform the job themselves. Many games ended up with only a few seconds in difference. Failures were now due mostly to problems in communication protocols. Figure 5 shows one example of a maze used during a contest. The robots must agree on who goes out of their starting location first, or there will be a collision right at the beginning.

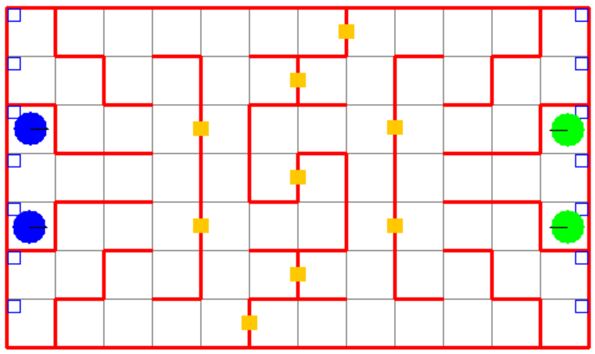


Fig. 5. Example maze used during the 2004 edition of the contest. The blue (green) circles indicate the robots’ starting location. Robots with same color are part of the same team. Walls are indicated in red, and gold pieces are in orange. The blue empty squares indicate locations where one gold (maximum) can be dropped off. Note that most gold pieces are accessible by both teams, and that the teams are completely isolated from each other.

The shared maze challenge was tried once more a few years later, but it was still too hard, because the opponents were invisible. Because it is a source of motivation and pride to the participating students, it is important that the final assignment be hard, but it must also be feasible. As experience has shown, overly difficult assignments may be frustrating to students because, given the particular hardware configuration, no solution may exist to solve the problem. Possible improvements to the shared-maze challenge might be to broadcast the robot's position to every robot in the game, or to use vision to detect opponents.

D. Teamwork

Learning how to work efficiently in teams is crucial in many endeavors, and the *Mobile Robot Programming Laboratory* class provides a great teamwork opportunity to students. It has been determined that three-member teams provide the best teamwork experience. With a larger number (up to five), there would often be splits within teams, resulting in one or more students being left out or the creation of non-communicating subteams. A lower number was not enough to complete the task.

One interesting fact, also noted in [6], is that the team members should all be of the same gender whenever possible. Generally, we observe that all-female teams are more likely to encourage its member's active participation. We confirmed that observation with a quantitative test: same-gender team members felt that their team was more efficient than mixed-gender teams (Student's t test: $p = 0.05$, $n = 37$). It was also observed that the definition of roles (ex: robot hacker, master programmer, strategist, etc.) before team formation helped build balanced teams. A team performed more efficiently if all its members' strengths were compatible. Students from successful teams commented that they "*communicated well and knew how to focus on each other's strenghts*".

E. Future directions

We now briefly look at future directions the course could take, as it continues to be taught at Carnegie Mellon University. As presented in previous sections, many elements such as team formation, programming environment and hardware platform should be very similar, because the course now has reached a level of stability on those issues.

Over the years, the continuous increase in computing power has made the planning problem much easier. With the current hardware configuration, computing a plan across the whole maze of Figure 5 takes less than one second, and can be computed live on-board the robot. Therefore, it is no longer necessary to interleave planning and execution, as it was before, rendering the final contest easier. This problem should be addressed in the future, to keep the assignments challenging. A solution to this problem would be to introduce cooperation earlier in class, and have the teams compete in a shared maze. The robots would therefore have to create plans that take their opponents expected or possible movements into account. Moreover, teams could make more use of their cameras and detect the opponents with more advanced computer vision algorithms.

V. Conclusion

In summary, this paper presented the *Mobile Robot Programming Laboratory* class taught at Carnegie Mellon University for the past twelve years. We have shown that it is an essential part of an organized curriculum in robotics.

The presentation of its current status and the in-depth description and examples of the basic and advanced concepts learned by the students have shown that it is a very important part of a broad robotics curriculum. In addition, by providing students with hands-on experience with real robots, it also develops their problem-solving, teamwork, and observation skills. It has also been noted that flexibility is an important aspect, as it allows students to explore more particular subjects that arouse their curiosity. Furthermore, the evolution of the class over the years has taught us important lessons about this kind of class that are also relevant in different applications. The hardware and programming environment should allow students to develop their creativity. The assignments should be challenging, but within reach, and teams should be well-balanced and have a small number of members (three, in our case). It is hoped that the lessons learned and important concepts presented in this paper might be of use to others pursuing similar educational goals.

Acknowledgments

The authors would like to thank Professor Alonzo Kelly, who taught the class in the Fall 2004, for his invaluable feedback and support during the entire semester. Special thanks to Kashyap Chandrasekar, the third member of this team, and to Stuart Anderson, Thomas Stepleton, and Greg Armstrong for their help.

References

[1] T. Fong, I. Nourbakhsh, and K. Dautenhahn, "A survey of socially interactive robots," *Robotics and Autonomous Systems, Special issue on Socially Interactive Robots*, vol. 42, no. 3, 2003.
[2] I. Nourbakhsh, "Robotics and education in the classroom and in the museum: On the study of robots, and robots for study," in *Proceedings, Workshop for Personal Robotics for Education. IEEE ICRA*, 2000.
[3] M. Rosenblatt and H. Choset, "Designing and implementing hands-on robotics labs," *IEEE Intelligent Systems and their Applications*, vol. 15, no. 6, November-December 2000.
[4] U. Wolz, "Teaching design and project management with LEGO RCX," in *Proc. SIGCSE Conference*, 2000.
[5] C. Stein, "Botball: Autonomous students engineering autonomous robots," in *Proceedings of the ASEE Conference*, 2002.
[6] I. Nourbakhsh, E. Hamner, K. Crowley, and K. Wilkinson, "Formal measures of learning in a secondary school mobile robotics course," in *IEEE International Conference on Robotics and Automation*, 2004.
[7] "6.270 - MIT's autonomous robot design competition," Massachusetts Institute of Technology, 2006. [Online]. Available: http://web.mit.edu/6.270/
[8] B. Maxwell and L. Meeden, "Integrating robotics research with undergraduate education," *IEEE Intelligent Systems and Their Applications*, vol. 15, no. 6, November-December 2000.
[9] C. Cosma, M. Confente, D. Botturi, and P. Fiorini, "Laboratory tools for robotics and automation education," in *IEEE International Conference on Robotics and Automation*, 2003.
[10] J. Fernández and A. Casals, "Open laboratory for robotics education," in *IEEE International Conference on Robotics and Automation*, 2004.
[11] I. Nourbakhsh, "Property mapping: A simple technique for mobile robot programming," in *Proceedings of AAAI*, 2000.
[12] S. Russel and P. Norvig, *Artificial intelligence A modern approach*, 2nd ed. Prentice Hall, 2003.