Envisioning the Roomba as AI Resource: A Classroom and Laboratory Evaluation

Ben Tribelhorn and Zachary Dodds
Harvey Mudd College Computer Science Department
301 Platt Boulevard
Claremont, CA 91711
btribelh@cs.hmc.edu, dodds@cs.hmc.edu

Abstract

This paper investigates the suitability of iRobot's Roomba as a low-cost robotic platform for use in AI research and education. Examining the sensing and actuation capabilities of the vacuum base led us to develop sensor and actuation models more accurate than those provided by the raw API. We validate these models with implementations of Monte Carlo Localization and FastSLAM, algorithms that suggest the Roomba's viability for AI research. Classroom trials incorporated the Roomba into CS 1 and CS 2 courses in the Spring of 2006, and student feedback has been similarly promising for educational uses. While the platform has both benefits and drawbacks relative to similarly-priced alternatives, we conclude that the Roomba will interest many educators, especially those focusing on the computational facets of robotics or applications involving large, homogeneous groups of physical agents.

Introduction

iRobot's Roomba vacuum (Figure 1) represents the growing ubiquity of robotics perhaps better than any other single platform. Over two million Roombas clean floors in homes and businesses. The platform has become a standard for task-based, low-cost robotics: imitators have been quick to follow.

With this success as a backdrop, iRobot published a *Serial Command Interface* API for the Roomba in January of 2006 (iRobot 2006). (It is now called the *Roomba Open Interface*.) This API enables programmatic access and control over almost all of the robot's sensors and motors. This paper reports our initial experiments to assess the suitability of the Roomba as a classroom and research resource.

In this process of testing, we have developed Python drivers for the Roomba and have used the platform in two undergraduate courses. We built empirical models of sensing and actuation that improve upon using the system's raw kinematics and odometry. To provide focus to these experiments, we have used the Roomba as a testbed for several spatial-reasoning algorithms of current interest to the AI Robotics community. Thus, this paper provides context for several resources now available to AI educators and researchers:

- Classroom-tested, cross-platform drivers and support software for the Roomba
- Sensor and actuation models that improve upon the published *Roomba Open Interface* serial API
- Implementations of localization, mapping, and vision algorithms that have been tested on the Roomba

Based on these resources and the perspective of the past year of use, we conclude that the Roomba is a promising alternative to the many other low-cost robot platforms available for research and education (Figure 2). As presented, the Roomba will interest practitioners whose focus lies in the computational and/or applied robotics.

Sensors/Inputs

Wheel encoders (2)	Buttons (3-4)
Bump sensors (2)	Dirt-detection (1-2)
IR Wall sensor (1)	IR receiver (0-255)
Cliff sensors (7)	Virtual wall
Electrical (5)	Remote Control

Actuation/Outputs

Drive wheels (L & R)	LEDs (5-7, color)
Cleaning motors (3)	Piezospeaker

Figure 1: **Left** The Roomba, as available off-the-shelf for US$150, along with **right** its built-in sensory and actuation abilities. Proprioception is both capable and complete. Yet it comes with almost no sensing that reaches beyond the platform itself.

Background

Robotic vacuums are becoming increasingly pervasive. Indeed, vacuuming has been cited as the field's "killer app," bringing more robots into homes than any other (Grossman 2002). The enabling robotics research spans architectural insights (Brooks 1986), location-aware vacuuming (Domnitcheva 2004), a large number of environmental-coverage algorithms (Choset *et al.* 2005), and even outright predictions, now fulfilled (Brooks 1986). With iRobot's January, 2006 release of the serial API for its Roomba plat-

Platform	Cost	Sensing
Lego RCX	$200	Bmp,Lt
Roomba	$230	Bmp,Enc,Vis,Mic,WL
Lego NXT	$250	Bmp,Lt,Son,Enc,WL
Intellibrain	$300	Bmp,Lt,IR,a2d,WL
PalmPRK	$325	IR,a2d
HandyBoard	$350	Bmp,Lt,IR,a2d
KIPR XBC	$500	Vis,Bmp,Lt,IR,Enc
UMN eRosi	$500	Lt,Enc,Pyr,WL
HandyBoard2	$750	Vis,Bmp,Lt,IR,Enc,a2d,WL
Hemisson	$780	Lt,IR,WL
Garcia	$1725	Vis,IR,Enc,WL
Khepera	$2000	IR,Enc
AIBO	$2000	Vis,Mic,Bmp,Enc,WL

Figure 2: A comparison of several inexpensive robot platforms/controllers, their costs, and their standard set of sensing capabilities. **Legend**: **Bmp**, bump or tactile sensing; **Lt**, light sensing; **Vis**, vision; **Mic**, microphone; **Enc**, encoders or odometry; **WL**, wireless communication with a controlling PC; **a2d**, general analog/digital inputs; **IR**, infrared range sensing; **Pyr**, heat or flame sensing. The vision (and microphone) sensors on the Roomba are available from an onboard laptop computer, e.g., as in Figure 10.

form (iRobot 2006), there is now an opportunity for robotics researchers and educators to benefit from the successes of the home-robot industry.

This API makes any Roomba an ordinary serial device, though vacuums assembled before 10/2005 require a firmware upgrade. The API specifies a byte-level protocol; this protocol has been incorporated into software drivers written in Java (Kurt 2006), C++ (within Player/Stage) (Gerkey 2006), and Python (Dodds & Tribelhorn 2006b). Even before the release of the API, the worldwide community of robot enthusiasts had shown the platform's promise (Gerkey 2006) (Mecklenburg 2005). More recently, Bluetooth, USB, and RS232 serial interfaces have become commercially available (RoombaDevTools 2006). Figure 3 summarizes these devices and their current costs.

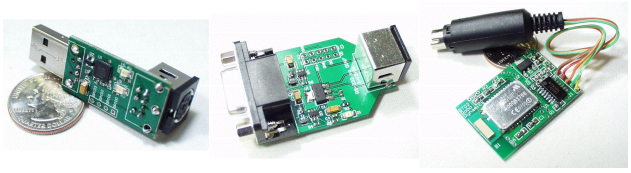

Figure 3: Commercially available USB, RS232, and Bluetooth serial interfaces to the Roomba provide researchers and educators an inexpensive platform that requires no custom hardware or construction at all. When obtained with a $150 Roomba, these devices are US$10, $5, and $80, respectively (RoombaDevTools 2006).

With iRobot-based software support and third-party hardware interfaces, off-the-shelf Roombas are now programmatically accessible without any modification whatsoever. Whimsical applications have emerged: the Roomba has been adapted to playing the game Frogger on a busy street (Torrone 2006). But it is not obvious that the Roomba can serve as a resource for validating AI researchers' algorithms or enabling students to implement and experiment with those algorithms in a lab setting. To our knowledge, this paper is the first to address these questions.

Communication Performance

Drivers

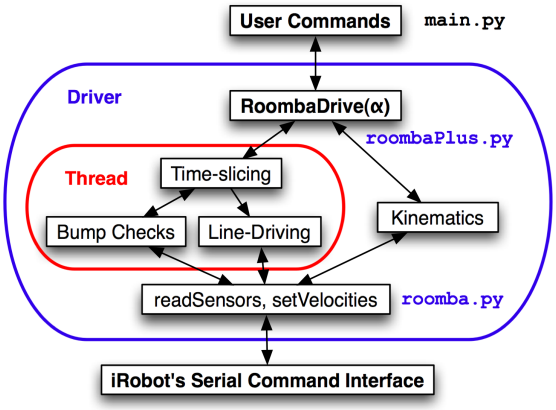

Figure 4: The architecture of our Python-based software driver for the Roomba platform.

To test our Roombas we have written two Python layers atop iRobot's byte-level serial command interface (now known as the *roomba open interface*). The lower layer provides full access to the Roomba's sensors, speaker, motors, and built-in behaviors. Our upper layer allows for straight-line translation (oddly, not part of iRobot's provided API), and it includes our odometric correction model described in the following section. Figure 4 summarizes the software architecture; the code itself is freely available from (Dodds & Tribelhorn 2006a).

Bluetooth

The most flexible method of communication is Bluetooth, which uses the unlicensed 2.4 GHz ISM (Industrial Scientific Medical) band. Figure 3's Bluetooth device, nicknamed the *RooTooth* is Class 1, allowing a range of up to 100 meters. Connection quality over distance drops slowly and our tests indicate that adequate connections can be made at up to 200ft. In theory, the number of Bluetooth devices that can be used simultaneously is large, as there are 79 channels available. Our tests have demonstrated that a single laptop can easily interact with multiple devices; we tested 5 concurrently in a small, confined space without interferences or reduction in throughput to the individual Roombas.

Throughput

USB polling of the Roomba's full suite of sensors averages a throughput around 66 hz; Bluetooth is considerably slower.

A single RooTooth will only peak at 16 hz in Fast Data Mode and 6 hz in its normal mode. Thus despite the additional setup effort, an onboard laptop not only can provide a significant boost to a Roomba's sensing capabilities, it enables more effective use of existing sensors because of the considerably faster polling rate.

Simulation

Building atop our python drivers, James Snow has created an interface to the Roomba available within the Python Robotics (Pyro) toolset (Snow & Fossum 2007). Player/stage support is also available (Gerkey 2006). Although both of these resources provide a capable and sophisticated interface to all of the hardware platforms they support, we used a simple, homegrown simulator for the Roomba as a testbed for our algorithm implementations. This pure-Python, 2d visualizer (depicted in Figure 9 and at right in Figures 8 and 11) can emulate the platform's sensing and is packaged with the drivers at (Dodds & Tribelhorn 2006a).

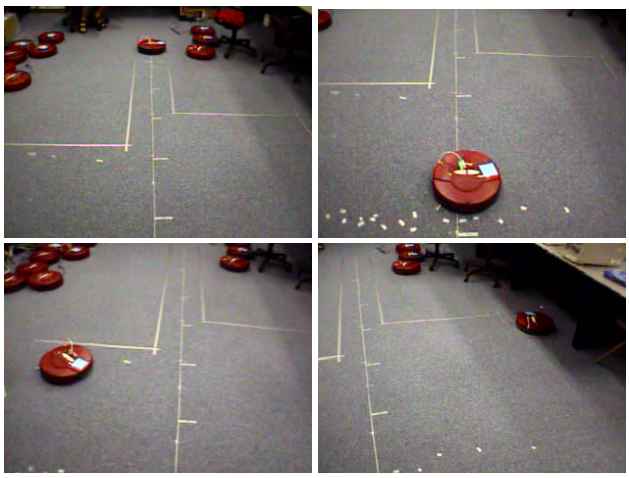

Figure 5: **Upper left** The central robot shows the starting position for all three runs, each heading straight towards the viewer **Lower left** The final position after the straightest-possible run with right-hand bias **Lower right** The result of the same command with left-hand bias **Upper right** The final position after 3 meters of straight-line translation using the odometric model presented in this section.

Modeling the Roomba

A small number of discrepancies between the Roomba's published programming interface and the platform's behavior motivated our own modeling of its actions and accuracy. For instance, the API cites special codes to indicate straight-line driving (iRobot 2006, p. 4). However, as Figure 5 attests, actual behavior did not differ from those commands' large radius of curvature (ROC) limits, published at two meters. Discussions among many experimenters have confirmed the difficulty of straight-line motion through the API alone. Second, we found that Roombas within a single production run are quite consistent in their behavior. However, those from different production runs can be very different in their responses to identical API actuation commands under identical environmental conditions.

Odometry

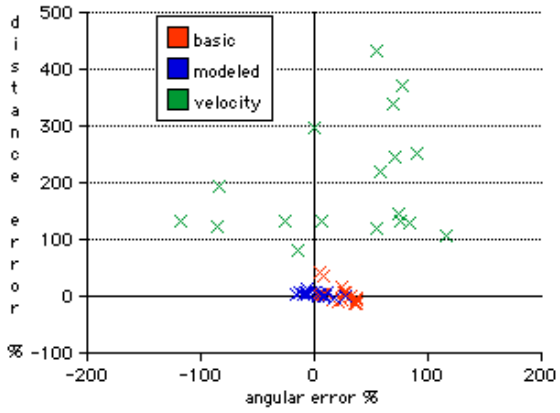

Figure 6: Comparison of basic kinematic odometry, velocity-integrated odometry, and our modeled odometry on 18 runs of two robots from separate manufacturing batches.

Roombas do provide odometry. Unfortunately, there is a huge bias in translating left and right turning around the maximum radius of curvature (ROC), e.g., between $-30°$ and $80°$ over 15s at 20cm/s. Our *RoombaDrive* software layer compensates for this *lean* by time-slicing left and right turns at the maximum ROC. Denoting this time-slicing parameter α, the resulting odometric models map from raw *angle* and *distance*, available from the API, to actual angle θ and distance r (in cm) as follows:

$$r = \frac{distance}{10.0} * (0.705 + \alpha - \alpha^2) - 5 \quad (1)$$

$$\theta = \frac{angle}{129.0} * (0.705 + \alpha - \alpha^2) \quad (2)$$

These models are derived in detail in (Tribelhorn & Dodds 2007). In essence, the 5cm represents constant slippage and the constrained quadratic factor represents a compensation for the *lean* of each individual robot, α. Figures 6 and 7 demonstrate the improvement of this model over velocity integration and the API's unmodified odometric feedback.

Local Sensing

The Roomba comes solely with local sensing in the form of bump and IR sensors. The bump sensors have four states which result from a left and right sensor attached to a rigid bumper. These states are left, right, front, or no bump. The collision angle that will produce a front bump varies with the sampling rate. For an ideal situation, bumps within the cone of $\pm 20°$ cause both sensors to trigger (due to bumper rigidity). However at sampling rates of around 4Hz or less,

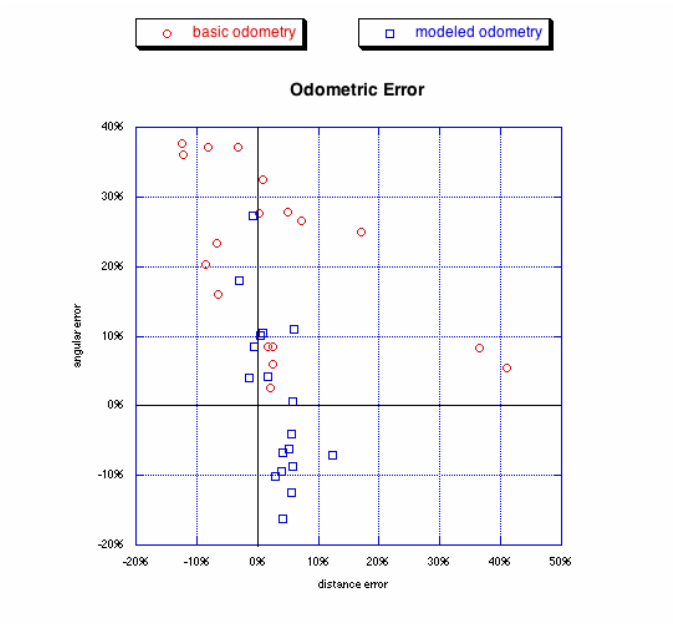

Figure 7: Here raw odometry demonstrates noticeable skew (21.5%) in comparison to our modeled, adjusted odometry.

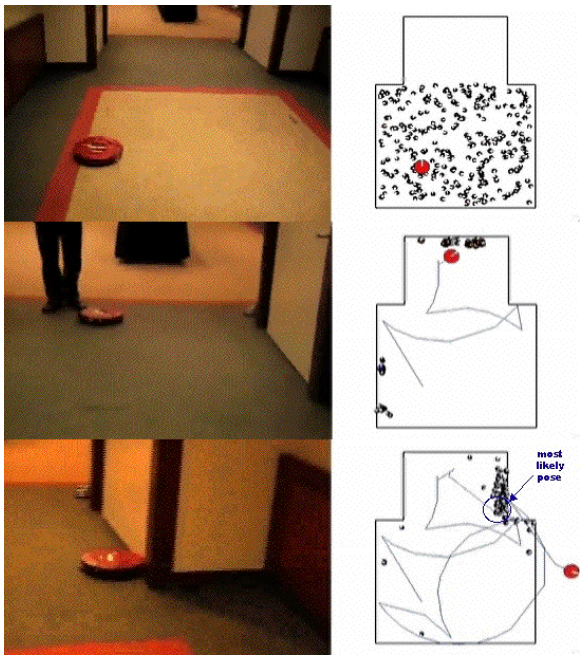

Figure 8: Recorded at AAAI 2006, this shows a Roomba successfully localizing itself using MCL. In the middle image note that the wall in question is actually a virtual wall, encountered as an observer paused to watch the action.

a front bump will be detected at a range of $\pm 60^\circ$ or more. If the response to this bump is similarly slow, the robot will usually slip and actually end its motion facing the wall which can adversely effect odometry. Our modeled odometry lessens, but does not eliminate, this effect.

Algorithmic Validation: MCL and FastSLAM

Monte Carlo Localization (MCL) is a probabilistic estimation of pose in which Bayesian updates combine uncertain sensing and odometric information (Thrun *et al.* 2001). Using only the Roomba's local sensing (tactile and odometric), we implemented and demonstrated successful MCL pose tracking at AAAI 2006. An example of one such run is shown in Figure 8.

Both our AAAI and lab runs used a 35% uniform-error model in distance and 25% in angle to compensate for the inaccuracy of the naive kinematic model on which it relied. These large uniform errors were necessary to create sufficiently robust particle coverage for MCL in these two environments. The relatively large (and computationally expensive) population of 300 particles was needed to successfully localize the Roomba. This demonstrates the substantial advantage of using off-board computation: the onboard microcontrollers that support smaller low-cost platforms would struggle to hold and update this population in real time.

Adding Vision

In order to tackle mapping (more precisely, simultaneous localization and mapping), most algorithms of current interest rely on richer sensing than the Roomba provides out-of-the-box. We chose to mount a laptop on the Roomba with velcro to attain enough onboard processing for vision. This setup with Apple's iSight camera can be seen in Figure 10.

We discovered that the iSight provides pixels natively in YUV 4:2:2 colorspace, a lossy encoding of RGB data. The YUV data width provided is 16 bits: a pair of UY or VY bytes alternately for each pixel. This loss in color information is generally undetectable by the human eye because our eyes are more sensitive to the luminance (Y) than differences in color (UV). Initially we converted the YUV values to RGB, but color segmentation does not improve with RGB thresholding. To save on computation we reverted to using the raw YUV values.

Using Mac OS X as this project's primary development environment, we have devloped a C library that enables access to the pixel values provided by the iSight camera (or any other QuickTime input). These drivers are available from (Dodds & Tribelhorn 2006a). To our knowledge there is prior work in this area, but it has not allowed for pixel-level access to image streams, e.g., (Heckenberg 2003).

Atop these low-level routines, a set of vision algorithms find the largest connected components segmented by color and calculates a set of shape statistics on that object which are used for classification. For example, segmentation on "red," suitably defined, will match a teddy bear's clothing or a piece of construction paper. Figure 10 summarizes the visual statistics computed.

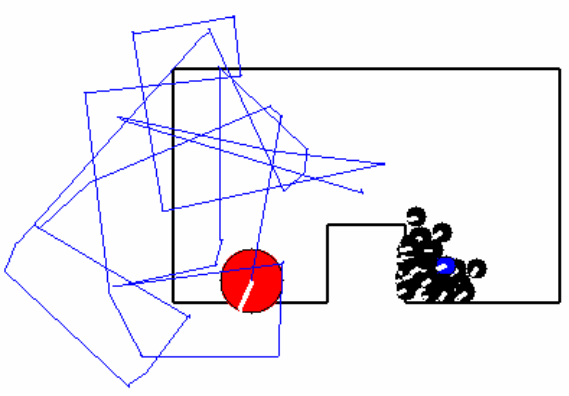

Figure 9: An MCL run in another environment (a "maze" in our lab). The most probable pose (the small blue circle among the 99 other less likely hypotheses) was indistinguishable from the ground-truth pose, while the odometric estimate (the large red circle) resulted in a location far from the actual robot.

FastSLAM

With visual input now available, we implemented FastSLAM 1.0 with known data correspondence as detailed in *Probabilistic Robotics* (Thrun, Burgard, & Fox 2005).[1] FastSLAM uses a point-feature tracker, so vision is a natural fit. We created visual landmarks by placing construction paper on the walls as seen in Figure 11. In a joysticked run of the robot, the vision system correctly identified and plotted the poses and uncertainties of the four landmarks as shown in Figure 11. The loop closure that occurs when the red landmark is seen for a second time significantly reduces that feature's pose uncertainty.

Bump-only Roomba Mapping

Mapping without vision on the Roomba presents a stiff challenge because of the platform's lack of built-in range sensing. We have designed a preliminary set of mapping algorithms using only local bump sensing and odometry. To compensate for this impoverished sensory data, we assume strong prior knowledge about the environment:

- that it consists only of straight-line walls.
- that all of those wall segments are either parallel or perpendicular.

These assumptions allow several interpretations of the incoming data, e.g., line fitting to raw odometry of the bumps. Our results from this algorithm and other variants are presented in (Tribelhorn & Dodds 2007).

Educational Trials

The Roomba was used as the basis for several assignments in a CS1/CS2 course sequence taught at Chatham College, an all-women's institution in Pittsburgh, PA. Low cost was one reason for choosing the Roomba. The more compelling reason, however, was that the Roomba, as a simple serial peripheral, integrated effortlessly into the environment in which these courses were already being taught.

[1] Five notational/typographical errors were identified in this version of the algorithm; these have been reported to the book's authors and website.

Shape Statistic	Value	Shape Statistic	Value
best ellipse angle	23.1°	pixel count (area)	1918
major axis length	76.5	roundness	0.42
minor axis length	45.5	color label	"red"

Figure 10: **Top left** Mounting a laptop and the iSight on the Roomba **Top right** The largest connected component defined as "red" is highlighted in blue. Smaller red components are green. **Bottom** The shape statistics computed for that largest connected component.

This CS1/CS2 trial included an external assessment effort to determine the extent to which the Roomba (and robots in general) affected students' feelings and capabilities in learning introductory computer science. The results have shown that the physical interactions had a significant impact. One student indicated that the impact was intellectual:

> Like when you're just working on the screen it's like 'oh the little dot is moving.' When you're working with the actual thing [the Roomba], you're like okay, problem solving. Because it's a lot easier to solve the problem if it's 3D, in front of you, and you can see exactly what the problem is.

Another student described the robot's impact in affective terms: "Playing with the Roomba made it a lot more fun."

A third student pointed to overcoming some of the Roomba's idiosyncracies when asked *Which activities do you think have been most useful this semester in making you a better programmer?*:

> I would say that probably working with the Roomba definitely just because the first day we worked with it we were trying to get it to go in a straight line because it has like a natural curve to it so it doesn't go straight.

Overall, the Roomba added excitement to the classes, and it provided hands-on, task-specific applications for the programming concepts covered. Moreover, the Roomba did **not**

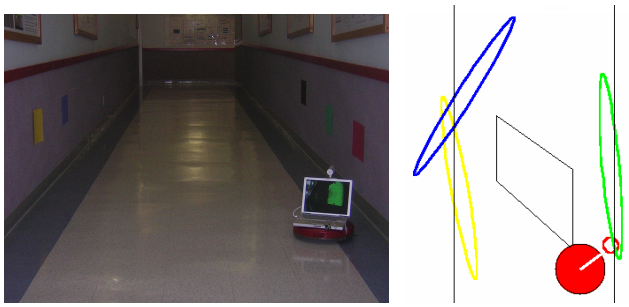

Figure 11: **Left** Five landmarks, distinguished by color, placed along a hallway. Only four were used. The segmentation of the red landmark appears on the monitor; the robot is in the location of its second sighting of that red landmark, an event which "closes the loop," reducing the uncertainty in the potential location of that map feature. **Right** A closed-loop run of vision-based FastSLAM. Note that the ellipses themselves, representing the pose uncertainty of the similarly-colored landmarks, are the most-likely computed map. FastSLAM maintains many such maps in a particle filter, each with a distinct odometric run, here shown as the diamond-shaped quadrilateral. The thin vertical black lines are not used in the computation; they are the "real" map, superimposed for comparison. In this case, color was used to provide the correspondence between different sightings of the individual landmarks.

add the time-intensive overhead of constructing and maintaining Lego-based or other hand-built platforms, nor did it require us to change the programming language or OS on which the class was based. In contrast to many of the other platforms in Figure 12, the Roomba can be used to support an *existing* CS and AI curriculum, rather than requiring a curriculum designed especially for it.

Perspective

These extensions and applications of the Roomba only scratch the surface of what is possible, enabling users an inexpensive basis on which to design systems that run "with our initiation, but without our intervention." (Brooks 1986) As this paper demonstrates, even the ubiquitous, unmodified Roomba platform can support far more than the vacuuming tasks for which it was designed. As an educational resource, the Roomba is pedagogically scalable: it is as suitable for reinforcing beginning programming concepts as it is for exploring algorithms of current interest to the robotics community. As a research resource, the Roomba empowers investigators who want to *use* robots, rather than build them. For example, it offers researchers involved in the fields of multi-agent systems, HRI, or many other subfields of AI and CS an off-the-shelf means to embody and test their work without having to spend time constructing or modifying hardware.

Ultimately, the Roomba offers the robotics community both an example of the widespread commercial viability of autonomous robots and a novel resource we can leverage toward our educational and research goals. It heralds the advent of *robotic peripherals* that can take advantage of all of the computing power and cost-efficiency of today's commodity laptop and desktop machines. This paper provides an improved odometric model of the Roomba, some strategies for handling its idiosyncrasies, and and an initial assessment of the Roomba's capabilities. We believe it won't be long before there emerge a wide variety of applications of this modest platform.

Acknowledgments

This work was made possible by funds from NSF DUE #0536173, as well as funding and resources from Harvey Mudd and Chatham Colleges.

References

Brooks, R. 1986. Achieving Artificial Intelligence through Building Robots. Technical report, Massachusetts Institute of Technology, Cambridge, MA, AI-Memo 899.

Choset, H.; Lynch, K.; Hutchinson, S.; Kantor, G.; Burgard, W.; Kavraki, L.; and Thrun, S. 2005. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press.

Dodds, Z., and Tribelhorn, B. 2006a. Erdos. http://www.cs.hmc.edu/∼dodds/erdos.

Dodds, Z., and Tribelhorn, B. 2006b. Erdos: Cost effective peripheral robotics for AI education. In *Proceedings, 2006 AAAI*, pp. 1966–1967.

Domnitcheva, S. 2004. Smart vacuum cleaner - an autonomous location-aware cleaning device.

Gerkey, B. 2006. Mapping with the iRobot Roomba. http://www.ai.sri.com/∼gerkey/roomba/index.html.

Grossman, L. 2002. Maid to order. *Time Magazine* September.

Heckenberg, D. 2003. Using Mac OS X for Real-Time Image Processing. In *Proceedings of the Apple University Consortium Conference*.

2006. Roomba SCI specification. www.irobot.com/hacker.

Kurt, T. 2006. *Hacking Roomba: ExtremeTech*. Wiley. to appear.

Mecklenburg, P. 2005. Roomba SLAM. http://www.cs.unc.edu/∼prm/roomba/roomba-slam.pdf.

2006. RoombaDevTools. www.roombadevtools.com.

Snow, J., and Fossum, T. 2007. How platform-independent is pyro? In *Proceedings, AAAI 2007 Spring Symposium, "Robots and Robotics: Resources for AI Education"*, (to appear).

Thrun, S.; Fox, D.; Burgard, W.; and Dellaert, F. 2001. Robust monte carlo localization for mobile robots. *Artificial Intelligence* 128(1-2):99–141.

Thrun, S.; Burgard, W.; and Fox, D. 2005. *Probabilistic Robotics*. MIT Press.

Torrone, P. 2006. Roomba Tronic. *Make Magazine* Volume 06: Robots.

Tribelhorn, B., and Dodds, Z. 2007. Evaluating the Roomba: A low-cost, ubiquitous platform for robotics research and education. submitted to ICRA 2007.