

Roomba and Mac OS X: Cross-platform Vision and Robotics for AI

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 both research and education. We present sensor and actuation models and implement algorithms that demonstrate the Roomba's viability. 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 robotic 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). This API enables programmatic access and control over almost all of the robot's sensors and motors. This paper reports the initial experiments we have run to assess the suitability of the Roomba as a classroom and research resource.

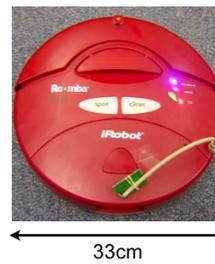
In the process of this testing, we have developed Python drivers for the Roomba and have used the platform in two undergraduate courses. In addition, experiments with the platform have yielded empirical models of sensing and actuation which improve upon using the system's raw kinematics and odometry. To provide focus to these experiments, we have used the Roomba as a testbed for several spatial-reasoning algorithms.

As a result, both research and educational communities can now take advantage of the Roomba using the following contributions:

- Cross-platform software drivers
- Sensor and actuation models
- Implementations of localization, mapping, and vision algorithms

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Based on these resources and the perspective of eight months of use, we conclude that the Roomba is a promising alternative to the many other low-cost robot platforms available for research and education, particularly for researchers and educators whose focus lies in computational or applied facets of 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: The Roomba available off-the-shelf for US\$150, along with 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.

Related Work

Robotic vacuums are increasingly pervasive; location-aware autonomous vacuuming has been done on the Trilobite and Roomba platforms. (Domnitcheva 2004) The Roomba has even been adapted to playing the game Frogger on a busy street (Torrone 2006).

Probabilistic techniques are a promising fit for the Roomba; previously, variants of SLAM have been used in conjunction with the Roomba by installing a laser range finder (Gerkey 2006) and by mounting stereo vision (Mecklenburg 2005). Even among these efforts, to our knowledge this work represents the first evaluation of the Roomba per se for research and educational purposes.

Communication and Control

Hardware

Three months after iRobot released the Roomba's serial API, commercially available hardware interfaces simplified communication with the platform. Figure 2 illustrates several

of the available connectors: USB, Bluetooth, and RS232 (RoombaDevTools 2006).



Figure 2: 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 Roomba, these devices cost US \$10, \$5, and \$80, respectively (RoombaDevTools 2006).

The most flexible method of communication is Bluetooth, which uses the unlicensed 2.4 GHz ISM (Industrial Scientific Medical) band. Figure 2’s Bluetooth device, nicknamed the *RooTooth* is Class 1, allowing an optimal range of 100 meters. Connection quality over distance drops slowly and our tests indicate that adequate connections can be made at up to 200ft. The number of Bluetooth devices that, in theory, 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 without a reduction in throughput to the individual Roombas.

Drivers

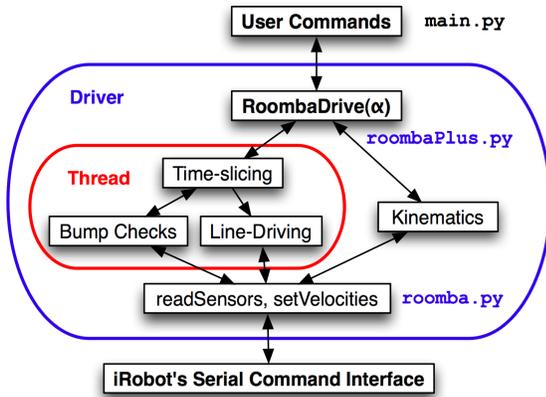


Figure 3: The architecture of our Python-based software driver.

We have written two Python layers atop iRobot’s byte-level API. The lower layer provides full access to the Roomba’s sensors, speaker, motors, and built-in behaviors. Our top 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 3 summarizes the software architecture; the code itself is freely available at (Dodds & Tribelhorn 2006).

Throughput

USB polling of the Roomba’s full suite of sensors averages a throughput around 66Hz; Bluetooth is considerably slower. A single RooTooth will only peak at 16Hz in Fast Data Mode and 6Hz in its normal mode. A possible solution to this limit in bandwidth would be to mount a micro-controller which reacts to the sensors for stopping quickly and allows higher level decisions to be made by a computer which would only send and receive a less rich set of data. In this paper, however, we maintain focus on the capabilities of the unmodified Roomba.

Simulation

Building atop these python drivers, James Snow has created an interface to the Roomba available within the Python Robotics (Pyro) toolset. In addition, the Roomba is controllable via the player/stage simulator. 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 Python 2D visualizer (depicted at right in Figure 6) can emulate the platform’s local sensing and is packaged with the drivers.

Modeling the Roomba

Odometry

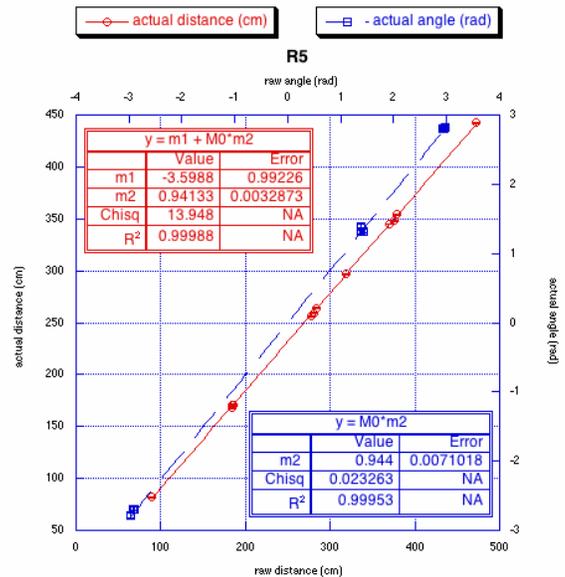


Figure 4: Regressions of translation and rotation yield m , the robot’s *lean* for a specific Roomba, named R5. We created equations for odometric distance and angle from similar analyses across several robots.

Roombas do provide odometry. Unfortunately, there is a huge bias in translating left and right turning around the maximum radius of curvature (ROC) (80° vs. -30° over

15s at 20cm/s). Our "RoombaDrive" layer compensates for this bias by enabling both simple rotation and straight-line translation. Because the API does not provide straight-line translation, it is achieved in software by time-slicing left and right turns at the maximum ROC. This time-slicing parameter is denoted α .

Running several tests of our line driving and turning code allowed us to extract a motion model. In Figure 4, data from a single robot is depicted. Linear regressions of raw data versus actual position in linear translation and rotation shows that the slope m is constant between line driving and turning. We ran the tests over multiple robots of different batches and found this correlation to be consistent.

Equations for distance and angle (r, θ) as functions of the robots' raw data and the robot specific α become the following:

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

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

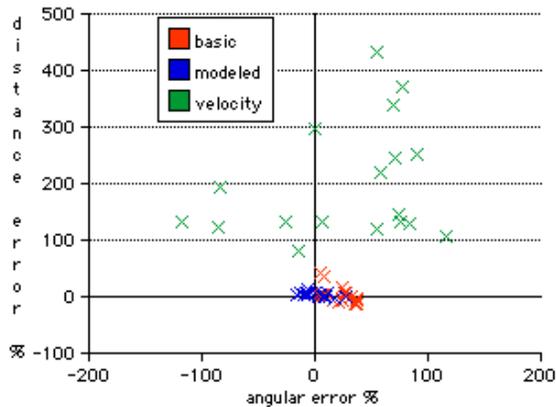


Figure 5: Comparison of basic kinematic odometry, velocity estimated odometry, and our modeled odometry on two robots from separate batches. For this sample $N = 18$.

Results from testing of our motion model against basic odometry and velocity-based odometry are illustrated in Figure 5. The sampled error for naive odometry is $\sigma_r = 11.5\%$ and $\sigma_\theta = 12.4\%$ which depends strongly on the specific robot. Integrating the commanded velocities to estimate position results in even more significant errors, as shown. Our corrected model has error of $\sigma_r = 2.9\%$ and $\sigma_\theta = 6.1\%$, which significantly improves odometric accuracy.

Local Sensing

The Roomba has only local sensing in the form of bump and IR sensors. When the Roomba bumps it will regularly slip as it pushes forward and this causes the robot to rotate. This rotation is not measured by the odometry and as such must be compensated by increasing error within the software. The IR sensors can detect virtual walls (provided with the Roomba) and cliffs.

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 $\pm 20^\circ$ cause both sensors to trigger (due to bumper rigidity). However at sampling rates of around 4Hz or less, 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, however, does not suffer as much as the uncorrected odometry.

The virtual wall is difficult to use for MCL and other algorithms as a single wall of many because the uncertainty of hitting the cone of the virtual wall is rather large especially compared to the ground truth of a physical wall bump. Rather than model the virtual wall explicitly, however, the algorithms that follow simply incorporated additional uncertainty into their probabilistic motion models.

Algorithmic Validation: MCL

Monte Carlo Localization (MCL) is a probabilistic estimation of pose in a known map that combines range sensing and odometry. Using only the local sensing of the Roomba we were able to implement and demonstrate successful pose tracking at AAAI 2006. An example of a successful MCL run is shown in Figure 6.

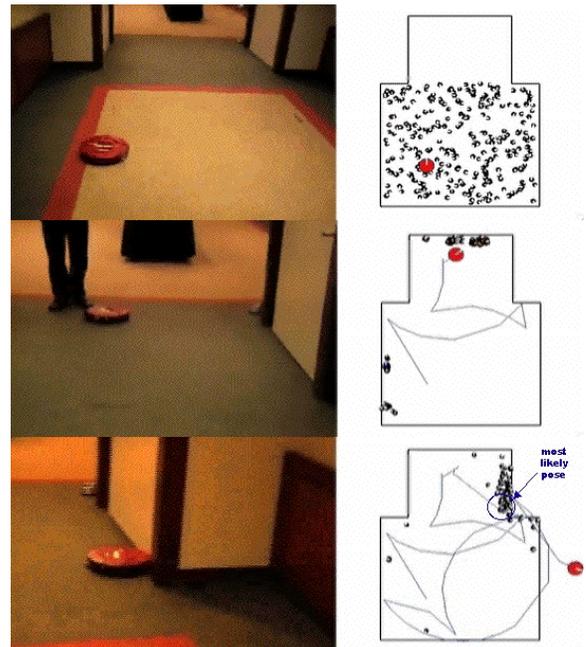


Figure 6: Recorded at AAAI 2006, this shows a Roomba successfully localizing itself using MCL. In the middle image note that the wall in question is actually a virtual wall, encountered as an observer paused to watch the action.

Our video recorded at AAAI and initial runs used the following model of uniform error in motion: 35% in distance

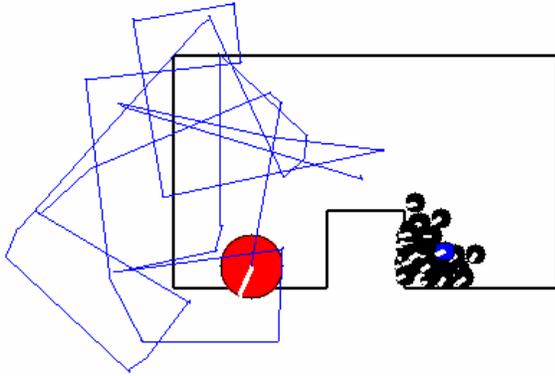


Figure 7: A successful in lab run of MCL where the most probable pose (blue dot) was identical to the ground truth pose.

and 25% in angle which compensated for the inaccuracy of the naive kinematic model. Recall that when driving in large arcs the Roomba's actual motion deviates substantially from its odometric readings. Thus, these large uniform errors were necessary to create a sufficiently robust cloud of particles for MCL. We used 300 particles to successfully localize the Roomba.

Straight-line translation, in general, is more useful than this arc-based motion model both for pedagogical reasons (specifying line segments to travel is a natural starting point for robot programming) and because of the reductions in odometric uncertainty shown in Figure 5. Thus, it was with this improvement to the raw, API-provided driving ability that we sought to implement mapping on the Roomba. We did this in two ways. The first was by adding vision to the system's capabilities through an onboard laptop and an attached webcam. The second approach used only the built-in bump sensor along with strong assumptions about the Roomba's environment.

Adding Vision

Mapping is a logical algorithm to try after localization, but the most popular mapping algorithms today require richer sensing than the Roomba provides. We chose to mount a laptop on the Roomba with velcro to attain onboard processing for vision processing. This setup with Apple's iSight camera can be seen in Figure 8.

Using the iSight, we discovered that the iSight provides pixels in YUV 4:2:2 natively. YUV space is not an absolute color space; it is only a lossy encoding of RGB data. The actual data is 16 bits per pixel providing either a set of UY or VY values for that 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. Thus, to save on computation we reverted to using the raw YUV values. However, for human interaction, e.g., drawing atop the image, YUV/RGB conversion is still re-



Figure 8: Mounting a laptop and the iSight on the Roomba.

quired:

$$Y = ((66R + 129G + 25B + 128) \gg 8) + 16$$

$$U = ((-38R - 74G + 112B + 128) \gg 8) + 128$$

$$V = ((112R - 94G - 18B + 128) \gg 8) + 128$$

Using Mac OS X as this project's primary development environment, we have developed a C library that enables access to the pixel values provided by the iSight camera (or any other QuickTime input). Prior work in this area has not allowed for pixel-level access of values (Heckenberg 2003).

We implemented a set of vision algorithms that 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" will match the teddy bear's clothing. The data on the segmented object is in Table 1. Additional information such as the centroid is not listed. These shape statistics are very useful in identifying objects of the same or similar color. Consider that a cone would have a measured angle near vertical while a red strip on the wall would be horizontal. It is also easy to use the pixel count to eliminate noise by enforcing a minimum size before classifying an object.

The information that this method provides is a good starting point for more complex online algorithms. In many cases this data analysis is sufficient for discerning a small set of known objects.

FastSLAM

We implemented FastSLAM 1.0 with known data correspondence as detailed in *Probabilistic Robotics* (Thrun, Burgard, & Fox 2005). FastSLAM uses a point-feature tracker,

Shape Statistic	Value
ellipse angle	23.1°
major axis	76.5
minor axis	45.5
pixel count	1918
roundness	0.42
color	“red”

Table 1: Shape statistic values for finding the bear.



Figure 9: The largest connected component of teddy bear t-shirt red is colored in blue. The shape statistics are listed in Table 1.

so vision is a natural fit. We created visual landmarks by placing construction paper on the walls as seen in Figure 10. In a joysticked run of the robot, the vision system correctly identified and plotted the uncertainty 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.

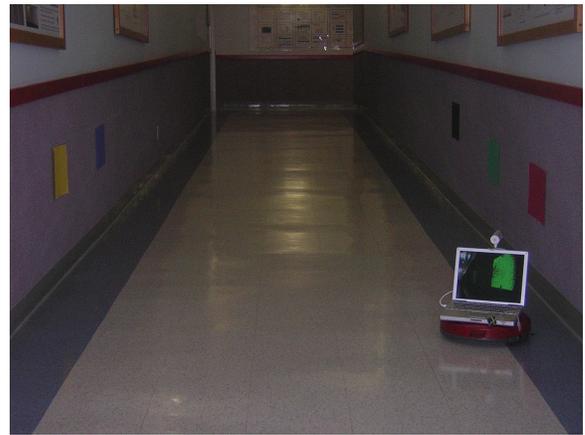


Figure 10: The final position of the Roomba when it closes the loop.

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 others 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.

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**

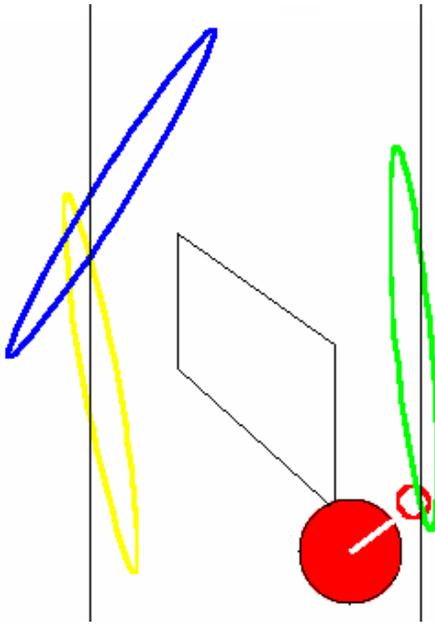


Figure 11: A closed loop run of vision-based FastSLAM.

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

Platform	Cost	Sensing
Lego RCX	\$200	Bmp,Lt
Roomba	\$230	Vis,Mic,Bmp,Enc,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 12: 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;

for handling its idiosyncrasies, 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 College and Chatham College.

References

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

Dodds, Z., and Tribelhorn, B. 2006. Erdos. <http://www.cs.hmc.edu/~dodds/erdos>.

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>.

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.

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

2006. RoombaDevTools. www.roombadevtools.com.

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. Submission to ICRA 2007.