Teaching Robot Localization with the Evolution ER1

Zachary Dodds, Steven Santana, Brandt Erickson,

Kamil Wnuk, Jessica Fisher, Matt Livianu

Department of Computer Science
Harvey Mudd College
Claremont, CA 91711
dodds@cs.hmc.edu

Abstract

Monte Carlo Localization (MCL) is a robust, probabilistic algorithm for estimating a robot's pose within a known map of the environment (Thrun et al., 2001). Now a crucial component of many state-of-the-art robotic systems, MCL's simplicity, flexibility, and power make it a technique particularly suitable for an undergraduate AI or robotics classroom. Several popular low-cost mobile platforms, such as the Lego RCX and the Handyboard, lack the visualization bandwidth and/or the processing power required to effectively experiment with MCL. This paper describes a series of assignments that guide students toward an implementation of MCL on a relatively new, moderately priced platform, Evolution Robotics' ER1. We also report on experiences and lessons learned over the past year of using the ER1 for teaching undergraduate robotics.

Introduction

Since early 2002, Evolution Robotics has offered a small, reconfigurable mobile platform, the ER1, to hobbyists and hardware hardware resellers (Figure 1). At $900, the platform straddles the gap between low-cost, microcontroller-based systems such as the RCX or Handyboard and research-level platforms like the Pioneer or Khepera. As a result, this "mid-level" system holds particular promise for robotics educators: it offers the possibility of providing a laboratory of robotic platforms powerful enough to allow experimentation with a broad range of algorithms but inexpensive enough to support hands-on experience for more than a few students.

We have been using the ER1 since January 2003 at Harvey Mudd College, to support both our course in AI robotics and independent student projects. This paper describes the past year's experiences, including

- Effective laboratory, hardware, and software organization with the ER1.
- A step-by-step set of labs that motivate and lead to student implementations of MCL for the platform.
- Perspective on the "real costs" of an ER1 and the future of low-cost robotic platforms for education.

Overall, our experience with the ER1 has been positive, though not without its challenges. In the following we describe how we have used the platform to help students experiment with algorithms that have only recently begun to appear in undergraduate AI and robotics courses.

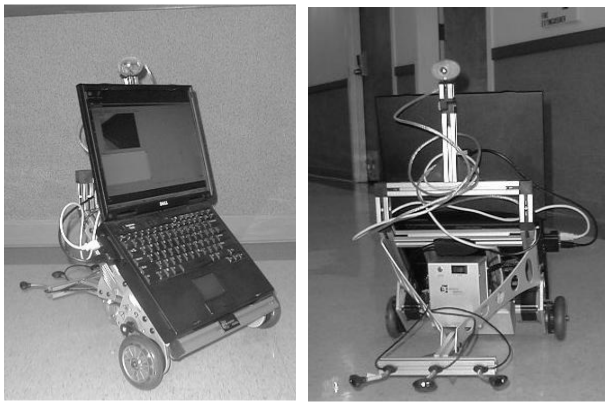


Figure 1: Two views of an Evolution ER1 platform. Perhaps best considered a robotic peripheral, the basic ER1 consists of a laptop with three USB devices: the motors, three IR sensors, and a webcamera.

Philosophy

The ER1 differs from most robotic platforms in that it does not provide any on-board processing. Indeed, the ER1 is perhaps more accurately considered a robotic *peripheral* than a stand-alone system. A user must provide computation in the form of a laptop computer; the drive system and sensors are then attached via USB.

While the modularity of the ER1 is not novel, the impact of using general-purpose computer resources in the form of students' or departments' laptops is substantial. For one, the processing power available is superior to microcontroller-based systems. Indeed, it is superior to that of many existing research-level platforms because it can leverage the steady, cumulative improvements of commercial, off-the-shelf hardware.

While computing power is important, especially to support hands-on experience with probabilistic robotic algorithms, we feel that raw computation is secondary in importance to students' visualization of system data and program code. Watching a system "think" through a familiar GUI offers students much higher work efficiency than reading short LCD displays or interpreting beep sequences. Having that GUI visible via a laptop or wireless connection is standard practice for research platforms, especially when navigating human-scale, rather than table-top, environments.

Yet perhaps the most important impact of the ER1's modularity is on students' views of what constitutes a robotic system. Because the ER1 is simply a set of peripheral devices, its form strongly reinforces the idea that a computer *is* a robot -- one simply needing actuators and sensors to realize that inherent potential. This computational, rather than engineering, focus recommends the ER1 as a hands-on tool for teaching and experimenting with AI algorithms, particularly within computer science departments.

Hardware

The ER1's chassis consists of aluminum x-beams held together with removable joints and connectors, allowing a variety of sensor and motor configurations. The stepper motors power a differential drive platform with hard rubber wheels 10cm in diameter and a caster wheel at the front. This combination provides reasonably accurate odometry in typical indoor environments with low-pile carpet or tiled hallways.

For robotics education, we have found the ER1 shown in Figure 1 most effective. Evolution's default configuration has the controlling laptop *facing* the forward direction of robot motion, i.e., the direction of the camera and IR sensors. This design suits human/robot interaction, as it provides feedback when the user is in the sensors' views. Because our focus is programming for robotic autonomy, we instead point the IR sensors and camera away from the laptop so that the robot interacts with the world. The programmers can then follow, watching the laptop and assessing their system's reasoning.

Sensors

The most important component of any robotic system used to teach AI is its sensors. In addition to proprioceptive odometry and power-level sensors, an ER1 comes with an *irez* webcamera and three "wide-beam" IR range sensors (Figure 2). In our experience the IRs provide reliable range information only for obstacles 25-55 cm away, making them useful for obstacle avoidance, but not an adequate replacement for sonar. Though it does exhibit some lens distortion, the webcamera meets our needs adequately without correction (e.g., see Figure 7). The price/performance information summarized in Figure 3 compares the ER1's sensing capabilities with those of other platforms commonly used for hands-on AI and robotics education. Though it lists for $400, the ER1 has been priced at $900 in this chart to reflect the cost of the recently released software API.

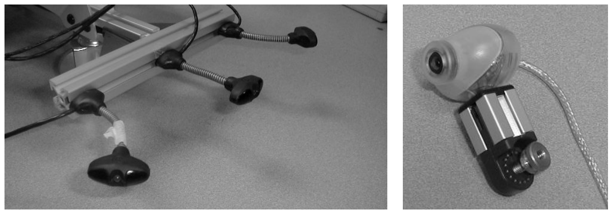


Figure 2: The three elliptical-beam IR sensors, which provide about 30" x 20" of coverage, and the *irez* webcamera that are part of our ER1 configuration.

Model	Price	Sensing
Lego RCX	$200	LT, BMP
Hemisson	$275	LT
Handyboard	$300	LT, BMP, IR
Exp. Handyboard	$450	above + SON, ENC
ER1	$900	IR, CAM, ENC
Amigobot	$1500	SON, ENC
Khepera	$1800	IR, ENC
Cye	$2995	CUR, CAM, ENC
Koala	$4900	CUR, IR, ENC
Magellan	$9550	BMP, SON, ENC

Figure 3: Comparing default configurations of educational robotics platforms. Additional sensors and a variety of upgrades are available for each. **Key**: bump sensing (BMP), light sensing (LT), infrared (IR), motor encoders (ENC), sonar (SON), motor current (CUR), camera (CAM). Computational power and reliability are not charted here.

Overcoming hardware difficulties

Though Evolution recommends a laptop with an 800MHz processor for the ER1, we have used donated computers with 200MHz processors successfully (albeit more slowly) to control the platform. More troubling than the speed, in fact, is the need for power. While the sensors and motors have worked reliably on our ER1s over the past year, the need to connect all of them together has proven a hurdle. Older laptops and many new ones do not have three USB ports. Powered USB hubs tether the robot to the wall; the unpowered hubs we tried would not provide sufficient current to support all of the peripherals or even just the camera and motors together. From Evolution's extensive user community discussion boards (www.evolution.com), we gather that this is not an uncommon difficulty.

To solve this problem, we turned to the PCMCIA card in Figure 4. This $40 off-the-shelf device provides two additional USB ports, allowing the use of the motors, camera, and IR sensors even on our donated laptops, which have only one built-in port. We have found that current-supply and sensor-bandwidth problems can be skirted by (1) ensuring that the motors are turned on (and thus are externally powered) and (2) using the built-in port for the camera, which has higher bandwidth than the PCMCIA ports. These precautions did not suffice when using unpowered hubs.

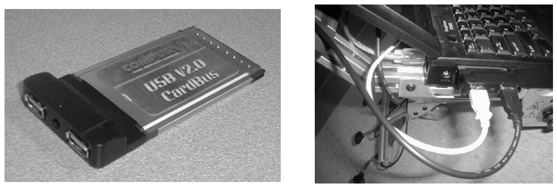


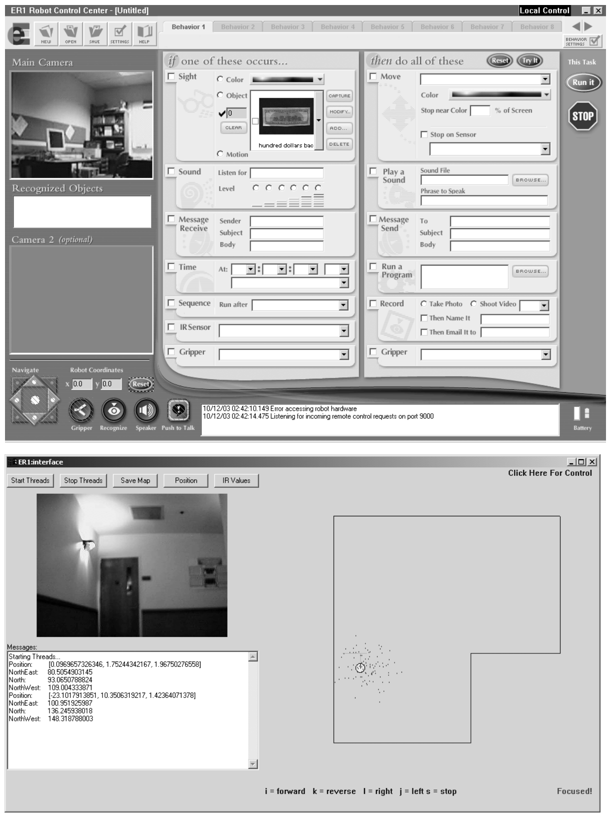
Figure 4: A PC card and USB hub (and the resulting connections), enabling older laptops with only one USB port to control an ER1.

Software

The greatest frustration we encountered was in using the ER1 with the original GUI (Figure 5) and software development kit. This hobbyist interface is too high-level to allow for the implementation of algorithms that require substantial sensor processing. The original SDK for the ER1 supported a python interface, though spotty documentation and only partial access to the underlying hardware led to a frustrating spring '03 semester for a team of students who wrestled with the API, rather than with robotic algorithms.

In May of 2003, Evolution released the Evolution Robotics Software Platform (ERSP), a comprehensive API for Evolution's and other robots. With the beta version, we found that we could quickly and easily access the ER1's sensors and actuators. The software provides layered access to the platform: at the lowest level, the camera, IRs, and the motors may be addressed directly; a middle tier allows behavioral specification of sensor/actuator relationships; the top layer creates tasks from sequential and parallel combinations of these behaviors. The software platform includes a variety of example behaviors and sample code and is very well-documented.

The robot API is accessible through either C++ or python. We chose to develop under python for three reasons: (1) in an educational setting, we feel that efficiency of development outweighs efficiency of execution, (2) the most popular python interpreter is freely available (www.python.org) and has a learning curve less steep than its C++ compiler counterpart, (3) there are a number of python tools, e.g., PyRo, available specifically for teaching robotics (Blank, Meeden, and Kumar 2003). We have not yet integrated PyRo with our software, but it remains an important objective.



```
10/12/03 02:42:10.149 Error accessing robot hardware
10/12/03 02:42:14.475 Listening for incoming remote control requests on port 9000
```

```
Starting Threads...
Position:      [0.0969657326346, 1.75244342167, 1.96750276558]
NorthEast:     80.5054903145
North:         93.0650788824
NorthWest:     109.004333871
Position:      [-23.1017913851, 10.3506319217, 1.42364071378]
NorthEast:     100.951925987
North:         136.245938018
NorthWest:     148.318788003
```

Figure 5: (**Top**) The ER1 default interface, with controls that allow a user to connect sensing to a dictionary of predefined actions. The access to raw sensor values and arbitrary motor commends is quite limited. (**Bottom**) Our GUI, displaying encoder-estimated position. Included are IR values, images, and a 2d map with position, path, and particle estimates superimposed.

A GUI for classroom ER1s

Although there are excellent, freely available robot servers and simulators that run under linux (Gerkey, Vaughan, and Howard, 2003), we have not yet found similar resources available under windows. As a result, we created a small, lightweight GUI (Figure 5) that would help moderate students' interactions with the ER1. Its purpose is to leverage the "visualization bandwidth" that a laptop computer brings to robotic experimentation. Written in python using the wxPython graphics toolkit, it easily interfaces with the ER1 to display raw sensor readings, encoder-based position estimates, the camera's unprocessed or processed image stream, and a global view of the robot's current understanding of its surroundings.

Simple enough to allow robotics students to add or change its functionality, the GUI is also complete enough to allow them to focus on their algorithms, rather than the GUI itself. In addition, it supports a "simulated" mode that allows students to prototype algorithms when away from the robot hardware. In prior semesters most robotics students had to work in the department's lab space. The use of windows, which is ubiquitous on students' desktop or laptop machines, and python, which is freely available, has allowed students to experiment with assignments anywhere they have a computer.

Localization Labs with the ER1

The robotics course at Harvey Mudd emphasizes the computational and AI aspects of the field; as such, it has borrowed from many successful undergraduate curricula (Konolige; Maxwell; Greenwald and Kopena, 2003; Kumar and Meeden, 1998). Our introductory series of course labs motivate and implement Monte Carlo Localization, a readily accessible topic that benefits from the ER1's capabilities and offers a solid launching point for a large swath of active robotics research. Despite the relatively recent release of the ER1 and its API, these labs have been developed and refined from three experiences: a spring '03 course offering with work done on a Handyboard, small groups of students working in summer/fall '03, and a spring '04 course using the ER1. These labs are suitable for student groups of 2-4 students and require approximately 2 weeks each.

Lab 1: Python and Probabilistic Models

To begin, students must develop a working knowledge of the python programming language. In our experience, python is straightforward enough that computer science majors pick it up almost immediately, though they initially resist the language's formatting and indentation requirements. A benefit of python is that nonexperts can also start using the language quickly. As a testament to this accessibility, the GUI of Figure 5, as well as an implementation of MCL, was built during an eight-week summer span by a freshman engineer whose programming experience consisted of a single term of Java-based CS1.

In order to focus their efforts in learning about the ER1's motor and IR systems, each group is asked to create the following probabilistic models, with notation borrowed from (Thrun et al., 2001):

- p(x'|x,a) the robot's new pose (x') probability, given a previous pose (x) and encoder readings (a). This distribution models the ER1's odometric error.

- p(o|x,m) the probability of the robot obtaining an IR range-observation (o), given that the robot's pose is x and the map of the environment is m. This distribution models the noise/bias in the sensors.

To build these models, the students start to collect data by hand and then must write python scripts to automate the process as much as possible, i.e., by commanding batches of robot motions and sensor readings and then finding the mean and standard deviation of the resulting data sets. A particle representation of a Gaussian pose model is shown projected from (x,y,θ) to (x,y)-pose in Figure 6 (Left). For the IR model we have also experimented with triangular distributions, used elsewhere for robot localization and mapping (Thrun, Burgard, and Fox, 1998), because they are simpler than Gaussians and finite in extent.

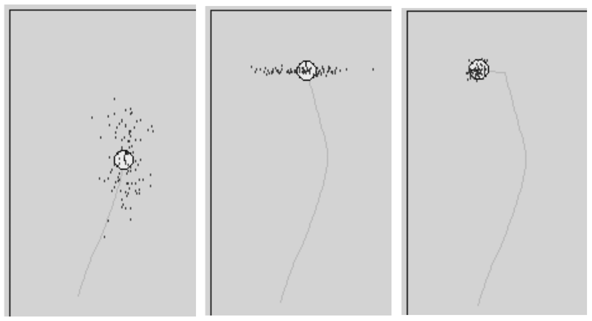


Figure 6: Three snapshots of particle filter tracking and MCL. (**Left**) The robot is not close enough to a wall to obtain meaningful data from the IR sensors, resulting in a large accumulated uncertainty in pose. (**Center**) The IR sensors provide accurate distance-from-wall information. (**Right**) Once the robot has turned to face another wall, the remaining ambiguity is largely resolved.

Lab 2: Particle Filter Tracking

In the second lab, students implement the basic survival-of-the-fittest algorithm of MCL. Rather than globally localizing the robot, the location hypotheses start at the robot's initial position, so that the resulting particle filter tracks the robot's position and estimates its uncertainty. This is a simple form of the Condensation algorithm used for visual tracking (Isard and Blake, 1998):

1. Initialize N particles at the environment's origin, which represents the robot's initial position.
2. At each time step, the poses x = (x,y,θ) of the N particles are probabilistically updated using the encoder readings and the distribution p(x'|x,a) estimated in Lab 1.
3. The probability weight of each particle is updated via Lab 1's p(o|x,m), the latest IR readings, and a provided map of the environment.
4. The N particles are normalized and resampled, and the algorithm loops back to step 2.

As the algorithm runs, the most likely particle or the average of the top few particles may be used as an any-time estimate of the robot's location. The probability-weighted differences from this estimate to all N filter particles, in turn, measure the system's certainty in its location.

Because the geometric computation of the distance from the sensor's position to a wall is so important in step 3, code is provided to students in the form of a python **`Geometry`** class.

In addition, the students are not expected to implement a wandering or wall-avoidance routine. Rather, they set up sequences of motions that will let them test their algorithm under a variety of conditions. An example of the graphical output from such a stylized test is shown in Figure 6.

This lab, in effect, postpones the development of behaviors like wall-following and obstacle avoidance from their usual place early in many robotics curricula. At times, the sensor- and environment-specific parameter tuning required to create robust wall followers or obstacle avoiders has left students with an accurate impression of the difficulty of intelligent environmental interaction, but an inaccurate impression of the field of AI robotics as a whole.

At the end of this lab, students have incorporated Evolution's **`wander`** behavior into their location-tracker. We have found this routine very effective at avoiding obstacles, including moving obstacles, in the IR sensors' field of view. This allows students to experiment early with a truly autonomous system that also explicitly reasons about where it is in the world.

Lab 3: Image Processing and Modeling

The third lab introduces students to image processing using the ER1's USB camera. The students' are guided through the following steps:

- Categorizing image pixels based on RGB thresholds and displaying the results.
- Converting between RGB and HSV color space and categorizing pixels based on HSV values.
- Convolution with directional and isotropic edge-detection kernels
- Line-fitting and outlier detection

We make these common image-processing exercises both concrete and applicable to the students' existing ER1 systems by using two features that happen to distinguish the hallways of our computer science department. As Figure 7 shows, a red piece of molding runs waist-heigh through much of the space. In addition, there are high-contrast lines running down the sides of the hallways formed by the floor tiles. While these features are “natural” for our indoor environment, the flexibility of MCL allows swapping these for almost any architectural features that can be reliably extracted from images.

To reinforce the probabilistic modeling done in Lab 1, this lab also asks students to create Gaussian models of different pixel colors in order to find effective thresholds. Figure 7 illustrates the results of this processing and the resulting recognition of pixels as part of the floor.

Image processing runs on the order of 5s per 320x240 frame on our 800 MHz desktop PC. Python hooks easily into the compiled C++ code we use, with which the resulting image processing well outpaces the 10fps we ask of the camera. Even one image per second noticeably enhances the performance of MCL, as presented in Lab 4.

In this third lab, students continue work on their MCL engine by incorporating heuristics that adapt the number of particles maintained within each generation. In particular, initializing the filter with a large set of uniformly-distributed hypotheses allows global localization of the robot, though perhaps slowly. When most of the robot’s configuration space has been effectively eliminated, the number of particles decreases to a more computationally manageable size.

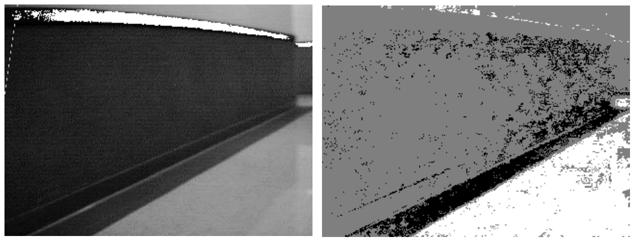


Figure 7: Processed examples from typical hallway scenes. (**Left**) HSV thresholding extracts strips of red molding (shown as white). (**Right**) Pixel classification based on Gaussian models of RGB data. Intensity indicates the probability of membership in the class of floor pixels.

Lab 4 and Possible Extensions

We have prototyped a fourth lab that incorporates the visual information extracted from Lab 3 into the MCL algorithm of Lab 2. To date, we have included the presence or absence of the identifiable red molding into the weights of the particle filters’ hypotheses. To do this, students first annotate the floor-plan map with the the molding’s locations. Then, they create a simple model for the likelihood of correctly detecting (or not detecting) that feature, e.g., 99% confidence of correctly identifying either event. By integrating this model into step 3 of their existing MCL implementation, students observe a collapse of the cloud of possible robot poses along one dimension when a transition in the presence/absence of that visual feature occurs.

There are a large number of possible extensions to the groundwork laid by these labs’ foundation. One team of students has used a learned model of the *image height* of the molding in order to weight poses based on their distance and angle with respect to the wall. Another option is to delve more deeply into the performance of the localizer, e.g., with robust MCL (Thrun et al., 2001) in which potential poses are estimated “backward from the map” as well as forward from previous estimates.

With the computational resources available to a laptop-bearing ER1, the options for an independent follow-up project might include investigating and improving wandering/avoidance behaviors to handle dynamic obstacles in a static map (Thrun et al, 1998), map-building with accurate localization via occupancy grids (Moravec, 1988; Greenwald and Kopena, 2003), and map-building without accurate localization (SLAM) (Thrun, Burgard, and Fox, 1998). All of these would directly extend the spatial-reasoning capabilities of the system. Students in our robotics course also have the option of choosing another robotic platform, e.g., a Handyboard, RCX, or simply by reconfiguring their ER1, if they wish to investigate robot morphology. Even those who choose this path, however, have a foundation in the probabilistic use of data for reasoning about the surrounding environment.

Perspective on the ER1

What is the *real* cost of an ER1?

The $900 price tag cited in Figure 3 both overstates and understates the true cost of an ER1. First, it does not include the cost of the required laptop, which would push the per-platform price closer to $1800. However, a laptop that runs an Evolution is a much more flexible resource than typical processing power provided on board robotic platforms. Further, we have successfully relied on student laptops supplemented by three older, donated laptops in order to run our Evolutions thus far. One advantage of such an approach is that the capabilities of the ER1s will keep pace with the increasing capabilities of commercial computer hardware.

Though the ER1 considered in this paper lists at $400, the second nuance in the $900 price tag is that it accounts for the $5000 cost of the controlling API (averaged over 10 robots). Larger labs would amortize this cost more effectively; smaller labs less so. It is also entirely possible to use the ER1s without this API. However, our experience without it might suggest that laboratories think carefully before trying to do so.

Conclusion

Our efforts over the past year with Evolution's ER1 suggest that it is a promising platform for undergraduate robotics and AI education. The Monte Carlo Localization labs we have designed and tested demonstrate that the ER1 can open up algorithms for student experimentation that would be difficult or impossible to try on less expensive robots. Although the real cost of an ER1 is higher than its $400 price tag, its modularity and use of commercial hardware distinguishes it from other platforms in the $1000 - $2000 range. The ER1 thus represents one of a growing number of acessible, “mid-level” platforms that, we believe, will help redefine the content and practice of robotics and AI education over the next ten years.

References

Blank, D., Meeden, L., and Kumar, D. Python Robotics: An Environment for Exploring Robotics Beyond Legos. In *Proceedings, 34th SIGCSE Symposium on Computer Science Education*, 35(1):317-321, 2003.

Gerkey, B., Vaughan, R. T., and Howard, A. The Player/Stage Project: Tools for Multi-robot and Distributed Sensor Systems. In *Proceedings, Int. Conf. On Advanced Robotics (ICAR)*, pp. 317-323, 2003.

Greenwald, L. and Kopena, J. Mobile Robot Labs. *IEEE Robotics and Automation Magazine*, 10(2):25-32, June 2003.

Isard, M. and Blake, A. Condensation – Conditional Density Propagation for Visual Tracking. *International Journal of Computer Vision*, 29(1):5-28, 1998.

Konolige, K. www.ai.sri.com/~konolige/cs225b/cs225B.html

Kumar, D. and Meeden, L. A Robot Laboratory for Teaching Artificial Intelligence. *Proceedings, 29th SIGCSE Symposium on Computer Science Education*, 30(1), 1998.

Maxwell, B. palantir.swarthmore.edu/~maxwell/classes/e28/s00

Moravec, H. P. Sensor Fusion in Certainty Grids for Mobile Robots. *AI Magazine*, 9(2):61-74, 1988.

Thrun, S., Bucken, A., Burgard, W., Fox, D., Frolinghaus, T., Henning, D., Hofmann, T., Krell, M., and Schmidt, T. Map Learning and High-speed Navigation in Rhino. In Kortenkamp, D., Bonasso, R. P., and Murphy, R., eds., *Artificial Intelligence and Mobile Robots*, pp.21-52. MIT Press, 1998.

Thrun, S., Burgard, W., and Fox, D. A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots. *Machine Learning*, 31:29-53, 1998.

Thrun, S., Fox, D., Burgard, W., and Dellaert, F. Robust Monte Carlo Localization for Mobile Robots. *Artificial Intelligence*, 128(1-2):99-141, 2001.