Teaching Artificial Intelligence with Low-Cost Robots

Lloyd Greenwald and **Donovan Artz**
Department of Computer Science
Drexel University
Philadelphia, PA 19104
{lgreenwa,dono}@cs.drexel.edu

Introduction

While robot platforms have played a role in artificial intelligence and robotics education for over 30 years, the cost and size of these platforms have limited their reach. Recently, low-cost robot platforms have emerged, extending hands-on educational benefits to a diverse audience. In other work (Greenwald & Kopena 2003), we present and discuss the construction and implementation of a course based around a series of detailed lab exercises using these platforms to tackle basic problems in computer science, artificial intelligence, robotics, and engineering. In that work we discuss the overall educational lessons and curricular themes that can be accomplished with these platforms. We observe that in that course, as in many similar courses, the extensive time spent on low-level engineering and computer science leaves little time for artificial intelligence education. In this paper we focus on the use of these platforms to achieve artificial intelligence education goals, assuming as pre-requisites basic engineering and computer science lessons.

We first discuss the tradeoffs an educator must face when deciding to employ low-cost robots in artificial intelligence education, using localization as an example exercise. We then provide step-by-step instructions for using a Handy Board-based mobile robot kit to teach neural networks. We then extend this lesson to teaching Bayesian networks. These example exercises demonstrate that low-cost platforms have matured sufficiently to become a standard tool for teaching artificial intelligence and robotics to advanced undergraduate and beginning graduate students.

Tradeoffs in Education with Low-Cost Robots

Educational goals can be achieved at a wide range of costs. AI modeling, algorithms and applications may be taught with anything from paper-and-pencil exercises, to traditional computer programming, to hands-on robotics programming. The benefits of hands-on robotics have been demonstrated repeatedly (Greenwald & Kopena 2003; Beer, Chiel, & Drushel 1999; Kumar 1998; Kumar & Meeden 1998). However, hands-on robotics benefits come at a variety of costs. An educator choosing a specific robotics platform is limited to the educational exercises possible with that platform. As an example we look at the typical educational goal of teaching *localization*.

Localization is the problem of determining the robot's current position. Methods can be characterized (Borenstein, Everett, & Feng 1996) into either relative or absolute localization methods with the former category including odometry (measuring wheel rotation) and inertial navigation (measuring rate of rotation and acceleration), and the latter category including active beacons, artificial and natural landmark recognition, and map matching. An educator can teach one or more of these localization techniques depending on the platform's capabilities and the students' background preparation in mathematics and algorithms. With respect to the platform the primary choices include sensor variety and cost, and available processing power.

At the lowest cost end, an educator can teach localization with any platform that includes a timer and a way to record motor commands. Simple odometry or dead reckoning can then be used to figure out where a robot has traveled with respect to a known initial pose using simple trigonometry. Although not universal, most low-cost platforms provide enough processing capability to compute trigonometric equations, most likely in real time. Localization taught in this way provides some educational value but is too inaccurate to be built upon in further lessons, such as map building.

At a slightly higher cost is true odometry for localization, using a sensor that measures wheel rotation, for example a break beam IR attached to a Handy Board (Greenwald & Kopena 2003), or an axle rotation sensor with the RCX. These are very cheap sensors but are sufficient for educational purposes. These sensors permit the teaching of kinematics and inverse kinematics and provide localization that can be reliably used for way-point navigation. With additional algorithmic lessons for error correction (Borenstein & Feng 1995), these platforms can then be used to teach map building or at least simple vector field histograms (Borenstein & Koren 1991). Note that adding a wheel rotation sensor to a robot platform permits more advanced algorithms for localization but is not worth the expense if the targeted students are not ready for these algorithms. It is also not worth the expense if the processing power of the target platform can not compute the required equations (floating point math) efficiently (in time and space) or cannot process the rotation measurements rapidly. Another low cost method for localization includes the use of a ground sensor, such as a reflective optosensor, and artificial landmarks such as black tape on a white background.

While these approaches to localization are educational, they are not considered to be part of a modern artificial intelligence curriculum. A suitable educational lesson is to teach a probabilistic method such as Monte Carlo localization (Thrun *et al.* 2001). However, in order to teach such methods the robot platform must be equipped with odometry and a proximity sensor that is accurate enough to predictably model the probability of a sensor reading in a given pose. Low-cost IR-based proximity sensors are not sufficient for this task and higher cost sonar-based proximity sensors are still fairly noisy and difficult to use reliably on platforms such as the Handy Board. The sensor of choice for Monte Carlo localization, the laser range finder, is not yet available with low enough power requirements or cost for low-cost robot platforms. A recent paper (Dodds *et al.* 2004) describes a step-by-step approach to teaching Monte Carlo localization using a camera, a laptop computer, and the odometry built into the Evolution ER1 platform. The educational value of this platform might justify its cost compared to a Handy Board or RCX-based solution, especially if the necessary processing power of the laptop computer can be inexpensively acquired.

Teaching Artificial Intelligence

Artificial intelligence encompasses methods for dealing with uncertain and unknown environments. In (Greenwald & Kopena 2003) we note that the infrared sensors used in our robot building lab class were surprisingly unreliable. Students reported frequent invalid readings in any non-shielded application, making them useless as proximity sensors for obstacle avoidance. We initiated a project to see whether these sensors where actually useless or whether the sensor processing needed to make use of the sensor readings was more sophisticated than that being attempted by the students (and taught in the introductory exercises). The resulting project demonstrated not only that these inexpensive sensors could be used for obstacle detection but that their inherent unreliability provides a practical motivation for teaching advanced artificial intelligence techniques for sensor processing. We describe here how to take advantage of a low-cost robot platform with inexpensive sensors to motivate and teach the artificial intelligence topics of *neural networks* and *Bayesian networks*.

The goal of each exercise presented in this section is to produce a program that takes as input four sensor readings and returns as output a classification of whether or not an obstacle is in the path of the robot. Specifically, the inputs are two infrared sensors, one angled to the left front and the other angled to the right front of the robot, and two photocells (ambient light sensors), one pointing to the left front and one pointing to the right front of the robot; as depicted on the robot in Figure 1. The output is one of four values: no obstacles, obstacle in center, obstacle on right, and obstacle on left. Approaches that didn't work include thresholding and hysteresis with simple calibration of infrared sensors to specific obstacle colors and room light situations as well as manually derived dynamic calibration using the ambient light sensors as filters. We describe three different artificial intelligence methods that may be taught to produce this program (1) neural networks, (2) naive Bayesian networks, and (3) Bayesian networks.

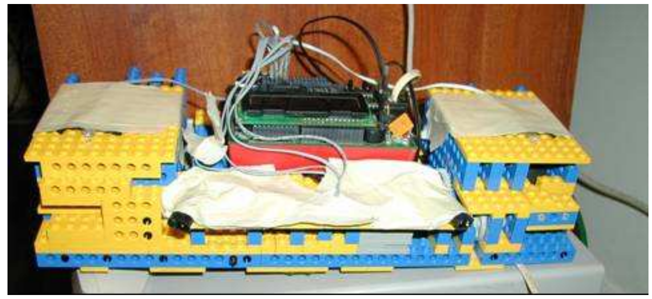

Figure 1: The robot used for the educational exercises in this paper. Notice the Handy Board, 2 forward-facing infrared sensors and 2 forward-facing photocells (and a lot of tape).

A Low-Cost Robot Platform

The educational exercises discussed in this paper are achieved using a mobile robot kit similar to those used elsewhere; centering on the Handy Board (Martin 1999) microcontroller board, LEGO construction pieces, and sensors built with parts from various vendors; and programmed in the Interactive C programming language. A differential drive mobile robot is built that carries its own microcontroller and batteries, and eventually includes paired forward facing infrared and photosensors, a ground facing reflectance sensor, multiple bump switches, wheel encoders for odometry, and a servo-mounted sonar module and photosensor. To help others replicate the educational exercises in this paper, we sketch the major components of our mobile robot kit in Table 1.

A similar kit may be constructed that substitutes the *Mindstorms RCX* and alternative sensors for the Handy Board. The RCX differs from the Handy Board most significantly in that it is limited to three sensor inputs and three motor outputs. Note that the robot used in the exercises in this section does not require the sonar sensor, servo turret, or odometry sensors. This makes the exercises more easily transferable to other low-cost platforms such as the RCX.

Teaching Neural Networks

In this section we provide step-by-step instructions for using our low-cost robot platform to teach neural networks. In the next section we extend this lesson to Bayesian networks. The following steps are described: (1) Building the robot, (2) Gathering experimental data, (3) Designing a neural network, (4) Implementing the neural network, and (5) Analyzing the results.

Step 1: Building the Robot The robot (depicted in Figure 1) is constructed from LEGOs, and it uses a Handy Board as a computation platform and controller for the sensors and motors. The motors each have their own set of

Category	Total Pieces	Details
Handy Board	1	Includes battery, adapter, and cables
Microswitches	4	Bump sensors, digital
Reflective optosensors	3	IR emitter/detector pairs, analog
Break beam optosensors	2	IR, for odometry (analog or digital)
Photocells	3	Light sensors
Range sensor	1	Devantech SRF04 UltraSonic Ranger (or Polaroid 6500)
Motors	4	3 9-volt DC gear motors with cables and 1 servo motor
Wheels	22	Large drive wheels for better odometry; small wheels for casters
Gears	70	40, 24, 16, 8 tooth gears plus others; 6 hole pulleys for odomotry
Plates	300	widths: 6,4,2,1
Bricks	250	lengths: 16,12,10,8,6,4,2
Bushings	180	Full and half bushings
Pins	300	Various types
Axles	84	Various sizes
Misc.	50	Arms, cranks, connectors, flashlight, tape, toolbox with lock

Table 1: Low-Cost Robot Platform Parts List

gears, enabling them to transfer rotational power to their corresponding wheel. The wheels are placed on the left and right of the robot, giving the robot a differential drive. This enables the robot to move forward, backwards, turn left or right, or pivot left or right. The gear ratio from each motor to its wheel is 45 to 1, trading off speed for power. The head of a plastic spoon is used as a front caster; it is glued to LEGO pieces and attached so that the robot can slide stably on a geometric plane.

There are four sensors connected to the robot: 2 infrared receiver/transmitter pairs (IR sensors) and 2 ambient light sensors (light sensors). Each IR sensor has a light sensor associated with it. The light sensor is intended to provide data about the amount of ambient light near the associated IR sensor. The light sensor is placed to avoid the IR sensor's transmitter, while detecting the ambient light being received by the IR sensor's receiver.

The IR sensor transmits infrared light away from the robot. Reflected IR signals are received if an object is sufficiently near the sensor. The color, surface texture, angle, and other factors affect the distance required to register reflected IR signals in the IR sensor's receiver. High amounts of reflected infrared light yield high signal values. If there is little or no reflected infrared, the IR sensor's receiver registers a low signal value.

The sensors are placed approximately in a two-dimensional plane. To differentiate between an obstacle on the robot's left or right, the IR sensors must be placed sufficiently far apart. However, these sensors must not be placed too far apart, or obstacles of small width located directly between the sensors will not be detected. IR sensors have a very short range in which they are effective at detecting obstacles (our sensors operate best at approximately 6 inches from an obstacle, as determined by empirical analyses).

If the robot is moving towards an obstacle, early detection is critical to prevent the robot from colliding with the obstacle. The IR sensors must also be placed such that they will receive reflected IR light in the robot's path as soon as possible. This is achieved by placing the IR sensors along the front edge of the robot (assuming a mostly forward moving robot). The implemented robot has its IR sensors placed 6 inches apart along its leading edge.

The robot's primary movement is forward, and its primary method for changing directions is pivoting. The IR sensors are each angled approximately 20 degrees away from the center. This angling allows the robot to "see" obstacles at the front corners of the robot. The ambient light sensors are placed 10.5 inches apart on flat panels that are elevated above the plane on which the IR sensors sit. The light sensors point straight ahead, and are not shielded (although more accurate information might be obtained by shielding the light sensors from directed light).

Step 2: Gathering Experimental Data Training and validation data is collected from a series of experiments. Each experiment consists of reading samples from the robot's sensors while it is placed in a static environment. The robot remains stationary during each experiment. The data read from the sensors during an experiment is stored internally on the robot, and transferred via a serial line to a desktop computer for processing. The raw data is then processed into a form that can be used for training or validating a neural network.

The objective of each experiment is to collect data from the sensors that represent a specific state of the robot's world. In addition to the presence or absence of an obstacle, there are several other parameters of the robot's world that affect sensor readings. For example, if a bright incandescent light bulb is shining near the robot, the infrared sensors will receive extra infrared light, even if no obstacle is present. In order to generate a robust neural network that can detect obstacles in a wide range of environments, it is necessary to train on data that varies these environmental variables:

- *Obstacle Position* (4 states, primary parameter):

  The presence of an obstacle is described in one of four states: *left*, *right*, *center*, or *none*. *Left* indicates there is an obstacle detected by the left IR sensor, and only this sensor; similarly for *Right*. *Center* indicates there is an obstacle detected by both IR sensors. *None* indicates that neither IR sensor detects an obstacle.

- *Obstacle Surface* (2 states):

  As infrared light reflects differently off of different surfaces, we use objects light in color with two different surfaces: *dull* and *shiny*. We used a dish towel as the *dull* surface, and a plastic coated office binder as the *shiny* surface.

- *Obstacle Distance* (2 states):

  The closer an object is to the sensors, the greater the signal registered by the IR sensors. We test using two obstacle distances: *near* and *far*. In our experiments, *near* is measured as approximately 1 to 2 inches, and *far* is measured as approximately 5 to 6 inches.

- *Ambient Light* (3 states):

  Ambient light significantly affects the signal received by the IR sensors' receivers. If a lot of ambient light is present, the IR sensors will deceptively register high signals. We use three states of ambient light in our experiments: *high*, *medium*, and *low*. *High* light is achieved by using both the overhead room lights with fluorescent light bulbs and a desk light with an incandescent bulb. *Medium* light is achieved by using only the fluorescent overhead lights. *Low* light is achieved by turning off all light in the room and using a flashlight or the light of a computer monitor to conduct the experiment. No sunlight is present in either the experiments or demonstration of the robot.

There 12 possible combinations of states for each of the obstacle positions, *left*, *right*, and *center*, and an additional 3 possible states (ambient light variation) when there is no obstacle; for a total of 39 unique experiments. In each experiment, 1000 samples from each sensor are recorded.

We note that the two different obstacles being used both have relatively flat surfaces which are placed parallel to the robot's front edge in the experiments. The experimental obstacles are not intended to model any particular obstacles, but simply serve to alter the amount of light reflected in each case. The obstacle distance parameter accounts for the varying readings caused by obstacles that are not parallel to the robot's front edge.

Step 3: Designing a Neural Network The inputs to the neural network are:

1. Left Infrared Sensor Pair (*LI*)
2. Right Infrared Sensor Pair (*RI*)
3. Left Ambient Light Sensor (*LA*)
4. Right Ambient Light Sensor (*RA*)

Each sensor is analog, and the conversion to digital yields an integer value in the range $[0, 255]$. Higher values indicate lower sensor readings. For example, if there is very little ambient light, LA and RA should return very high values when sampled. For use in a neural network, each sensor input S_i is normalized to a floating point value in the range of $[0, 1]$: $S_i = S_i/255$.

The outputs from the neural network are:

1. Obstacle on Left (*O1*)
2. Obstacle on Right (*O2*)

O1	**O2**	**state**
0	0	*none*
0	1	*right*
1	0	*left*
1	1	*center*

Table 2: The interpretation of obstacle position from the neural network's rounded outputs.

Each output is a floating point value in the range $[0, 1]$. For interpretation, the outputs are rounded to the closest integer value: 0 or 1. The values of the rounded outputs yield the states described in Table 2.

We use a fully connected, feed forward neural network. Back propagation updating has been used during the training phase. The activation function $a(x)$ of each non-input node is a logistic function: $a(x) = 1/(1 + e^{-x})$.

As mentioned in the previous section, the raw data from the robot is downloaded and processed into a form that can be imported into standard neural network software. Any neural network software may be used on the host computer to design, train and validate the network. In our experiments we use either JavaNNS or SNNS (Zell *et al.* 1992).

We divide the experimental data into a training set of 900 samples and a validation set of 100 samples, per experiment. The validation set is used to avoid overfitting the network to the training data. The experiments with no obstacle in the robot's sensor range were repeated 13 times in the training and validation set to give them an equal weight with the experiments containing an obstacle. This results in 78000 patterns for use in training and validation.

The number of hidden layers in the neural network and the number of neurons in each hidden layer are determined by trial and error in our exercise. More sophisticated network design methods may also be taught. The students might first attempt to solve the problem with a single layer perceptron network and then re-try the exercise with hidden layers, testing whether or not this classification task is linearly separable. In our tests, a perceptron network was not as effective as one with two hidden layers.

The best network we could design is depicted in Figure 2.i. This neural network consists of two hidden layers: the first with 16 nodes, and the second with six nodes. Mean squared error for both the training and validation set went to zero in less than 50 training cycles.

Step 4: Implementing the Neural Network In order to be incorporated into a robot control program, once a neural network is designed, trained, and validated on the host PC it must be converted into code that runs on the robot. A valuable feature of SNNS is its ability to automatically generate C code, through *snns2c* (a tool included with SNNS distributions), to implement a neural network. However, the C code generated will not compile for Interactive C (the primary language used to program Handy Board-based robots). This is due to the large memory usage and the linked list type data structures used in the code generated by *snns2c*. A source of difficulty is that the Handy Board is not able to store and

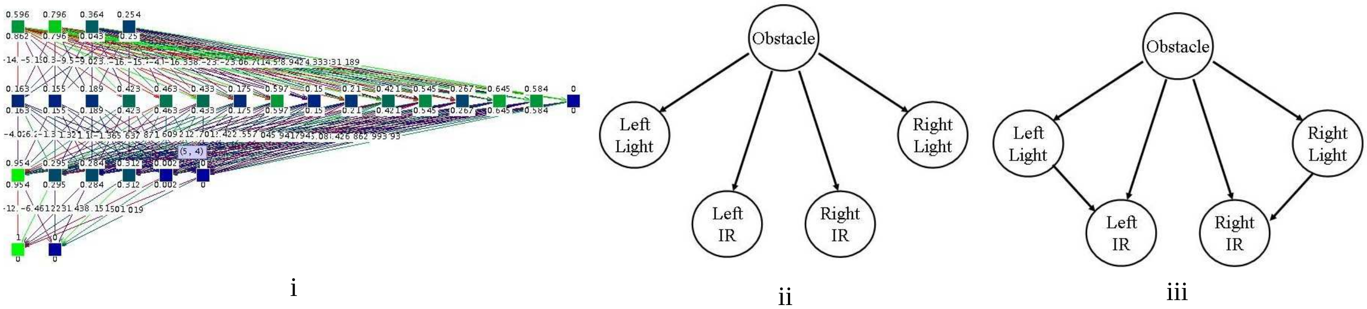

Figure 2: (i) A neural network designed to demonstrate robust obstacle detection and classification using low-cost infrared sensors. The network classifies obstacle location (none, center, left, right) given left and right infrared measurements and left and right ambient light measurements. (ii) A naive Bayesian network for the same classification task. (iii) A Bayesian network for the same classification task, removing the assumption of conditional independence of sensors given the cause (obstacle).

work with large stores of floating point numbers. We developed software that transforms the automatically generated C code into legal Interactive C code. Our neural network conversion software is limited to feed-forward neural networks using the logistic activation function, as described in this paper. This software may be used on any SNNS supported platform. Note that since SNNS generates C code this exercise can be ported to many existing low-cost robot platforms.

To test whether or not the implemented neural network provides useful classifications of obstacles from sensor data, we downloaded the resulting Interactive C code to the robot and incorporated it into a robot control program. The neural network code itself is contained in a separate file and does not contain any Handy Board specific instructions. We first successfully tested the code in the same static environments as used for gathering experimental data, using the LCD display to show the classification results in real-time. We then tested the code by programming the robot to move in a straight line until it encounters an obstacle (as determined by the neural network code). If an obstacle is encountered on the left, the robot backs up and pivots right to avoid the obstacle. If an obstacle is encountered on the right, the robot backs up and pivots left to avoid the obstacle. If an obstacle is encountered in the center, the robot backs up to make some turning room and pivots 180 degrees (turns around).

Step 5: Analyzing the Results Test runs of the resulting neural network control code were successfully performed in a variety of ad hoc, indoor obstacle courses. The trained neural network is effective at detecting objects in most tested conditions. Both moving objects (hands or feet), and static objects (books, bags, boxes, and walls) are detected as expected. While we did not detect any decrease in classification accuracy with robot movement, faster moving robots had more difficulty reacting to a detected obstacle. The robot used in this exercise is relatively slow (due to the high gear ratio), and thus does not appear to be affected by potential sensor illusions caused by movement. An interesting extension to this exercise would be to include robot speed as a input to the neural network and gather data from dynamic runs of the robot.

We empirically determined that the IR sensors used with the robot are not capable of detecting dark obstacles. Neither a black-shiny obstacle (a black, plastic office binder) nor a black-dull obstacle (a black laptop bag) caused the sensors to register values even slightly different from the those readings taken when no obstacle is present. Thus, dark obstacles were eliminated from the experiment. In real-world environments including dark obstacles, the robot will need a different method for detecting dark obstacles. Obstacles that are closer to dark than light in color simply take longer to register as an obstacle, causing the robot to move closer to the obstacle than expected.

The complete set of experimental data files (the training and validation sets) and the supporting scripts and software are available at `http://www.cs.hmc.edu/roboteducation`.

Teaching Bayesian Networks

It is instructive for students to attempt the same classification task using Bayesian networks. Bayesian networks may be taught using a similar set of step-by-step instructions to that of our neural network exercise. Steps 1 and 2 (building the robot and gathering data) are identical. In fact we exploit this fact by using the data generated in our neural network exercise to teach Bayesian networks in a non-robotics course. As mentioned, this data is freely available for others to replicate these exercises.

Our introduction to Bayesian networks lessons focus on representations and inference algorithms that assume discrete conditional probability tables. To adapt the neural network data for these exercises we had to first discretize the continuous data. We experimentally determined that using four signal ranges per sensor variable was inadequate and using 20 bins per variable led to huge tables. In the following exercises we discretize each sensor signal into 10 uniformly sized bins. Teaching Bayesian network representation and inference methods using continuous valued variables would avoid this step.

Our Bayesian network exercises consist of the following abbreviated steps:

1. Build robot (see previous section)

2. Gather experimental data (see previous section)
3. Use the training data to build the full joint probability distribution table over all atomic events. While it is possible to build the conditional probability tables directly from the data, we found it instructive to have the students build the full joint table and then implement marginalization and normalization functions to obtain conditional distributions from the joint.
4. Derive (from the joint) the conditional probability tables needed to complete the naive Bayesian network depicted in Figure 2.ii. The students first try to solve the classification task using naive Bayesian networks before introducing more complex variable relationships.
5. Implement the naive Bayesian classification computation.
6. Evaluate the classification accuracy of the naive Bayesian network on the validation set. Note that any fraction of the original data may be set aside to provide separate training and testing sets. For simplicity we used the original training set (90% of the data) for training and the validation set (10% of the data) for testing.
7. Derive (from the joint) the conditional probability tables needed to complete the Bayesian network depicted in Figure 2.iii. Note that this network removes some of the conditional independence assumptions of the naive Bayesian network and permits the students to evaluate any increased classification accuracy due to the richer representation.
8. Derive an equation to compute the maximum a posteriori query for the obstacle variable given the sensor variables. Implement this equation.
9. Evaluate the classification accuracy of the Bayesian network on the validation set. Compare this classification accuracy to that of the naive Bayesian network.
10. Implement stochastic sampling using likelihood weighting to perform other (non-classification) queries on the Bayesian network.

Comparing the naive Bayesian network of Figure 2.ii and the Bayesian network of Figure 2.iii the students learn that the Bayesian network captures additional influences among variables compared to the naive Bayesian network. Theses influences lead to better classification accuracy. Additionally, modeling the data with a Bayesian network permits the study of different inference algorithms with more varied queries (other than classification). Although we employed this exercise in a class without actual robots the use of data from real robot experiments made the task more interesting to the students.

In addition to teaching the students about different representations for uncertainty, and differing inference algorithms for classification, students can compare the classification accuracy of the resulting Bayesian networks to the neural network of the previous section. Other comparisons between the two representation methods can be discussed such as the "readability" of the resulting networks, continuous versus discretized variables, and the ability to simultaneously capture both classification output and degree of belief in that output with the Bayesian networks. A perceptron network could also be trained from the data to provide a comparison to the resulting neural network as well as the naive Bayesian network.

Acknowledgements

We thank Yogi Mehta and Brian Summers for their efforts in testing the Bayesian networks exercises. This research is sponsored in part by a National Science Foundation (NSF) Instrumentation Award under grant CISE-9986105.

References

Beer, R. D.; Chiel, H. J.; and Drushel, R. F. 1999. Using autonomous robotics to teach science and engineering. *Communications of the ACM* 42(6).

Borenstein, J., and Feng, L. 1995. Correction of systematic odometry errors in mobile robots. In *Proceedings of the 1995 International Conference on Intelligent Robots and Systems (IROS '95)*.

Borenstein, J., and Koren, Y. 1991. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation* 7(3):278–288.

Borenstein, J.; Everett, H. R.; and Feng, L. 1996. *Where am I? Sensors and Methods for Mobile Robot Positioning*. The University of Michigan.

Dodds, Z.; Santana, S.; Erickson, B.; Wnuk, K.; and Fischer, J. 2004. Teaching robot localization with the Evolution ER1. In Greenwald, L.; Dodds, Z.; Howard, A.; Tejada, S.; and Weinberg, J., eds., *Accessible Hands-on Artificial Intelligence and Robotics Education*. American Association for Artificial Intelligence Press.

Greenwald, L., and Kopena, J. 2003. Mobile robot labs: On achieving educational and research goals with small, low-cost robot platforms. *IEEE Robotics and Automation, Special Issue on Robotics in Education: An Integrated Systems Approach to Teaching* 10(2):25–32.

Kumar, D., and Meeden, L. 1998. A robot laboratory for teaching artificial intelligence. In *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education (SIGCSE-98)*, volume 30(1) of *SIGCSE Bulletin*, 341–344. New York: ACM Press.

Kumar, D. 1998. Curriculum descant, teaching about embedded agents. *SIGART Bulleting* 9(1):7.

Martin, F. G. 1999. The Handy Board technical reference. Technical report, MIT.

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

Zell, A.; Mache, N.; Huebner, R.; Schmalzl, M.; Sommer, T.; and Korb, T. 1992. SNNS: Stuttgart neural network simulator. Technical report, Stuttgart.