

Using Educational Robotics to Motivate Complete AI Solutions

*Lloyd Greenwald, Donovan Artz, Yogi Mehta,
and Babak Shirmohammadi*

■ Robotics is a remarkable domain that may be successfully employed in the classroom both to motivate students to tackle hard AI topics and to provide students experience applying AI representations and algorithms to real-world problems. This article uses two example robotics problems to illustrate these themes. We show how the robot obstacle-detection problem can motivate learning neural networks and Bayesian networks. We also show how the robot-localization problem can motivate learning how to build complete solutions based on particle filtering. Since these lessons can be replicated on many low-cost robot platforms they are accessible to a broad population of AI students. We hope that by outlining our educational exercises and providing pointers to additional resources we can help reduce the effort expended by other educators. We believe that expanding hands-on active learning to additional AI classrooms provides value both to the students and to the future of the field itself.

Artificial intelligence has a number of wonderful domains that help motivate undergraduate students to enroll in AI courses and work hard to grasp advanced AI representations and algorithms. One particularly compelling domain is robotics. Robotics combines the fantasy of science fiction with practical real-world applications and engages both the imaginative and sensible sides of students. Examples of robotics-inspired AI span from early work on AI planning with Shakey

(Nilsson1984) to more recent work on multi-agent systems and machine learning in RoboCup (Stone 2000).

In addition to providing inspiration, exploring artificial intelligence representations and algorithms using robotics helps students to learn complete solutions. A complete solution is one in which a student considers all the details of implementing AI algorithms in a real-world environment. These details range from system design, to algorithm selection and implementation, to behavior analysis and experimentation, to making the solution robust in the face of uncertainty. In our classes we find that robotics problems encourage students to investigate how AI algorithms interact with each other, with non-AI solutions, and with a real-world environment. Students investigate how to convert sensor data into internal data structures, how to weigh the costs and benefits of physical exploration, whether or not to use offline simulation and tools, and how to deal with the severe resource limitations and time constraints of embedded computation. Despite the added costs of building complete solutions, experience with real-world environments helps ground lessons and stimulates thinking about new challenges and solutions.

Although robotics has inspired research on AI representations and algorithms, until recently the cost and size of these platforms has made it difficult to convey the same line of in-

investigation to undergraduate students. Low-cost robot platforms, as surveyed earlier in this issue (Dodds et al. 2006), have the potential to mitigate this cost barrier. Investigators have begun to report successful demonstrations of hands-on robotics using low-cost platforms in AI courses (Greenwald and Kopena 2003; Mayer, Weinberg, and Yu 2004, Dodds et al. 2004). These reports, as well as the instructional material on advanced AI topics in this special issue, provide critical supplements to existing texts and curricular material on general undergraduate artificial intelligence, engineering, and computer science topics.

Even at low cost (\$300–\$600 per robot kit), it is possible to teach advanced AI with the hardware and software resources of these platforms. Low-cost platforms provide varying levels of mobility, sensors, development environments, and processing power. Lego-based platforms like the RCX (Klassner 2002), or Handy Board (Martin 1999) are designed for flexible experimentation with mobility. Development environments include many standard programming languages like Java, C, Pascal, Forth, and ADA. Even Lisp is available, as detailed elsewhere in this issue of *AI Magazine* (Klassner 2006). Commonly available sensors include photoresistors, infrared, touch, wheel encoders (for odometry), and, most impressively, sonar. With respect to implementing advanced AI algorithms, the processing power of low-cost platforms can be problematic. At the low end, we get from 2 megahertz to 16 megahertz of central processing unit (CPU) speed and around 32 kilobytes of random-access memory (RAM). The education lessons discussed in this article have been carefully designed to work successfully with such constrained resources. We have found that these resource constraints often provide additional educational opportunities to investigate AI topics like resource-bounded reasoning. These platforms additionally provide communications protocols so that off-board processing can be included in complete AI solutions.

We have used low-cost robot platforms in the classroom to teach many AI topics including heuristic search, planning, probabilistic planning, representing uncertainty, and machine learning. In this article, we outline our classroom-tested material for teaching Bayesian networks, neural networks, and particle filtering. This article is organized around the two educational themes just introduced: (1) *robotics problems motivate AI solutions*, and (2) *robotics problems encourage complete AI solutions*. The first theme is presented in the context of solving a robot obstacle-detection problem using

neural networks or Bayesian networks and implementing the solution on a Handy Board-based robot. The second theme is presented in the context of solving a robot-localization problem using particle filtering and implementing the solution on an RCX-based robot.

The projects described in this article were designed primarily as part of a stand-alone robotics course for computer science majors. The hands-on topics covered in this course include robot structure, drive trains, sensors (infrared, photo, bump, active sonar), control (open loop, closed loop), simple behaviors (obstacle detection, light following, obstacle avoidance), reactive control, multiple behavior coordination (algorithmic, priority-based, subsumption), odometry using wheel encoders, forward kinematics and localization, inverse kinematics, way-point navigation, simple feature detection using sonar, probabilistic localization, local and global map building, sensor fusion, vector field histograms, and simple path planning. For interested students this course provides hands-on experience that complements material taught in artificial intelligence courses for advanced undergraduate and early graduate students. The projects in this article have also been used in workshops, including a recent NSF-sponsored workshop for teachers learning to use the RCX in education. Please refer to Greenwald and Kopena (2003) and Greenwald and Artz (2004) for further information on the structure and labs of this course.

Obstacle Detection with Neural Networks, Bayesian Networks, and the Handy Board

A typical task in a robotics class is to program a robot to detect and avoid obstacles. In our robot-building lab course (Greenwald and Kopena 2003) we initially assigned this as a non-AI exercise in processing noisy sensor readings. We chose to use inexpensive and unreliable infrared sensors as proximity detectors for this exercise. In lectures we taught students standard methods for processing sensor data, including ambient light correction, surface reflectance calibration, thresholding, averaging, and hysteresis. The students were later asked to use these processing methods to classify sensor data into one of four obstacle position states: left, right, center, or none.

This seemingly simple exercise repeatedly led to frustration and failure. Students reported frequent invalid infrared readings in any non-shielded application, indicating that these sensors were useless as proximity detectors for ob-

stacle detection and avoidance. The following exercise helped us determine whether the source of the problem was the sensors or the programmers, and at the same time uncovered our first educational theme: *robotics problems motivate AI solutions*.

We initiated a project to see whether or not these sensors were actually useless. The resulting project demonstrated not only that these inexpensive sensors could be used for obstacle detection but also 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*.

We focus this section on step-by-step instructions for using a low-cost robot platform to teach neural networks (including perceptrons). We then describe how to extend and modify this lesson to teach Bayesian networks (including naive 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. For added educational impact, this lesson can be preceded by non-AI approaches.

Step One: 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. To help others replicate the educational exercises in this article we sketch the major components of our mobile robot kit in Greenwald and Artz (2004). The motors each have their own set of 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: two infrared (IR) receiver/transmitter pairs (IR sensors) and two 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 trans-

mitter 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 cannot 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 six inches from an obstacle, as we determined empirically).

If the robot is moving towards an obstacle, early detection is critical to prevent the robot from colliding with it. 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 six 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).

Step Two: Gathering Experimental Data

Training and validation data are 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 are stored internally on the robot and transferred over a serial line to a desktop computer for processing. The raw data are then processed into a form that can be used for train-

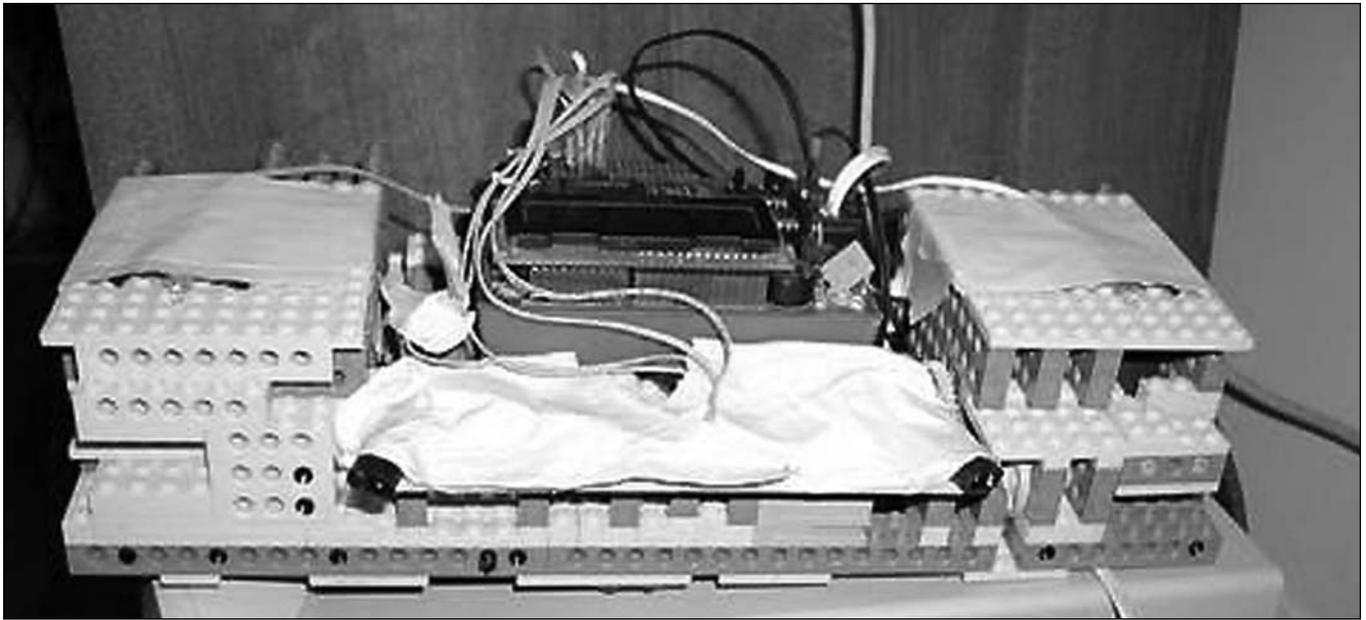


Figure 1. A Robot Used to Teach Neural Networks and Bayesian Networks.

Notice the Handy Board, two forward-facing infrared sensors, and two forward-facing photocells (and a lot of tape).

ing or validating a neural network. The primary programming environment for the Handy Board, Interactive-C, has several methods for transferring data from the robot to a desktop computer, including an "upload array" tool.

The objective of each experiment is to collect from the sensors data 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 lightbulb 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 vary these environmental variables:

Obstacle Position (four 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 on the left of the robot (and should be detected by the left IR sensor, and only this sensor); similarly for *right*. *Center* indicates there is an obstacle in front of the robot (and should be detected by both IR sensors). *None* indicates that no obstacle is present.

Obstacle Surface (two 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 (two states): The closer an object is to the sensors, the greater the signal registered by the IR sensors. We test this using two obstacle distances: *near* and *far*. In our experiments, *near* is measured as approximately one to two inches, and *far* is measured as approximately five to six inches.

Ambient Light (three 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 are thus 12 possible combinations of states for each of the obstacle positions, *left*, *right*, and *center*, and an additional three 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.

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

Table 1. The Interpretation of Obstacle Position from the Neural Network's Rounded Outputs.

We note that the two obstacles in our experiments have relatively flat surfaces that are placed parallel to the robot's front edge. The experimental obstacles are not intended to model any particular obstacles, but simply serve to alter the amount of light reflected in each case.

Step Three: Designing a Neural Network

The inputs to the neural network are: left infrared sensor pair (LI), right infrared sensor pair (RI), left ambient light sensor (LA), and right ambient light sensor (RA). Each sensor is analog, and the Handy Board 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 obstacle on left (O1) and obstacle on right (O2). 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 1.

We experiment with fully connected, feed-forward neural networks with varying number of hidden layers, trained using back propagation.

The activation function $a(x)$ of each noninput node is a logistic function: $a(x) = 1/(1 + e^{-x})$.

As mentioned in the previous section, the raw data from the robot are dumped to a host computer 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 additional times in the training and validation set to give them an equal weight with the experiments containing an obstacle.

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 retry the exercise with hidden layers, testing whether or not this classification task is linearly separable. In our tests (described later on), a perceptron network was not as effective as one with two hidden layers.

Figure 2a depicts our most successful network. This neural network consists of two hidden layers, the first with 16 nodes and the second with 6 nodes. Mean squared error for both the training and validation set went to zero in fewer than 50 training cycles.

Step Four: Implementing the Neural Network

Once a neural network is designed, trained, and validated on the host computer it must be converted into code that runs on the robot. A valuable feature of SNNS is its ability to automatically generate C code to implement a neural network, through *snms2c* (a tool included with SNNS distributions). However, the C code generated did not compile for the version of Interactive C (the primary language used to program Handy Board-based robots) that we use in the classroom. This is due to the large memory usage and the linked list type data structures used in the code generated by *snms2c*. Another source of difficulty is that the Handy Board is not able to store and work with large stores of floating-point numbers (though these problems may be alleviated as Interactive C continues to improve). We developed software that converts 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 article. 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 pro-

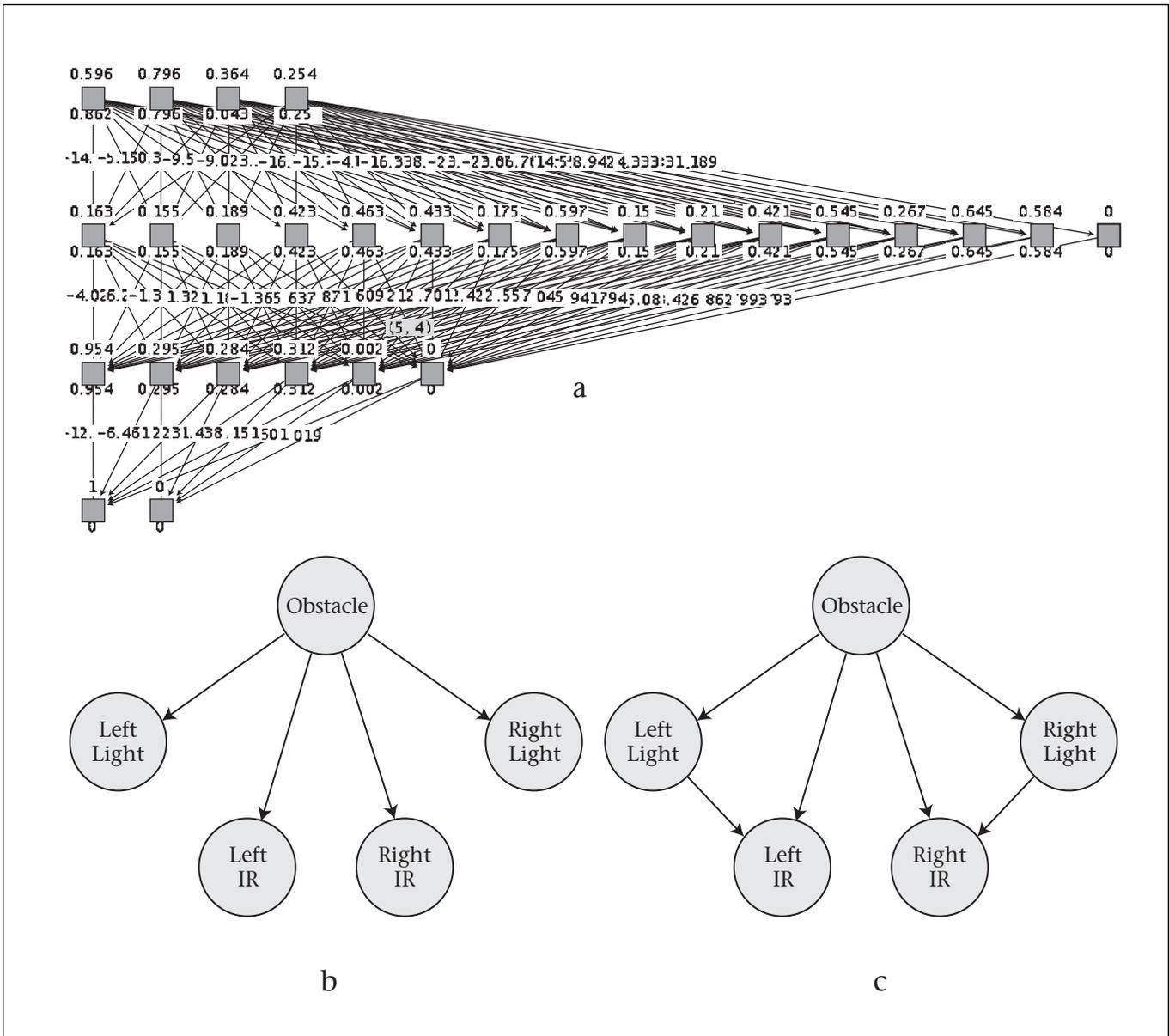


Figure 2. Demonstrating Robust Obstacle Detection and Classification Using Low-Cost Infrared Sensors.

(a) 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. (b) A naive Bayesian network for the same classification task. (c) A Bayesian network for the same classification task, removing the assumption of conditional independence of sensors given the cause (obstacle).

gram. 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 Five: 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. Although we did not detect any decrease in classification accuracy with robot movement, as expected, 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 an 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 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.

Teaching Bayesian Networks

We have also successfully employed the robot obstacle-detection problem to motivate students to learn Bayesian networks. Bayesian networks may be taught as a solution to this classification task by 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 nonrobotics course. As specified elsewhere in this article, this data is freely available for others to replicate these exercises.

In our undergraduate AI course we focus on Bayesian networks with discrete-valued variables and 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 4 bins per sensor variable was inadequate and using 20 bins per variable led to huge,

computationally unmanageable 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 ten abbreviated steps:

Step 1: Build robot (see previous section).

Step 2: Gather experimental data (see previous section).

Step 3: Use the training data to build the full joint probability distribution table over all atomic events. Although 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.

Step 4: Derive (from the joint) the conditional probability tables needed to complete the naive Bayesian network depicted in figure 2b. The students first try to solve the classification task using naive Bayesian networks before introducing more complex variable relationships.

Step 5: Implement the naive Bayesian classification computation.

Step 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 percent of the data) for training and the validation set (10 percent of the data) for testing.

Step 7: Derive (from the joint) the conditional probability tables needed to complete the Bayesian network depicted in figure 2c. 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.

Step 8: Derive an equation to compute the maximum a posteriori query for the obstacle variable given the sensor variables. Implement this equation.

Step 9: Evaluate the classification accuracy of the Bayesian network on the validation set. Compare this classification accuracy to that of the naive Bayesian network.

Step 10: Implement stochastic sampling using likelihood weighting to perform other (nonclassification) queries on the Bayesian network.

Comparing the naive Bayesian network of figure 2b and the Bayesian network of figure 2c, the students learn that the Bayesian network

Representation	Correct (out of 7500)	Accuracy (classification percentage)
Naive Bayes	5999	79.99
Bayes	6365	84.86
Perceptron	6200	82.66
NN (1 hidden/4)	6200	82.66
NN (1 hidden/8)	6400	85.33
NN (2/4,8)	6400	85.33

Table 2. Classification Accuracy on Discretized Data.

captures additional influences among variables compared to the naive Bayesian network. These 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 learning about different representations for uncertainty, and differing inference algorithms for classification, students can compare the classification accuracy of the resulting Bayesian and neural networks. In our experiments with continuous data, the perceptron network provided a 94.66 percent accuracy while all variants of the multilayer neural network achieved 100 percent classification accuracy. Our experiments with discretized data show that the Bayesian network and a neural network with at least one layer of eight hidden nodes perform the best. Naive Bayesian networks and perceptron networks are slightly less accurate. Example results using discretized data are summarized in table 2. Note that discretization leads to many redundant training examples for the neural network experiments. This redundancy caused only a fraction of the testing file to contain distinct values.

Further comparisons between Bayesian and neural networks can provide student insight into the human readability of the resulting networks, the trade-offs between continuous and discretized variables, and the ability to simultaneously capture both classification output and the degree of belief in that output with Bayesian networks. These educational lessons demonstrate that the real-world problem of ro-

bot obstacle detection provides an effective method to motivate and teach AI solutions.

Localization with Particle Filtering and the RCX

Localization is also a great problem to motivate AI solutions. Localization is the problem of determining a robot's position within a local or global reference frame (for example, a map). Contrasted to the obstacle-detection exercise described above, localization is considered a more challenging problem in need of intelligent solutions. Localization solutions of varying complexity and real-world applicability exist and can be characterized (Borenstein, Everett, and Feng 1996) into either relative or absolute localization methods. The former category includes odometry (measuring wheel rotation) and inertial navigation (measuring rate of rotation and acceleration), and the latter category includes 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.

At the lowest-cost end, an educator can teach localization with any platform that includes a timer and a way to record motor commands. Odometry or dead reckoning can then be used to figure out where a robot has traveled with respect to a known initial pose using trigonometry. However, this solution leads to fairly inaccurate localization. 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



Figure 4. Real-Time Belief-State Visualization Tool Showing Initial Localization Particles.

the RCX communications tower. The visualization program runs on the PC and provides a view of the belief state in real-time, as depicted in figure 4.

Step One: Building the Robot

The main components of the robot needed for this exercise are a drive train for moving the robot parallel to the wall, axle sensors to read wheel rotations for use in odometry estimates, and a sonar attachment fixed to point toward the wall. There are many sources of design options for single and differential drive robots using Lego pieces, including the attachment and programming of axle sensors using LEJOS (Bagnall 2002). LEJOS is the Java development environment for the RCX that we use in this exercise.

There are several sonar sensors designed and sold for the RCX (Dodds et al. 2006). We selected a low-cost sensor with a range of 30–150 centimeters and an accuracy of 4–5 centimeters. This sensor becomes difficult to use if the robot is permitted to move too close to or too far away from the wall. The sensor is also problematic if readings are taken when the sensor is oriented more than a few degrees away from perpendicular to the wall. For these reasons, students are encouraged to engineer either a hardware or software solution so that the robot moves as close as possible to parallel to the

wall, at a constant distance away from the wall.

A hardware solution for parallel movement is to carefully design a single drive-train robot base. This is a difficult task to engineer with Lego pieces. A combination hardware and software solution is to build a differential drive robot and teach the students to program a wall-following routine. A wall-following routine uses either the sonar sensor or additional proximity sensors and a closed-loop control program to adjust robot orientation to maintain a fairly consistent distance from the wall.

In addition to building and programming a robot that moves parallel to the wall, students design their robot base so that they can get as much useful information from wheel-rotation sensors as possible. Slippage and friction cause wheel-rotation measurements to yield poor odometry estimates. Students address these problems by observing good design principles such as building larger wheelbases to avoid excess orientation errors and building castor wheels that do not bear significant weight or induce uneven frictional forces. Students also learn to limit robot speed while making orientation adjustments and to limit accelerations in movement. Although the purpose of this exercise is to understand Monte Carlo localization, students learn that implementing an AI solution in the real world includes paying careful attention to the non-AI components of the complete solution as well. AI educators and students can take advantage of the resources discussed throughout this special issue and select a desirable level of time and effort to apply toward learning complete solutions. These resources include tools for building and programming low-cost platforms, simulation alternatives, and robot programming abstractions.

Step Two: Building the Models

The robot-localization problem is a state-tracking problem in which the values of state variables are updated over time in response to movement actions and sensor feedback. The state variables track robot location and orientation. In a global frame of reference on a two-dimensional plane the variables are x -position, y -position, and orientation μ . In probabilistic localization we maintain probability distributions over these three variables, also called the *belief state*.

Localization state variables cannot be observed directly in this exercise. They are hidden variables whose values at a given time can be estimated from the observed variables, namely sensor readings and actions. The mappings from values of observed variables to those of hidden localization variables are noisy due to

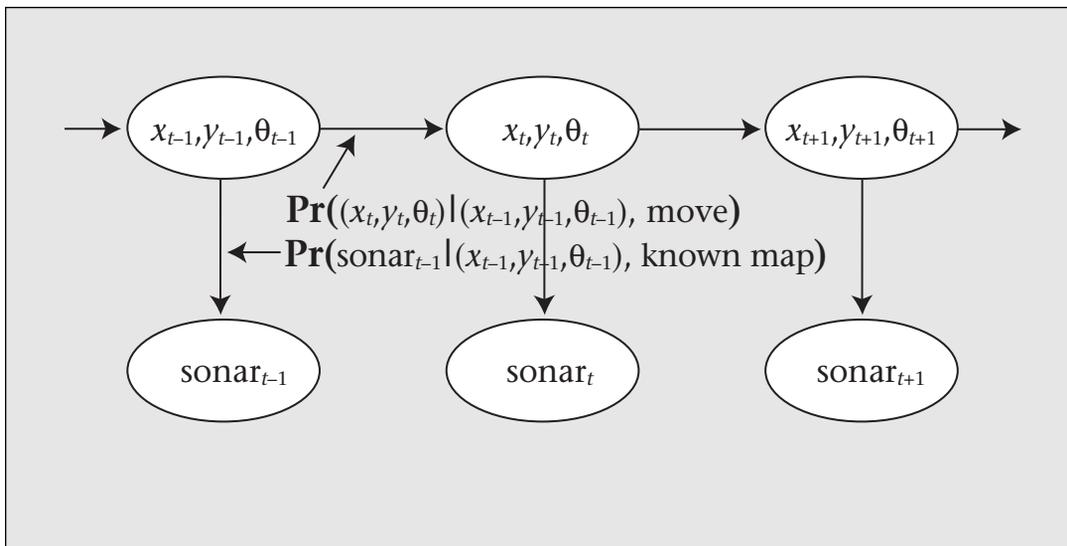


Figure 5. A Dynamic Bayesian Model for the Simplified Robot-Localization Problem.

sensing and movement noise. Modeling the relationship between observed and hidden variables can be done using a hidden Markov model or dynamic Bayesian network, as depicted in figure 5. This figure differentiates the observable sonar sensor readings from the hidden localization variables. It also depicts the two models that students are asked to build in order to implement a probabilistic localization algorithm, the movement model and the sensor model. Rather than learning particle filtering using instructor-supplied models, implementing a complete probabilistic localization solution on a robot includes building models that capture the movement uncertainty and sensor uncertainty of the students' specific platform.

The *movement model* models how the localization state variables change over time in response to robot movement.

In a Markov model the state variables at time t are a probabilistic function of the state variables at time $t - 1$ and the last movement action, $\Pr((x_t, y_t, \mu_t) | (x_{t-1}, y_{t-1}, \mu_{t-1}), \text{movement action})$. This model is robot-specific and varies with the drivetrain design, wall-following code, and odometry implementation for a specific robot (as well as environment-specific noise). Students can build these models through a combination of empirical experience with issuing movement commands to the robot and any prior knowledge of the robot motion, including the use of wheel-rotation sensors and odometry calculations. In the simplest case we assume the robot is always given a single movement action, such as move forward for one second, and the students empirically measure the

distribution of observed changes in x , y , and μ . If wheel rotation and odometry are employed, students will need to measure the wheel diameter, drive length, and gear ratio of their robots.

The *sensor model* models the expected sensor readings in different locations of the problem's frame of reference. The sensor model depends on how successfully the students design a robot that takes sonar readings only when it is a fixed distance from and parallel to the wall. In the best case, in our simplified localization problem, the sensor model need only capture the distribution of sonar readings when either in front of a wall or in front of a door (or somewhere in between, if necessary), $\Pr(\text{sonar}_{t-1}) = \text{wall or door} | (x_{t-1}, y_{t-1}, \mu_{t-1}), \text{features of known map}$). Note that the fixed-position sonar removes any need for pivoting toward the wall. We can also ignore the y -position and orientation variables if wall following is implemented well.

Step Three: Implementing Particle Filtering

Given a dynamic Bayesian network capturing movement and sensor models (including known map information), we can ask students to implement any probabilistic inference algorithm to solve the robot-localization problem (see Russell and Norvig [2003] for discussion). In this exercise we ask the students to implement approximate probabilistic inference using particle filtering (that is, Monte Carlo localization). Particle filtering has three basic steps: (1) particle propagation, (2) weighing, and (3) resampling. However, to implement particle fil-

tering on a physical robot, students consider additional steps to form a complete solution.

A complete particle filtering implementation includes the following steps: (1) distribute initial samples over x -locations in the map; (2) move robot a fixed distance (or duration)—include wall following and odometry computations here; (3) if end-of-course reached, reverse direction and go to 2; (4) propagate particles according to movement model (be careful to deal with end-of-map conditions correctly); (5) read sonar (take several sonar readings); (6) weigh particles according to sensor model; (7) resample particles according to weights; (8) send new belief state (particles) to PC host for visualization; and (9) calculate the variance in location across particles; if variance is small, stop the robot—location converged; else go to 2.

Students can experiment with the number of particles used in order to trade off accuracy for real-time updating and gain experience with limited-memory constraints. Implementing particle filtering on a low-cost robot platform really gets to the heart of these trade-offs. In our experiments with the RCX we are able to store at most 125 samples in the available memory (with communication protocol in memory for off-board visualization). This number of samples is enough to repeatedly achieve successful localization in this simplified problem. However, we provide the following additional tips to help replicate this success. For efficient memory usage, locations can be discretized and movements rounded to the nearest location bin. With so few particles, students may observe rapid convergence to wrong answers. The limited memory makes it difficult to apply common solutions to misconvergence such as periodically adding additional random (noise) samples or redistributing existing samples during localization. Limited forms of these corrections can be applied, as well as taking care to understand the sonar sensor's response profile and use it to build a useful sensor model. Maintaining parallel fixed-distance movement also helps avoid problems.

We have employed this exercise successfully to teach particle filtering in the classroom using the RCX. Additionally, we have presented this exercise as a four-hour workshop to teach educators with no background in artificial intelligence how to replicate these lessons.

Discussion

This article uses two example robotics problems to demonstrate the role of robotics in both motivating AI solutions and providing students experience with implementing com-

plete AI solutions. We show how the robot obstacle-detection problem can motivate learning neural networks and Bayesian networks. We also show how the robot-localization problem can motivate learning how to build complete solutions based on particle filtering. These lessons are demonstrated using two different low-cost robot platforms and can be replicated on any robot platform providing mobility, sensing, an effective development environment and sufficient processing resources. These lessons are designed for low-cost platforms so that they can be accessible to the broadest population of AI students.

Using robotics to teach undergraduate AI is time consuming. In our experience, however, we find that this effort is justified by stimulating student imagination and connecting students more closely to real-world applications. Robotics problems additionally encourage students to investigate how AI solutions interact with solutions from other fields. This article and others in this special issue can help reduce the effort required for AI educators and students to employ robotics in the classroom by pointing out available resources and experience. In our opinion, the hands-on active learning that students gain using robotics is worth the effort and provides value both to the student and to the future of the field itself. While the intention of this article is to convey the educational value of robotics, the article also provides evidence that AI can make low-cost robotics possible in many other contexts as well. For example, we show how expensive sensors (for example, laser range finders) can be replaced by a combination of inexpensive sensors (for example, IR) and AI solutions.

Tools and experimental data files described in this article (for example, obstacle detection training and validation sets, supporting scripts, software) are available at our web site,¹ currently under the heading "Laboratory Exercises."

Acknowledgements

We thank Brian Summers for his efforts in testing the Bayesian network exercises. We thank Zachary Dodds and Jerry Weinberg for their feedback on drafts of this article. This research is sponsored in part by a National Science Foundation (NSF) Instrumentation Award under grant CISE-9986105.

Note

1. <http://www.cs.hmc.edu/roboteducation>.

References

Bagnall, B. 2002. *Core LEGO MINDSTORMS Programming*. Upper Saddle River, NJ: Prentice Hall.

Borenstein, J., and Feng, L. 1995. Correction of Systematic Odometry Errors in Mobile Robots. In *Proceedings of the IEEE 1995 International Conference on Intelligent Robots and Systems (IROS '95)*. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

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*. Ann Arbor, MI: University of Michigan.

Dodds, Z.; Greenwald, L.; Howard, A.; and Weinberg, J. 2006. Components, Curriculum, and Community: Robots and Robotics in Undergraduate AI Education: A Survey. *AI Magazine* 27(1).

Dodds, Z.; Santana, S.; Erickson, B.; Wnuk, K.; and Fischer, J. 2004. Teaching Robot Localization with the Evolution ER1. In *Accessible Hands-on Artificial Intelligence and Robotics Education*, ed. L. Greenwald, Z. Dodds, A. Howard, S. Tejada, and J. Weinberg, 18–23. Technical Report SS-04-01. Menlo Park, CA: AAAI Press.

Fox, D. 2003. Adapting the Sample Size in Particle Filters through KLD Sampling. *International Journal of Robotics Research (IJRR)* 22(12): 985–1004.

Greenwald, L., and Artz, D. 2004. Teaching Artificial intelligence with Low-Cost Robots. In *Accessible Hands-on Artificial Intelligence and Robotics Education*, ed. L. Greenwald, Z. Dodds, A. Howard, S. Tejada, and J. Weinberg, 35–40. Technical Report SS-04-01. Menlo Park, CA: AAAI 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.

Klassner, F. 2006. Launching into AI's October Sky with Robotics and Lisp. *AI Magazine* 27(1).

Klassner, F. 2002. A Case Study of LEGO MindStorms' Suitability for Artificial Intelligence and Robotics Courses at the College Level. In *Proceedings of the Thirty-Third SIGCSE Technical Symposium on Computer Science Education (SIGCSE-2002)*, 8–12. New York: Association for Computing Machinery.

Martin, F. G. 1999. *The Handy Board Technical Reference*. Technical Report. Cambridge, MA: Media Laboratory, Massachusetts Institute of Technology.

Mayer, G.; Weinberg, J.; and Yu, X. 2004. Teaching Deliberative Navigation Using the LEGO RCX and Standard LEGO Components. In *Accessible Hands-on Artificial Intelligence and Robotics Education*, ed. L. Greenwald, Z. Dodds, A. Howard, S. Tejada, and J. Weinberg, 30–34. Technical Report SS-04-01. Menlo Park, CA: AAAI Press.

Nilsson, N. 1984. *Shakey the Robot*. Tech Note 323. Menlo Park, CA: AI Center, SRI International.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*, 2nd ed. Englewood Cliffs, NJ: Prentice Hall.

Stone, P. 2000. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. Cambridge, MA: MIT Press.

Thrun, S.; Fox, D.; Burgard, W.; and Dellaert, F. 2001. Robust Monte Carlo Localization for Mobile Robots. *Artificial Intelligence Journal* 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, Germany: Institute of Informatics and Information, University of Stuttgart.



Lloyd Greenwald is currently a member of technical staff at Bell Labs in the Internet Research Department pursuing research on advanced algorithms for network security including anomaly detection, vulnerability analysis, penetration testing, wireless security, and mobile ad hoc networks.

Greenwald received a Ph.D. and Sc.M. in computer science from Brown University and a B.S.E. in computer science and engineering from the University of Pennsylvania. Greenwald's robotics education contributions were primarily pursued while he was an assistant professor of computer science and director of the Intelligent Time-Critical Systems Lab at Drexel University. He can be reached at lgreenwa@alumni.upenn.edu.



Donovan Artz is a Ph.D. student at the Information Sciences Institute, University of Southern California, and formerly a MS student in computer science at Drexel University. His interests include knowledge representation, multiagent systems, network security, and statistical machine learning. Although

Donovan's current work is modeling trust on the web, he is still actively involved in robotics as a mentor for a local high school club.



Yogi Mehta is pursuing a Master's in computer science at Drexel University. He is currently working at Susquehanna International Group as a co-op (software developer).



Babak Shirmohammadi earned a B.Sc. in computer science from Baha'i Institute for Higher Education (BIHE) in Iran in 1998. He has worked in industry in various areas including embedded programming, automation, and robotics. He earned his M.Sc. in computer science (AI and robotics track)

from Drexel University in 2005. He is currently working as a research specialist at GRASP laboratory in the University of Pennsylvania and is also working towards his Ph.D.