Exploiting MindStorms NXT: Mapping and Localization Projects for the AI Course

Myles McNally, Frank Klassner and Christopher Continanza

Alma College
Alma, MI 48801
USA
mcnally@alma.edu

Villanova University
Villanova, PA 19085
USA
frank.klassner@villanova.edu

Villanova University
Villanova, PA 19085
USA
christopher.continanza@villanova.edu

Abstract

This paper describes two major student projects for the artificial intelligence course – Mapping using Bayesian filter and Monte Carlo Localization. These projects are also suitable for a course focusing on autonomous mobile robotics. Implemented using the new Lego MindStorms NXT platform, a major improvement over the previous RCX platform, these projects are part of the LMICSE web site, a large repository of MindStorms-based computer science curriculum materials.

Introduction

With the use of robotics moving down the computer science curriculum into CS 1 and 2 (e.g., McNally 2006), there is a need to develop more sophisticated robotics projects for the traditional undergraduate artificial intelligence course. At the same time there is a need to control costs, as many institutions of higher education cannot afford to purchase advanced robotics platforms. We believe the new Lego MindStorms NXT platform to be a solution to the latter need, and in this paper we describe projects that attempt to meet the former need using this new robotics platform.

Lego's release of NXT is a major improvement over their previous platform, the RCX. It contains a more powerful processor, more memory, and employs enhanced sensor and effector suites. We believe it to be an effective platform for integrating robotics topics into the CS curriculum, particularly within the artificial intelligence course. We begin this paper with a brief description of this new platform. We then describe two major student projects: Occupancy Grid-based Mapping using Bayesian filtering and Robot Localization using the Monte Carlo approach (also known as Particle Filtering). We also describe a warm-up exercise in which students determine the capabilities of their sonar sensor unit. These projects are available at LMICSE – Lego Mindstorms in Computer Science Education (Klassner et al 2007), a major web repository of MindStorms-based computer science curriculum materials.

Contrasting NXT to the RCX

Lego released RCX MindStorms in 1997. Nine years later Lego released NXT. (A basic NXT robot design is shown in Figure 1.) The hardware and firmware of the new kit's programmable brick feature several advances over the RCX design. These include a local file system on 256KB flash RAM, 64KB RAM for program execution, a 48MHz ARM7 CPU with support for Bluetooth radio frequency (RF) communication, a large 100x64 pixel LCD display, a USB 2.0 port for program download, and four sensor ports. User programs can be as large as 160 KB (more if system sound files are removed from the flash RAM drive). NXT firmware uses the 64KB onboard RAM to allocate memory for programs in execution.

Figure 1: A basic NXT robot with side facing sonar

Like the RCX, the NXT supports "immediate command" and "program command" modes, which means it can be controlled remotely through IR signals or controlled directly through an onboard program. So, although the in-

creased memory of the NXT will allow many programming solutions to occur "on-board," the immediate command mode will support advanced, memory intensive AI projects by running system code on a PC while running a smaller remote-control program on the NXT. Since the NXT uses RF communication, the line-of-sight issues with the RCX's IR technology have been eliminated.

Odometry is improved with the NXT's integration of rotation sensors into the motors. Measuring 360 clicks per rotation, these sensors also do not tie up a sensor port, as in the RCX. The NXT kit also includes a quite accurate ultrasonic (sonar) sensor, another enhancement over the standard RCX kit.

As of this writing programming language support for the NXT is wanting, although ports of LeJOS (a Java based environment described by Bagnall 2002) and RCXLisp (Klassner 2004) are under development. At this moment we employ the Java-based NXT remote control package ICommand, still under development by Brian Bagnall (Bagnall 2006). This package is built on top of standard Java and permits the development of sophisticated AI solutions.

Project One: Sonar Testbed

As a warm-up project, students are asked to explore the capabilities of their robot's sonar unit. In particular they are asked to perform experiments that measure the readings returned by the unit when objects are placed at various distances from it. They are also asked to measure the unit's lateral resolution – its sonar cone. These results are then employed in project two. To make this project somewhat easier for students we have created a sonar testbed using a large-scale printer. Figure 2 shows the sonar sensor testbed set-up.

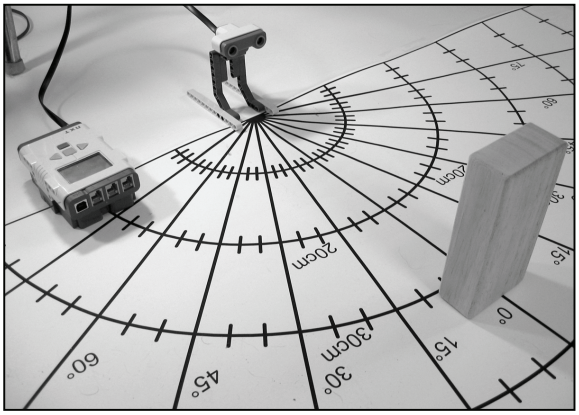

Figure 2: The NXT, sonar unit, and sonar testbed

This project requires that students write a simple program that repeatedly displays the current sonar reading. Students use ICommand for this task, using the NXT's immediate command model to remote control the NXT. As additional programming language support becomes available, students will be asked to write on-board programs in either Java or Lisp.

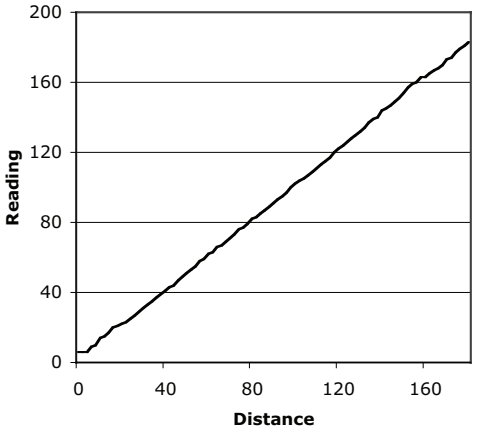

Figure 3: Sonar reading vs. object distance

The MindStorms sonar unit is advertised to return accurate measurements linear to distance. Figure 3 shows results of testing this unit. It proves to be very accurate in distances from 6 to 180 centimeters, with returned distances usually slightly larger than actual distances, but within one to four centimeters. Objects at distances beyond 180 centimeters were not reliably located. Lateral resolution is very good in a cone of 30 degrees out to 180 centimeters, and showed wider secondary bulbs in the usual way at shorter distances.

Project Two: Mapping

Overview

In this project students use the NXT, equipped with a side facing sonar sensor, to map one side of a "hallway." They are told to imagine a long hallway with a series of open doors along one side, unevenly distributed and not all of the same width. The robot can move back and forth along the hallway, and is at anytime either in front of one of the doors or is along a wall segment.

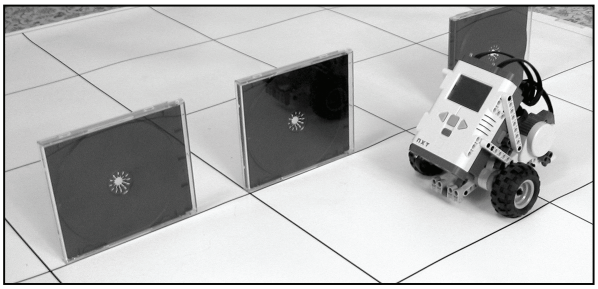

Figure 4: The hallway

We use the NXT robot configuration shown in Figure 1 for this project. Note that as compared to the Lego-

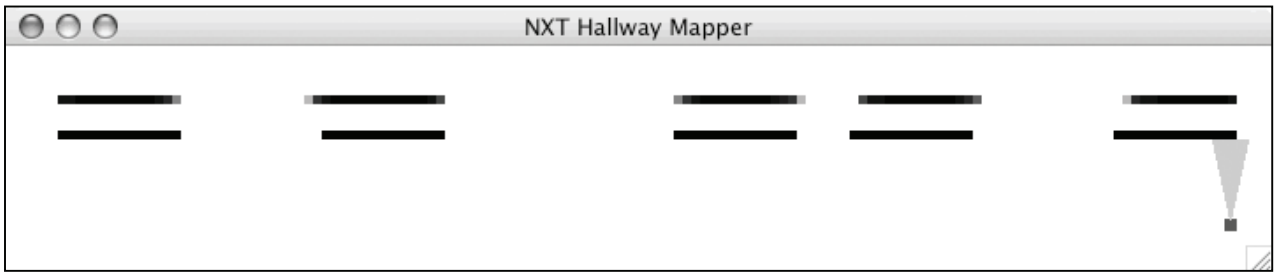

Figure 5: After a trip up and down the hallway

recommended basic NXT robot design the position of the NXT brick has been moved forward over the drive wheels for better movement accuracy. The side-facing sonar sensor is mounted just below the top of the NXT brick (not visible in Figure 4). Project two is to use this robot and its sonar sensor to map the hallway, which in our case is just a series of CD cases placed at various distances from each other.

Algorithm

This project is appropriate when discussing probabilistic models, in particular Bayesian filters, a topic addressed in many standard AI textbooks, such as Russell & Norvig 2002. Our approach employs occupancy grids and uses what Thrun 2001 calls the Standard Occupancy Grid Mapping Algorithm. Let m be the occupancy grid map, and m_x the grid cell at location x. The algorithm estimates grid cell values based on sonar readings given robot poses.

Inputs: occupancy grid X
measurement model p
time step limit T

// Initialization – set odds to zero
for all grid cells <x> in X:
$l_x = 0$

// Map for T time units
for all time steps t from 1 to T:
z_t = measurement (sonar and pose)
for all grid cells <x> in perceptual range of z_t:
$l_x = l_x + \log[\, p(m_x \mid z_t) \,] - \log[\, 1 - p(m_x \mid z_t) \,]$
move robot

// Recover of occupancy probabilities
for all grid cells <x> **do**
$p(m_x \mid z_1, \ldots, z_T) = 1 - 1/e^{lx}$

Figure 6: Log-odds occupancy grid-based mapping algorithm

Let $z_1,\ldots z_T$ be the measurements at times 1 through T (the sonar readings and current robot poses) and $p(m_x \mid z_i)$ the probability of grid cell m_x being occupied given measurement z_i. We provide to the student the following log-odds mapping algorithm, adapted from Thrun 2001 and simplified by the assumption that the prior probabilities of grid cells being occupied is 0.5. (See Thrun's work for a derivation of the algorithm.) We note here that the log odds of a cell being occupied may be any real value, and must be normalized back into probabilities.

One problem students must address is the development of an appropriate measurement model. Since the sonar beam is rather wide, many grid cells will be in the perceptual range of the robot during a sonar measurement. If any one of them is actually occupied, the sonar reading will likely be positive for "wall." But which one is occupied (or are they all)? And there is also a chance that the reading is simply wrong and no wall segment is present. We have found that a relatively low occupancy probability works well, even 0.6.

Visualization

We provide to the student a visualization class (HallwayMap.java) that displays the current state of the computation, along with a skeleton main class. The hallway mapping view is initialized with the actual map (for comparison), the initial state of the occupancy grid (all values zero), and the initial robot location. As shown in Figure 5, the view shows the robot along with the spread of its sonar beam below the actual map. Above that is shown the current state of the occupancy grid, using gray scale to show the current probability of each cell. The grid in this example is quite small, representing one centimeter.

The robot is assumed to start mapping at location 0 (the right end of the hall), moving in steps to the other end of the hall, and then return in steps. Figure 5 shows the result after a traversal down and back the hallway. While not perfect, a fairly accurate map has been computed. The algorithm requires precise pose information, but the NXT's odometry, while good for an inexpensive platform, is less than perfect. Hence the degree of error in the result.

Website Materials

On our website we provide the following to support this project:

- A complete project write-up, including references to appropriate literature and photos of robot and hallway designs.

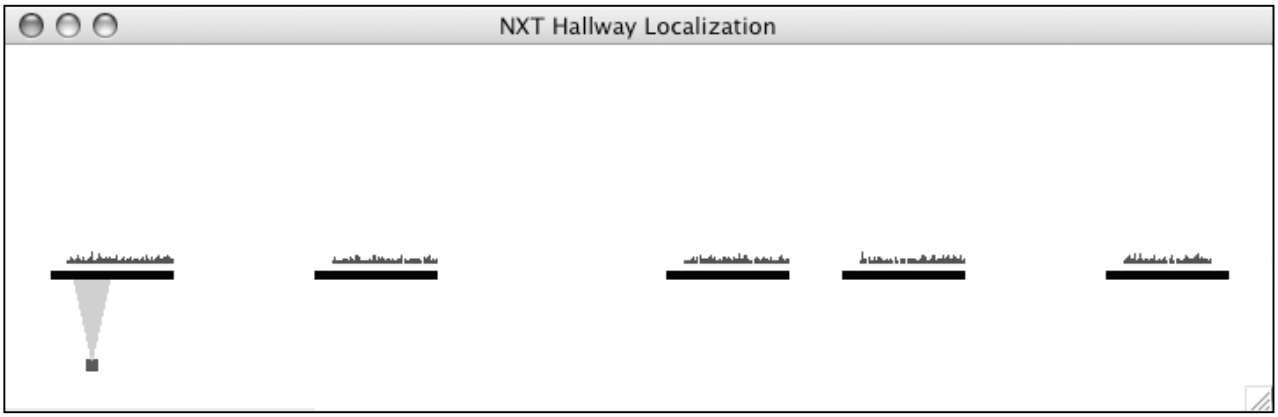

Figure 7: Robot has moved a short distance down the hallway

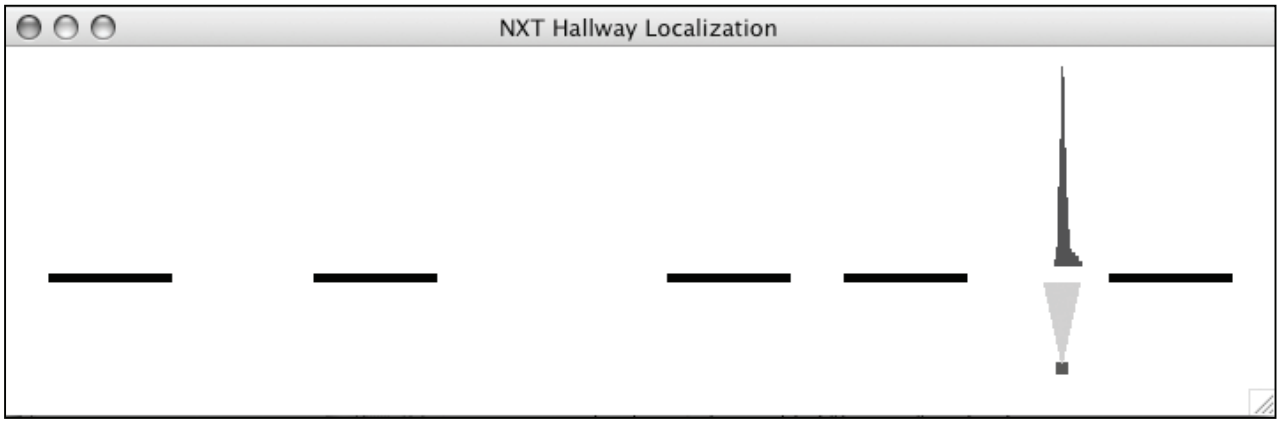

Figure 8: Robot is well localized as it approaches the end of the hallway

- A zip of the HallwayMap visualization class, along with a skeleton main program.
- Split-screen videos that show our robot solving the mapping problem. The top pane shows the robot moving linearly in front of the wall/door series, while the bottom pane shows the HallwayMap visualization.

Complete solution code will be provided to faculty on request.

Project Three: Localization

Overview

Monte Carlo Localization (MCL), also known as Particle Filtering, is a relatively new approach to the problem of robot localization - estimating a robot's location in a known environment, given its movements and sensor reading over time. In this project students solve the **global localization problem**, where the robot does not know its starting position but needs to figure out its current location. This is in contrast to the **position tracking problem**, where the robot knows its starting position and just needs to accommodate the small errors in its odometry that build up over time. As in project two students solve this problem in a one dimensional world, the hallway. We use the same robot design as in project two.

Our approach employs the particle filtering approach described by Fox 2003, whose linear example was implemented by Greenwald et al 2006. This approach can easily be adapted to handle the "kidnapped robot" problem, in which a robot, during localization, is suddenly transported to another location and must re-localize.

The Problem

Given a known map but an unknown starting location, the student's robot must attempt to move to a predefined goal point along the hallway. As in project two, to solve this problem, the robot is directed to repeatedly move forward a fixed distance and takes a sonar reading. If it thinks it has reached the end of the hallway, it starts backing up instead of moving forward. Often it needs to move back and forth along the hallway a few times before it accurately knows where it is, but usually it will be able to reliably determine its location in one pass back and forth. The length of time required depends on the level of ambiguity in the map (it

Inputs: Distance u_t,
sensor reading z_t,
travl distance u_t
sample set $S_t = \{(x_t^{(i)}, W_t^{(i)}) \mid i = 1,\ldots n\}$

// First update the current set of samples
for i = 1 **to** n **do**
    // Compute new location
    x_t = updateDist(x_{t-1}, u_t)
    // Compute new probability
    $w_t^{(i)} = p(z_t \mid x_t^{(i)})$

// Then resample to get the next generation
S_{t+1} = null
for i = 1 **to** n **do**
    Sample an index j from the distribution given by the weights in S_t
    // Add sample j to the set of new samples
    Add ($x_t^{(j)}$, $w_t^{(j)}$) to S_{t+1}

return S_{t+1}

Figure 9: Computing the next generation of particles (adapted from Fox 2003)

can't be totally ambiguous, otherwise the robot will not localize).

In addition to possible errors in sonar readings, we also no longer assume totally accurate odometry. So students must develop probabilistic models for both sonar and movement.

The Basic Approach of MCL

The Monte Carlo approach to localization is based on a collection of samples (which are also known as particles). Each sample consists of a possible location the robot may currently occupy, along with a value that represents the probability that the robot is currently at that location. Strictly speaking these are not probabilities and are often referred to as importance weights.

How many samples should be used? The more you have, the more quickly the program will converge to a correct solution, but at the cost of more computational time and hence slower robot movement.

The general idea is as follows:

1. Initialize the set of samples (the current samples) so that their locations are evenly distributed and their importance weights are equal.
2. Repeat until done with the current set of samples:
    a. Move the robot a fixed distance and then take a sensor reading.
    b. Update the location of each of the samples (using the movement model).
    c. Assign the importance weights of each sample to the likelihood of that sensor reading given that new location (using the sensor model).
    d. Create a new collection of samples by sampling (with replacement) from the current set of samples based on their importance weights.
    e. Let this new collection become the current set of samples.

The robot is considered to be at the location centered in the largest cluster of samples. When this is reasonably close to the goal location, the program terminates.

Figure 9 contains the details of the updating step (step 2 above) as adapted from Fox 2003. It generates at time t the next set of samples S_{t+1} from the current set S_t. The x_t are the locations and the w_t the probabilities – so an (x_t, w_t) pair represents a sample. The distance traveled is u_t, and the sensor reading is z_t. We use superscripts to indicate the particular individual samples, so $x_t^{(i)}$ is the location of sample i at time t. n is the number of samples.

Visualization

As in project two, we provide to the student a visualization class (HallwayLoc.java) that displays the current state of the computation along with a skeleton main class. The hallway view is initialized with the actual map, an initial robot location (assumed to location zero), and the initial particle set (which is uniformly distributed over the map length). The view shows the robot along with the spread of its sonar beam below the actual map. Above that is a histogram of the particle locations. See Figures 7 and 8.

In Figure 7 the robot has just begun its mapping, having been placed at the left end of the hallway and moving right. Note that the particles are fairly uniformly distributed over wall sections. No particles are shown above door sections. Figure 8 shows the state of the computation after the robot has moved right to the position shown. The robot is well localized by this point.

Website Materials

On our website we provide the following to support this project:

- A complete project write-up, including references to appropriate literature and photos of robot and hallway designs.
- A zip of the HallwayLoc visualization class, along with a skeleton main program.
- As before, videos of a sample solution in action, showing in split screens the actual robot in its mapping activities and the corresponding HallwayMap visualization. Three such videos are provided, with increasingly more ambiguous hallways and corresponding increasing difficulty in finding the goal location.

Complete solution code will be provided to faculty on request.

Conclusion

Lego MindStorms NXT provides enhanced opportunities for incorporating appropriate robotics projects in the undergraduate AI course or into a course in autonomous mobile robotics. In this paper we have described two major projects, mapping and localization, that utilize the new MindStorms NXT platform. These projects are available at LMICSE, a repository for MindStorms-based CS curricular materials.

Acknowledgements

This work was partially supported by a National Science Foundation CISE Educational Grant (EIA-0306096).

References

Bagnall, B. 2002. *Core LEGO MINDSTORMS Programming: Unleash the Power of the Java Platform*. Upper Saddle River, NJ: Prentice Hall.

Bagnall, B. 2002. *ICommand*. http:lejos.sourceforge.net. Last accessed on 11/20/06.

Fox, D., Burgard, W., Dellaert, F., and Thrun, S. 1999. Monte Carlo Localization: Efficient Position Estimation of Mobile Robots, In *Proceedings of the 1999 National Conference on Artificial Intelligence (AAAI 1999)*.

Fox, D. 2003. Adapting the Sample Size in Particle Filters Through KLD-Sampling. *International Journal of Robotics Research* 22(12):985-1003.

Greenwald, L., Artz, D., Mehta, Y., and Shirmohammadi, B. 2006. Using Educational Robotics to Motivate Complete AI Solutions. *AI Magazine* 27(1):83-95.

Klassner, F. 2004. Enhancing Lisp Instruction with RCXLisp and Robotics. In *Proceedings of 35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2004),* 214-218. Norfolk, VA: ACM Press.

Klassner, F., Lawhead, P. and McNally, M. (2007): Lego Mindstorms in Computer Science Education, http://www.mcs.alma.edu/LMICSE/. Accessed 1 Jan 2007.

McNally, M. 2006. Walking the Grid: Robotics in CS 2. In *Proceedings of the Eighth Australasian Computing Education Conference,* 151-155. Hobart, Australia: Australian Computer Society Inc.

Russell, S. and Norvig, P. 2002 *Artificial Intelligence: A Modern Approach (2nd Ed.)*. Upper Saddle River, NJ: Prentice Hall.

Thrun, S. 2001. Learning Occupancy Grids with Forward Models. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS 2001)* Hawaii.

Thrun, S. 2002. Robotic Mapping: in *A Survey in Exploring Artificial Intelligence in the New Millennium*. San Francisco, CA: Morgan Kaufmann.