Easy Integration of LEGO Mindstorms into Vacuum World Simulations

David R. Musicant, Carleton College (dmusican@carleton.edu), Andrew Exley (andy@moozer.com)

Summary

LEGO Mindstorms robots are a popular platform for teaching artificial intelligence (AI) due to their low cost and the availability of free, quality programming libraries. One of the challenges in integrating Mindstorms into pre-existing AI classes is in constructing new curricula and new assignments to encompass the use of robots. Additionally, students must take time to learn to program them. This programming is generally of a systems nature, and though certainly a worthwhile experience, takes class time away from other AI material.

We present VacuumBot, a robot design and a software library developed with the leJOS language [1] for LEGO Mindstorms. VacuumBot contains a high level library that allows students to implement Russell and Norvig's "Vacuum World" [3] with physical robots by programming in Java, but with a minimum amount of low-level programming to learn. Thus, an AI class that does not emphasize robotics can focus on other areas, while students can still experience the excitement of working with Mindstorms robots.

The Vacuum World Environment

The "Vacuum World" environment is a creative and simple way to introduce the idea of intelligent agents. A vacuum cleaner is described as living in a rectangular room, broken down by grid lines. Each grid square represents a discrete location that the vacuum may visit. There is a unique "home" location for the vacuum, and all other locations may be empty or may contain dirt. The goal of the vacuum cleaner agent is to traverse the room, clean up all the dirt, return home, and stop. This is made somewhat challenging in that the vacuum has fairly limited information: its sensors report only whether or not there is dirt below it on its current square, whether or not it is at its home location, and whether or not it has just bumped into a wall. Its actions are limited as well. The vacuum can move forward, turn left, turn right, or clean its current grid square.

To summarize, the following sensory info is available:

- At home? (Yes / No)
- On dirt? (Yes / No)
- Bump sensor activated? (Yes / No)

Likewise, the following actions are available:

- Turn left
- Turn right
- Go forward
- Suck dirt
- Stop

A variety of techniques can be used to solve this task, such as random walks, systematic patterns, and allowing the vacuum to "remember" where it has been previously. Many more variations are possible. Russell and Norvig have provided an excellent library in Lisp for observing this environment virtually on a computer screen. Their system displays a grid using text graphics, and provides an API for moving the vacuum around. One of the main advantages of this API is that it is fairly high level. Students can write code such as:

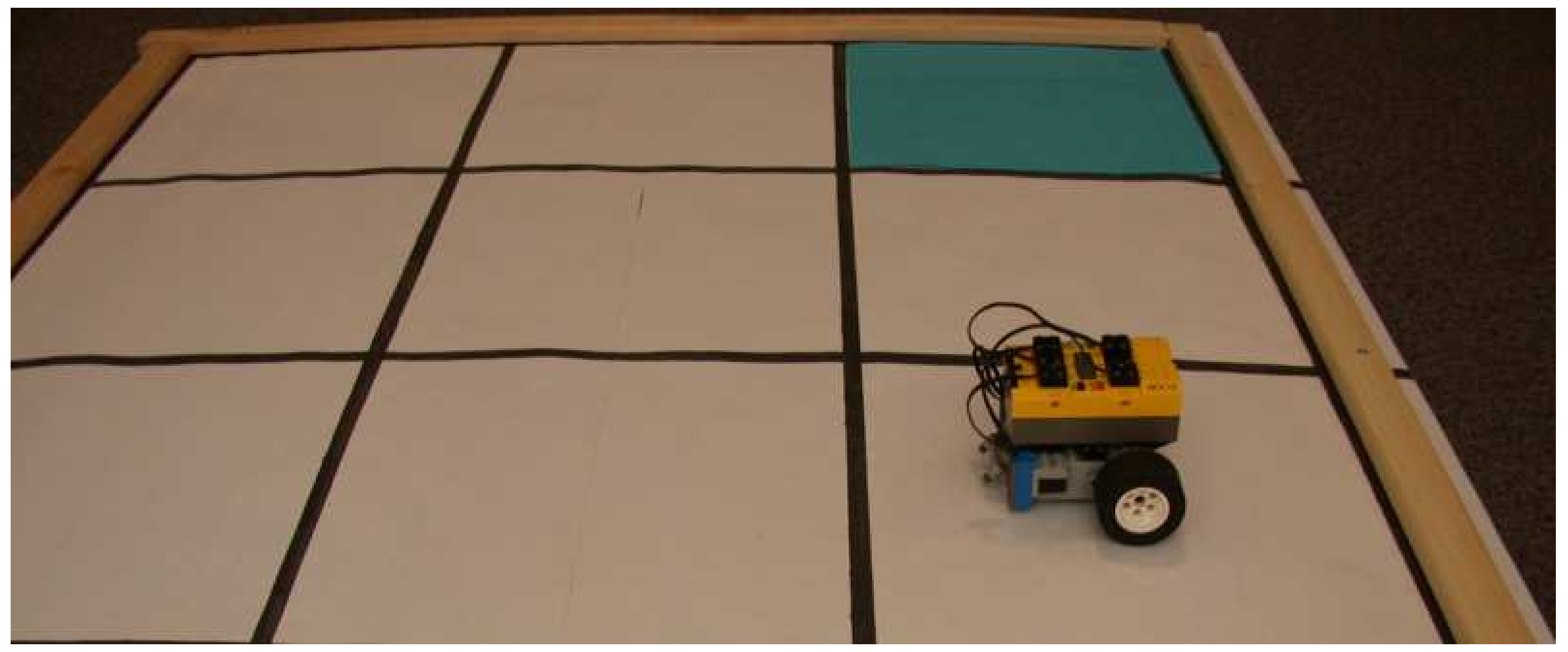
Russell / Norvig Style Lisp code

```
(forward)
(setf bump (first (get-percept)))
(if bump (turn-left))
```

In the past, we used Vacuum World in this manner with great success. However, students expressed a strong desire to do this with "real" robots.

The VacuumBot and Environment Physical Design

In order to help students get the excitement of using LEGO Mindstorms without having to implement them from scratch and handling the associated low level programming, we constructed VacuumBot. The design is minimalistic: it contains only two wheels [2]. The front of each robot is equipped with two light sensors, each mounted on opposite sides facing down towards the paper. These sensors serve two purposes: they detect whether the robot is on an empty space, dirt, or home, and they are used to correct the robot's motion. Likewise, the front of the robot is also equipped with a bump sensor that determines if the robot has hit the edge of the grid. All parts for this robot are found in a single basic Mindstorms kit. No extra parts such as rotation sensors are necessary.



The robot travels on grids that we have built on white paper, mounted to sheets of plywood. Black magic marker lines separate the sections. A blue piece of paper is used on a grid spot to indicate the home location, and a purple piece of paper is used to indicate dirt. When the robot is instructed to "suck dirt," it emits a beep. In this sense, our system is only an approximation to a true simulation: our robot indicates that it attempts to suck dirt, but we do not have an actual mechanism for removing it.

leJOS: The Backbone for our API

One of our primary goals was to ensure that controlling the robot remains at a very high level. leJOS and other LEGO languages typically require the user to control the motor for each wheel separately, indicating precisely how long each wheel should spin. For example, the following leJOS code instructs our robot to move forward until the bump sensor is triggered, then to turn left. Turning left is achieved by spinning the two wheels in opposite directions for a fixed period of time:

Sample leJOS program: move to bump, then turn left. Precise, but complex.

```
// Register a listener to activate when the bump sensor is hit
double bumpSensor;
Sensor.S2.setTypeAndMode(1,0);
Sensor.S2.addSensorListener(new SensorListener() {
  public synchronized void
      stateChanged(Sensor src, int oldValue, int newValue) {
    if(src == Sensor.S2 && oldValue > 932 && newValue < 933) {
      bumpSensor = true;
      // Back off the robot from the bump
      Motor.A.backward();
      Motor.C.backward();
      try { wait(400); }
      catch(InterruptedException e) { }
      Motor.A.stop();
      Motor.C.stop();
      bumpSensor = false;
    }
  }
});

// Move forward
Motor.A.setPower(1);
Motor.C.setPower(1);
Motor.A.forward();
Motor.C.forward();
// Wait for bump to happen
while (!bumpSensor);
// Reverse direction of one wheel to turn robot
Motor.A.backward();
Motor.C.forward();
try{ wait(TURN_DELAY); }
        catch (InterruptedException e) { }
Motor.A.stop();
Motor.C.stop();
```

Note that this code is not complete, as it does not interact with the light sensors to help keep the robot "between the lines."

The VacuumBot Programming Environment

We wanted our students to be able to issue a single command such as "moveForward" to move the robot. Our VacuumBot API thus hides away functionality that uses the two light sensors as a means to keep the robot centered in each square. If the robot crosses one of the black strips at an angle, the times at which the sensors observe the strip are different. We use this difference in time to take corrective action and straighten out the robot's path.

Ultimately, our students are thus able to use much simpler Java code for controlling the robot. The following VacuumBot code fragment does the same work as the above more complicated leJOS code:

VacuumBot version of leJOS program.

```
VacuumBot bot = new VacuumBot();
bot.moveForward();
if(bot.hitBump())
    bot.turnLeft();
bot.shutdown();
```

The simplicity of this API for controlling the robot frees the student to concentrate on more complicated algorithms and techniques for navigating the robot around the grid.

To extend this example further, the following VacuumBot fragment directs the bot to travel forward to the edge of the grid, turn around, come back, and shut down if it finds itself at "home" (on a blue piece of paper). Along the way, it emits a beep to simulate sucking dirt if it finds itself on "dirt" (on a purple piece of paper).

VacuumBot code to travel forward and back, looking for dirt.

```
VacuumBot bot = new VacuumBot();
bot.moveForward();
while(!bot.atHome()) {
  if(bot.hitBump()) {
    bot.turnLeft();
    bot.turnLeft();
  }
  if(bot.onDirt())
    bot.suck();
  bot.moveForward();
}
bot.shutdown();
```

At the start of each exercise, the student calibrates the robot using a separate utility that we provide. This allows the student to provide appropriate light sensor thresholds, as these thresholds vary from one light sensor to another and also depend on room lighting. The student also calibrates how much time a left or tight turn takes. This cannot be automated within our code because this calibration number changes as the batteries in the robot drain. Some of this could have been handled more automatically with optional LEGO accessories such as rotation sensors, but we were motivated to save money and use only that technology that came with a standard Mindstorms kit.

Conclusions

VacuumBot is a robot design and a simple open source library for implementing Russell and Norvig's Vacuum World with LEGO Mindstorms. It works quite well in our AI class at Carleton College, and seems to require almost no troubleshooting at all once the infrastructure is in place. Our software and robot designs can be downloaded from http://vacuumbot.sourceforge.net.

Special thanks to Carleton student Marie Joiner, who worked tirelessly on perfecting the portable grid for this presentation.

References

[1] leJOS: Java for the RCX. `http://lejos.sourceforge.net`.

[2] Frank Klassner and Scott D. Anderson. Using LEGO Mindstorms across the computer science curriculum. Workshop at SIGCSE 2002.

[3] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, Inc., New Jersey, second edition, 2003.

Carleton College