MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.01—Introduction to EECS I
Spring Semester, 2008

Software Lab 13, Issued Tuesday, May 6

Overview of this week's work

In software lab

- Work through the software lab. **Written solutions to problems 1– 21 will be due in design lab. Your answers can be terse, but you must show all computations.**

Before the start of your design lab on May 8 or 9

- **Hand in written solutions to all software lab problems.**
- Do the survey in tutor problems 13.1.
- Read the entire description of the design lab, so that you will be ready to work on it when you get to lab.

In design lab

- Work through the design lab. No written hand-in required for this lab, but you must be completely checked off.

Software Lab: State estimation part 2

Recap of how to use this software

This week we'll continue working with the non-deterministic grid-world simulator. *You should work with a partner on this.*

Don't use the Idle Python Shell for this lab. You can use the Idle editor, but you cannot run the code from inside Idle.

In the `lab12` folder you will find `se.py`. Open this file in Idle, but **do not try to evaluate the Python commands in the Idle Python Shell.**

Then, open a Terminal window (if you're on Athena, remember to do `add -f 6.01`), and connect to the `lab12` directory

```
cd ~/Desktop/6.01/lab12
```

and type `python`.

```
> python
Python 2.5 (r25:51918, Sep 19 2006, 08:49:13)
[GCC 4.0.1 (Apple Computer, Inc. build 5341)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

If you are using your own laptop, download the `lab12` *files from the calendar page on the web. If don't know how to do the operations above on your own computer, then you can start Python, and type*

```
>>> import os
>>> os.chdir('wherever\you\store\your\lab12\files')
```

And continue as follows.

In this Python, type

```
>>> from se import *
```

As the lab goes along, if you edit `se.py`, then you'll need to go back to this window and type

```
>>> reload(se)
```

And if you define a new name in `se.py`, you'll have to do

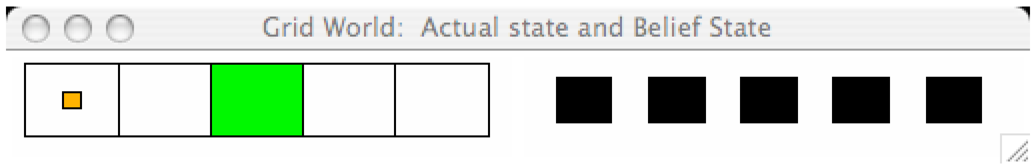
```
>>> from se import *
```

again as well.

You'll see a set of procedures at the end of the `se.py` file, which will make example worlds of different kinds. We'll start by working with the world defined by `make51p` (which stands for 5 by 1, perfect). In your Python (not Idle) shell, type

```
>>> w = make51p()
```

As a result, you should see a window that looks like this:



This is a world with 5 possible “states”, each of which is represented as a colored square (on the left). The possible colors of the states are white, black, red, green, and blue. In this example, four squares are white and one is green. There is a small orange rectangle representing the square that our simulated robot is actually occupying. On the right are five squares that start out being black; the color in those squares represents how likely the robot thinks it is that it’s in that square. This is called the *belief state.* The colors illustrating the belief state are: black, when the value is what the uniform distribution would assign (0.2 for 5 states), shades of green when the probability is higher than the uniform, and shades of red when it is below the uniform value.

The arguments to the `makeGridSim` function are:

- The dimension of the world in x
- The dimension of the world in y
- Four lists of coordinates, each specifying the location of colored squares. The first list gives the locations of black squares, the second red, the third green, the last blue. Squares unspecified in any of those lists are white.
- A pair of indices specifying the robot’s initial location
- A model of how the sensors work
- A model of how the actions work

So, this grid world is 5-by-1, with one green square, the robot initially at location (0,0), and perfect sensor and motion models.

You can issue commands to the robot by typing:

```
w.run()
```

and responding to the text prompts. Typing `done` returns control to Python, but the window will remain. **Do not kill the window until you are completely done with it.** You can type:

```
w.reset()
w.run()
```

to start interacting with the window again after you’ve typed `done`.

Belief state update

The robot’s *belief state* is a probability distribution over possible states of the world. In this world, there is one underlying world state for each square; each of these states has a probability value assigned to it, and these values sum to 1. In the simulator, the current belief state is shown by the squares on the right, with redder values closer to zero and greener values closer to one. It is also printed out whenever you move; in fact it is printed twice: once after the transition update and again after the sensing update, as explained below. (If you get tired of seeing the printout, you can always type `w.verbose = False` to shut it up.)

Just as in the HMM, the belief state is updated in two steps, based on the state transition model and the observation model. Study the state update rules in the notes, and convince yourself that at the end of this, all of the entries in the belief state will sum to 1.

Practice

This stuff is hard to build up an intuition for. We'll ask you to work through the state estimation equations for some simple examples, so you can come to understand in detail how things work.

Let's start by practicing in a world without noise. In our set-up, `w` is just such a perfect world. Either make a new instance of it or do `w.reset()`, which will set the belief state to $(0.2, 0.2, 0.2, 0.2, 0.2)$. This is often called a *uniform* distribution. The robot has no idea where it is, and considers all states equally likely.

Calculate an answer to these questions by hand before trying it in the simulator. If you get a different result than you expected, convince yourself either that there is a bug in the simulator or why it's right. Show your computation in detail when you hand in answers.

Question 1: First, what is the robot's prior belief $b(s_i)$ for each of the states?

Question 2: When you do `w.run()`, it automatically does a `stay` action. What is the belief state after taking the state transition (but not the observation) into account? Start by computing $P(s_i|s_j, \textit{stay})$ for all i, j. Now, use this to compute $b_{new}(s_i)$ for all i.

Question 3: Now, the robot will see "white" because it's in a white square and there's no noise. What will the belief state be after this?

- First, for each state, figure out $P(\textit{white}|s_j)$ for all states.
- Then compute $P(\textit{white}|s_j)P(s_j)$ for each state. Remember that at this point the $P(s_j)$ we are using is the belief state that resulted from the transition, not the original prior.
- Now, compute $P(\textit{white})$.
- Finally, compute $P(s_i|\textit{white})$.

Question 4: If we were to tell the robot to go east, what would the belief state be after taking the state transition (but not the observation) into account?

Question 5: Now, the robot will see "white" again because it's moving to a white square and there's no noise. What will the belief state be after this?

Question 6: Which action could the robot take at this point to make its location completely unambiguous?

Question 7: Now, do `w1 = make201pp()`. Try driving the robot toward the east. Don't solve the problem numerically, but be sure you can explain why the belief state is behaving the way it does.

Noisy Practice

Now, let's try adding in sensor noise (look at the `noisySensorModel` in `se.py`). We'll use `w2 = make51ns()`, which still has perfect motion but noisy sensors. Again, try to do these computations before running the simulation, using the schematic shown in the notes.

Question 8: After the initial stay action, but before the observation, what would the belief state be?

Question 9: Now, what's the distribution over what the robot sees? That is, what is $\mathsf{P}(\mathsf{O}_1 = \mathsf{o}_1 | \mathsf{A}_1 = \textit{stay})$, for all values of o_1?

Question 10: Compute the next belief state if the robot sees "white". That is, what is $\mathsf{P}(\mathsf{S}_1 = \mathsf{s}_1 | \mathsf{O}_1 = \textit{white}, \mathsf{A}_1 = \textit{stay})$, for all values of s_1?

Question 11: If we were to tell the robot to go east, what would the belief state be after taking the state transition (but not the observation) into account?

Question 12: Now, what's the distribution over what the robot sees? That is, what is $\mathsf{P}(\mathsf{O}_2 = \mathsf{o}_2 | \mathsf{A}_2 = \textit{east})$, for all values of o_2?

Question 13: Compute the next belief state if the robot sees "white". That is, what is $\mathsf{P}(\mathsf{S}_2 = \mathsf{s}_2 | \mathsf{O}_2 = \textit{white}, \mathsf{A}_2 = \textit{east})$, for all values of s_2?

Question 14: Compute the next belief state if the robot sees "green." That is, what is $\mathsf{P}(\mathsf{S}_2 = \mathsf{s}_2 | \mathsf{O}_2 = \textit{green}, \mathsf{A}_2 = \textit{east})$, for all values of s_2?

Question 15: Go east again. (That's twice now). Explain why, in this particular run, in your simulation, you got the result you got.

Question 16: What (approximately) would happen if you then did the `stay` action 10 times? Explain.

Now, let's try noisy actions only, using `w3 = make51nm()`, starting from the initial state. Show your computation when you hand in answers.

Question 17: When you do `w3.run()`, it automatically does a `stay` action. What is the belief state after the stay action, but before it gets the observation?

Question 18: What is the next belief state if it observes "white"?

Question 19: Move the robot to the green square. What is the belief state now?

Question 20: Can it ever get confused after moving some more? Why or why not?

Question 21: Make `w4 = make55()`. This world has noisy sensing and action. Drive the robot around for 10 or 20 steps. Try your best to confuse it. Describe a situation in which the location that has the highest probability in the belief state is not the actual location of the robot, and explain why it happened.