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

Work for Week 12, Issued Tuesday, April 29

Overview of this week's work

In software lab

- Work through the software lab.

Before the start of your design lab on May 1 or 2

- Read the class notes and review the lecture handout.
- Do the on-line tutor problems in section 12.1.
- 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

- Do the nano-quiz.
- Work through the design lab.

Before the beginning of your next software lab on May 6 or 7

- Do the on-line tutor problems in section 12.2.
- Submit written solutions to questions 2—9, 13, 14, and 15. All written work must conform to the homework guidelines on the web page.

Note: Exploration is due on Thursday May 8!!! That is the last legal due date.

Do `athrun 6.01 update` to get a directory `lab12` that contains the relevant files. In order to do this entire lab, it will also be important to have the latest version of SOAR. Athena machines and lab laptops will update automatically; to update your laptop, please download a new version of SOAR from the 6.01 software page.

Software Lab: State estimation part 1

In the next two software labs, we'll use basic probabilistic modeling to build a system that estimates the robot's pose, based on noisy sonar and odometry readings. We'll start by building up your intuition for these ideas in a simple simulated world, then we'll move on to using the real robots, next week.

This week we'll start by working with a 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

```
>>> import se
>>> 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 `make5ip` (which stands for 5 by 1, perfect). In your Python (not Idle) shell, type

```
>>> w = make5ip()
```

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`.

The state of the system is described with two integers, which are the discrete x and y coordinates of the robot's actual location in the world. The robot doesn't have access to this information though; all it gets to do is make an observation of the color of the room it is in. When the world is created, and after every step, it prints out the "belief" distribution over the array of possible robot locations. Then, it prints out what observation the robot actually makes of its world (if the observation function is probabilistic then the robot might observe "red" even if it is standing in a square that is white). Finally, it prompts you for an action that the robot should try to take. The simulator executes that action (using the motion model in the world, which may be stochastic), updates the hidden state of the world (the robot's location), and generates a new observation.