MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.01—Introduction to EECS I
Fall Semester, 2007

Assignment 1

Issued: Tuesday, Sept. 4

This handout contains:

- Pre-Lab activities to be done before Thursday, September 6 at 10 AM.
- General information about 6.01
- Robot Lab for Thursday, September 6
- Post-lab exercises due Tuesday, September 11 at 10 AM. Don't forget that some of the on-line Programming Tutor problems are also due before Tuesday lecture.

All pre- and post-lab exercises are to be written individually. Please see the collaboration policy on the web site for more information.

Pre-Lab Activities: To be done on or before Lab on Sept. 6

- Obtain a lab notebook (or just paper in a binder).
- Practice writing simple Python programs that manipulate lists and do arithmetic computations. The Online Programming Tutor for 6.01 (`http://sicp.csail.mit.edu/6.01/prog`) can be used as a starting point to learn the basics of Python.
- Go to the online tutor at `http://sicp.csail.mit.edu/6.01` or by clicking the On-Line Tutor button at the top of the class web page. Please fill in the very brief survey. Your username/password for the Online Programming Tutor should work.
- Install Python and IDLE on your personal computer (follow the instructions at `http://courses.csail.mit.edu/6.01/fall07/software/`) and do the simple exercises below. Or, you can use an Athena computer that has these installed. **Course staff will be available in 34-501 on Tuesday Sept. 4, 3:00–6:00 PM to help you with software installation.**
- Bring your laptop (if you have one) to class on Sept. 6. We'll guide you through additional software installation. If you don't have a laptop, that's ok, we have some in lab.
- Read this whole document.

Getting started

Schedule In general, the weekly schedule for 6.01 will be like this:

- Tuesday lecture from 10–11:30, followed by a software lab from 11:30 - 1:00. We will also hand out the homework for that week. The homework will include a part that is due on Thursday before lab and a part that is due the following Tuesday before lecture. The homework will have parts to write up and turn in, as well as parts to be done with the online tutor.
- Wednesday evening homework help session from 7–10pm. You are welcome to do the homework for Thursday on your own if you prefer, or in a self-organized study group. But we suggest that your time would be well spent if you do your pre-lab homework in the staffed Wednesday sessions. When programming, especially, it's easy to get lost in rat-holes and spend enormous amounts of time digging yourself out. Doing your work when there are staff members around can make things much easier.
- Thursday lab from 10:00–1:00. Each lab will end with a short quiz based on the pre-lab assignment as well as the material covered during the lab.
- Post-lab homework, due before the following Tuesday lecture.

Note that since the first meeting of the class this term will be on Thursday, there will be no Tuesday lecture on Sept. 4, no Wednesday homework help session on Sept. 5 and no quiz on Sept. 6.

Reading This is such a new way of looking at the material of intro EECS, there is no textbook. We will be handing out draft notes each week, and those will be your primary reading resource. Here are some additional resources for learning Python.

- Official Python tutorial: `http://http://docs.python.org/tut/tut.html`
- *How to Think Like a Computer Scientist: Learning with Python*, by Allen Downey, Jeffrey Elkner, Chris Meyers. This book assumes no programming experience. It is not a Python reference manual; it is a computer science text that uses Python as an example. `http://www.greenteapress.com/thinkpython/thinkCSpy.pdf`
- *Learning Python*, by David Ascher and Mark Lutz. This book also assumes very little/no programming experience, and is longer (not so great for learning Python in a hurry, but covers topics in great detail, so is good as a reference guide if you know what you're looking for). `http://proquest.safaribooksonline.com.libproxy.mit.edu/0596002815?tocview=true` (You need an MIT Certificate to view this one)
- *Learn Python in 10 Minutes*, by Poromenos. If you have very little time and a fair bit of programming experience in another language, this tutorial covers Python's syntax quickly. `http://www.poromenos.org/tutorials/python`
- Look at some of the other reference material on the class web site under "Resource Material" (under the "General Information" menu at the top of the web page).[1]

[1]If you use Windows with Internet Explorer as your web browser, you won't see the drop-down menu under "General Information". There's a "Resource Material" link at the bottom of the home page that you can follow instead.

Software

You should become familiar with using Python, (via IDLE or you can use Emacs Python mode), SoaR (to control a real or simulated robot). You will be running these on the lab laptops and you should be able to run it on your personal computer or Athena. The instructions below are for the lab laptops. Feel free to ask us for help to get you started on your own machines. You will also be using the on-line Tutor system to do some of your weekly homework.

Starting and using IDLE

IDLE (`http://www.python.org/idle/`)is a simple way to edit and run Python programs. To start it on one of the lab laptops, go to a shell and type

```
> idle
```

Python Shell The Python shell acts a little bit like a calculator. You type in expressions, Python evaluates them, and then prints the result. So, if you type

```
>>> 4 + 4
```

It will print out `8`. Play with the shell a little bit.

Edit a file You'll want to use the shell to test things out, but not to write your programs. If you want to start defining procedures, you should open a new file (choose `New Window` from the `File` menu) and write your definitions in there. So, start by making a file containing this definition:

```
a = 'hi'
b = 7
def f(x):
    return x + 1
```

Now, save the file and choose `Run Module` from IDLE's `Run` menu. It will act as if you have typed the text of your file into the shell. If there was something obviously syntactically wrong with your file (parentheses not closed, for example), IDLE will tell you about it and highlight the point in your file where the problem is. Otherwise, the shell will print out something like

```
>>> ================================ RESTART ================================
>>>
```

And now you can ask it to evaluate expressions, including things you've defined in your file.

- Use the Python shell to compute `f(f(f(b)))`.
- What happens if you do `f(a)`?

To do exercises, use the shell for experimentation, and write new procedure definitions in your file. Whenever you change your file, you'll need to do `Run Module` again.

Archive your file One key way to preserve your files between labs is to keep them on your Athena account. We have provided you a way of connecting to your Athena account from the lab laptops and to download released software there (see the separate instructions available on the course Web site). If there are any problems with this, you can also keep your files, for the duration of a lab, on the lab laptops.

When you do keep your files on a lab laptop (and not using your Athena account), you should always remember to mail or FTP your files back to yourselves. We don't guarantee that you'll always get the same laptop or that any files that you leave on it will remain there from week to week.

An easy way to keep your files safe is to use a webmail service. You may have a service such as Gmail or Hotmail that you can use. We'll explain the process using MIT webmail.

To use MIT webmail, first, you go to: `http://webmail.mit.edu`. To log in, type your Athena account user name and its password. You can compose an email to your email address (`user@mit.edu`, where `user` is your account name), and attach your files. When you're using any computer running some version of Unix, you can use the `scp` command. General usage of `scp` (when copying the file to the remote computer) is as follows.

```
scp file user@machine:file
```

For example, if you want to copy the file `lab1.py` to the directory `Courses/6.01` on your Athena account `user1`, you type:

```
scp lab1.py user1@athena.dialup.mit.edu:Courses/6.01/lab1.py
```

Note that the directory specified in the destination (`Courses/6.01`) must have been created before copying a file into it. Also note that `scp` command automatically overwrites the file you specified as the destination, without asking your permission. So, you have to be careful not to overwrite your important files using `scp`.

When you want to copy the file from Athena to a lab laptop, you can simply switch the destination and the source for `scp`:

```
scp user1@athena.dialup.mit.edu:Courses/6.01/lab1.py lab1.py
```

SoaR

If you're using a lab laptop, start up SoaR by typing

```
> SoaR
```

in a shell. If you're using your own laptop, follow the instructions for the OS you're using. Load the simulator (press *Simulator* (slider icon)) and pick one of the worlds (for instance, `tutorial.py`), and drive the robot around using the joystick (press *Joystick* (joystick icon) and then *Start* (right triangle icon)).

Now, we'll use a program to control the simulated robot. Select *Brain* (light-bulb icpn) and then pick `testBrain.py`. Then hit *Start*. The robot should zoom around the simulated world, avoiding obstacles.

Using the online tutor

Go to the online tutor at `http://sicp.csail.mit.edu/6.01` or by clicking the On-Line Tutor button at the top of the class web page. We will copy your account information from the Python programming Tutor to this version of the Tutor. If you did not have an account on the Python tutor, then register by clicking the register link and filling in the form; you need a valid MIT address. The system will mail you a password. Go back to the on-line tutor log-in page and log-in. Click the Problem Set button to see assignments for the week. **Note that you need to click the buttons (don't hit Enter or click on the words).**

Robot Lab for Thursday Sept. 6

This lab is organized into three sections, each of which contains some checkpoint questions. Be sure to show your work to a staff member at the end of each checkpoint and be ready to explain your answers. There are some extra suggestions for explorations to do if you have more time.

Basics

Let's start by making the robot move. You can drive the robot around by hand, using the joystick. Plug your robot's serial cable into the robot and into your laptop, and turn the robot on. Now start SoaR by typing

```
> SoaR
```

then press *Pioneer* (gear icon), and *Joystick* (joystick icon) and *Start* (right triangle icon). Click and drag in the resulting Joystick window to control the robot. This works with the simulator as well as with the real robot.

Our robots have *sonar* sensors for measuring the distance to obstacles in different directions and *odometry* sensors for measuring how far the robot has moved. It's easy to understand idealized sensors, but their actual behavior is more complicated. The goals of this lab are: to become familiar with reading the robot's sensors, and to get the robot to move. We also would like for you to come to appreciate that real-world sensors and effectors don't usually behave in the idealized way you might wish they would.

Brains for Beginners

SoaR interacts with the robot (real or simulated) via a *brain program.*

A brain has the following structure:

```
def setup():
    print "Starting"
def step():
    print "Hello robot!"
```

Your brains should have this form, but you don't need to worry right now about anything except the `step` function. This function will be called by SoaR about 10-20 times per second. The above brain, if run, will start by printing `Starting`, and then just keep printing `Hello robot!` in the SoaR message window until you stop it.

Try saving this code as `reallysimplebrain.py` in the `brains` subfolder inside the `SoaR` folder and running it by opening the simulator in SoaR, choosing *Brain* (light-bulb icon) instead of *Joystick*, selecting `reallysimplebrain.py`, and finally hitting *Start.*

As you can see, this brain is boring. Don't worry if you click the stop button and it keeps printing for a while. All that text has been stored up somewhere waiting to be printed.

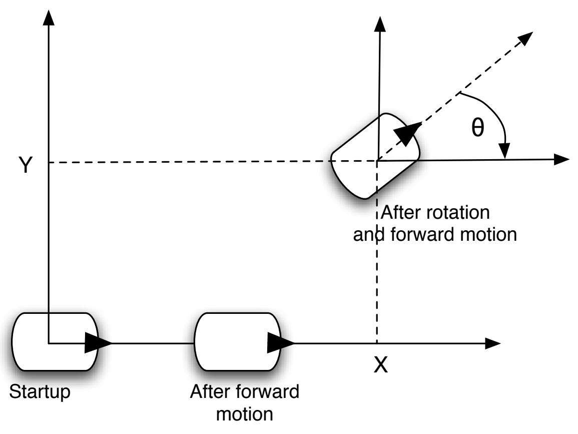

Figure 1: Global odometry reference frame

Sonar

The robot has a set of eight ultrasonic transducers (familiarly known as sonars). They send out a pulse of sound and then "listen" for a reflection. The raw sensors return time-of-flight values, which is how long it took the sound to bounce back. The robot's processor converts these into distance estimates.

The command `sonarDistances()` returns a list containing the robot's current sonar readings, all in meters. Try saving the following code (as say, `"simplebrain.py"`) and running it on both the simulator and the real robot:

```
def step():
    print "sonarDistances:", sonarDistances()
```

This brain prints out the current sonar readings on every step. Play with putting something in front of different sensors and seeing how the values change. Which value corresponds to which sensor?

Odometry

The robot has *shaft encoders* on each of its drive wheels that count (fractions of) rotations of the wheels. The robot processor uses these encoder counts to update an estimated *pose* (a term that means both position and orientation) of the robot in some global reference frame. You can access the robot's pose with the function `pose()` (which returns a tuple of three values containing the robot's x position, y position, and orientation in radians). This information is generally unreliable since the wheels slip on the floor, so counting wheel revolutions is not perfect.

Figure 1 shows the reference frame in which the robot's pose is reported. When the robot is turned on, the frame is initialized to have its origin at the robot's center and the positive x axis pointing out the front of the robot. You can now think of that frame as being painted on the ground; as you move or rotate the robot, it keeps reporting its pose in that original frame.

Motion

You can move the robot using the function `motorOutput(`*fvel*`,`*rvel*`)`, where *fvel* (in meters/sec) is the forward velocity of the robot and *rvel* (in radians/sec) is a rotational velocity. Experiment with this function a bit by adding it to a brain, just to get a feeling for how it works (but be careful not to crash the robot into anything—a velocity of 1 is *fast*, so use something a lot less). `motorOutput(0,0)` will ask the robot to stop, but it may continue to move a little bit. Why?

A well-formed brain should execute *exactly* one call to `motorOutput` on every execution of step. Of course, the brain can have multiple calls to `motorOutput` in it, embedded in `if` statements, such that only one is actually executed. The idea is that the brain will decide, on each step, how fast the robot should be moving.

Simple Brains

You should write some simple brain programs on the simulator.

Question 1. Write a brain that will move the robot forward.

Question 2. Write a brain that will make the robot rotate.

Question 3. Write a brain that will print out the values of the front two sonar sensors.
Hint: Read the section on Fancier Output Formatting in the Python Tutorial to see how you control the number of decimals in printing.

Question 4. Write a brain that will print out four values, each of which is the average of two adjacent sonar sensors.

Question 5. Write a brain that prints the robot's pose inside the `step` function while moving forward slowly. What is the robot's pose when your program starts? Does it get reset when the brain is stopped and restarted? Does it get reset when the brain is reloaded? Try it on the real robot. Does it behave the same way?

Checkpoint (should be completed by 11:30 AM)

Demonstrate your programs running on the simulator to your LA. Be ready to explain how they work and why you made them the way you did.

Robot behavior

Now we'll explore integrating sonar and motion to do some interesting things on the real robots. You will want to debug these in the simulator initially.

Question 6. Write a brain that will move the robot forward until it is 30cm from an obstacle in front of it.

Question 7. Augment your first behavior so the robot backs up if something gets closer than 30cm.

Question 8. Write a brain that will try to move along, parallel to a wall next to it. Try starting at different distances and angles to the wall.

Checkpoint (should be completed by 12:15 PM)

Demonstrate your programs running on the real robots to your LA. Be ready to explain how they work and why you made them the way you did.

Sonar Experiments

In this section, we'll experiment systematically with the sonar sensors to see how they behave. You can think of the transducer as emitting a cone-shaped beam of sound waves, which are reflected off surfaces back to the detector. The physical details of the shape of the cone and the way the reflections work imply that sonar readings are often not what one would expect. Take notes in your lab notebook.

Question 9. What happens when something is very close to the sonar?

Question 10. What happens when the closest thing is very far away?

Question 11. How do the results vary as a function of the angle between the transducer and the surface that it is pointed toward?

Question 12. How do the results vary as a function of the material of the surface? Try bubble wrap: what's going on?

Checkpoint (should be completed by 12:45PM)

Demonstrate your experiments to your LA. Be ready to explain your results.

Concepts covered in this assignment

Here are the important points covered in this assignment:

- You got experience with a *transducer* organization for building embedded programs, programs that interact in real-time with a world.
- You began your experience with a real robot system and began to understand that real sensors and real devices can be quite different from the simple models that we often have of them. We will need to (a) build better models and (b) devise strategies for control that take can cope with this less-than-ideal behavior.

You also got some programming practice with Python.

The Explorations section is not required. They're things that you can work on if you're interested. If you hand in a page or two describing some work you did on this, you are eligible for an extra half a point (10%) on the lab.

Explorations: Doorway

If you have time left over, write a program that can navigate the robot through a small opening in a wall in front of it, based on the sonar readings. It's fine for the robot to be pointed fairly directly at the opening. Start with a really big opening and then gradually make it smaller. See how small you can go! Don't worry if you have trouble getting this to work. It's hard to do it robustly, and we'll spend a lot of time in the coming weeks studying strategies for solving problems like this.

Post-Lab Writeup and Exercises: Due before lecture on Sept. 11

All post-lab hand-ins should be written in clear English sentences and paragraphs. We expect at least a couple of sentences or a paragraph in answer to each of the numbered questions in this lab handout.

We also want the code you wrote. When you write up answers to programming problems in this course, *don't* just simply print out the code file and turn it in. Especially, don't turn in long sections of code that we've given you. Turn in your own code, with examples showing how it runs, and explanations of what you wrote and why.

We also expect at least a couple of sentences or a paragraph in answer to the reflection questions below. We're in interested in an explanation of your thinking, as well as the answer.

Reflection on lab results

Question 13. How well would your robot behaviors have worked if the step function were called once per second, rather than 10 times per second? What if it were called once per minute?

Question 14. If the robot is at current pose $\langle x, y, \theta \rangle$ in a global reference frame and it gets a sonar value on the i^{th} sonar of h, where is the point that the sonar hit, in the global frame? What do you need to know about the sonars on the robot to answer this question? Assume answers for any parameters you don't know.

Question 15. Draw a picture of the coverage of the sonars. Are there areas where a thin pole might not be seen?

Question 16. Why do you think putting bubble-wrap on objects make the sonar readings more reliable?