MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.081—Introduction to EECS I
Fall Semester, 2006

Assignment for Week 1

- Issued: Tuesday, Sept. 5
- Includes homework due before lab on Thursday, Sept. 7
- Also includes post-lab homework due at the beginning of class on Tuesday, Sept. 14
- Readings:
  - Here is a list of Python tutorials. Pick one you like, glance at it to get a feel for what Python programs look like, and so that you can return to it later to use as a reference.
    - *Python for Newbies*, by David Borowitz. This tutorial goes through the basics at a good clip and only assumes a bit of programming experience.
      `http://npt.cc.rsu.ru/user/wanderer/ODP/Python_for_Newbies.htm`
    - *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`
    - *How to Think Like a Computer Scientist: Learning with Python*, by Allen Downey, Jeffrey Elkner, Chris Meyers. This book assumes no programming experience, and is quite long. `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 even 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)
  - 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]

Getting started

Yes, it's a dirty trick to distribute an assignment before the first class.

What we're actually hoping is that you'll take a quick look at this handout and bring it (together with a laptop, if you have one) to our **welcoming party and tutorial session** (free food!) on Wednesday, Sept. 6, from 6–9 PM in the Star conference room at the Stata Center, 32-D463.

We'll use this time to get acquainted, help you install software, register you for the online tutor, guide you through the first steps of Python and robot programming, and prepare for Thursday's lab.

In general, the weekly schedule for 6.081 will be like this:

[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.

- Tuesday lecture from 1–2:30 in 32-124, followed by posting of the homework for the coming 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 7-10. Attendance at these sessions is optional. 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 will be much better 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 afternoon lab. Each lab will begin with a short quiz based on the pre-lab assignment and other material previously covered.
- Post-lab homework, due before the following Tuesday lecture.

1. To do before lab on Sept. 7

Obtain a lab notebook (or just paper in a binder).

The rest of the pre-lab problems/instructions can be done at the party:

Install Python, SoaR, and the editor (GNU/Emacs or IDLE) on your computer (If you really can't make it, follow the instructions at http://courses.csail.mit.edu/6.01/fall06/software/), or find a way to get access to a computer that has these installed.

1.1. Setting up

Start Python and the editor, and evaluate some simple expressions, just to make sure things are working. For example: type `2+4` and press enter, or type `print "hello world!"` and press enter.

Start up SoaR (by double-clicking on `soar.py` in the SoaR directory, or by typing 'python soar.py' in the SoaR directory in a terminal window) and glance at the manual to see what's there. (Not a lot.) Load the simulator (press *Simulator* and one of the worlds (for instance, `tutorial.py`), and drive the robot around using the joystick (press *joystick* and then *start*).

As a general warning, SoaR will crash at some point (probably fairly often). If it does, close down all of its windows (if it's frozen in linux, type 'killall soar.py'), restart the robot if you're using the actual robot, and run it again.

1.2. Using the online tutor

Register for the online tutor at `http://sicp.csail.mit.edu/6.081` and do the exercises that are due for September 7.

1.3. Python programming

Use Python and the text editor (Emacs or Idle) to create a file that contains the following procedure definition:

```
def p(x):
    a=[]
    t=2
    while x >= t:
        if x%t == 0:
            a = a+[t]
            x = x/t
        else:
            t = t+1
    return a
```

(a) Run the procedure with various positive integer values for x. What result does this procedure compute?

(b) The code we've written betrays a *horrible* programming practice that you should *never* succumb to in your own code: It uses meaningless identifiers like p, a, x, and t, rather than names that give the reader an idea of what the program does and how it works. Redefine the procedure to use better identifiers.

(c) Try the procedure with some large numbers: 54321, 987654321, 987654321987654321, $10^{15}+1$, $10^{14}+1$. Does it work with all of these inputs? What can you say about how long the procedure takes to return its result as a function of the input?

(d) Suggest some ways to change the procedure so that it gets its result more quickly. Try to code these changes. Can you make a version that works with all of the above test cases? **Don't worry if you don't succeed.** We'll return to this next week.

2. To do in lab on Feb. 7—Getting to know your robot

Today's lab (and every Thursday lab) will begin with a mini-quiz that is based on the homework due for today, and on material covered in previous weeks.

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 (aim to finish checkpoints around the end of each hour; the last hour is leeway time) and be ready to explain your answers. There are some extra suggestions for explorations to do if you have more time.

2.1 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 double-click on `soar.py`, press `Pioneer`, and `Joystick` and `Start`. Click and drag in the resulting Joystick window to control the robot. This works with the simulator as well as with the real robot.

(If it doesn't work with the real robot, have you enabled the motors yet? No? Have you read the *Don't Panic manual*? Why not?)

Our robot has *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.

2.2 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 times per second. The above brain, if run, will just keep printing `Hello robot` in the SoaR message window.

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` 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.

If you want to use a variable to store a value between invocations of the `step` function, you can extend your brain to something like this:

```
#code that is called once at startup
def setup():
    print "start"
    robot.x = 0

#code that is called about 10 times a second
def step():
    robot.x = robot.x + 1
    print "Step ", robot.x
```

If you include a function called `setup`, it will be called once, when the brain is loaded. It's a good place to set things up! In this program, `setup` makes a new variable that will persist over time, called `robot.x`. This brain will print `start` when it is first loaded, and then print `Step 1`, `Step 2`, etc, when it is run.

Try writing a brain that prints "Hello robot" once every 10 steps.

Sonar

When you use the simulator, it shows lines protruding from the robot at various points: these are meant to be the signal paths of the distance sensors. The distance sensors in the simulator work perfectly, in that they always measure the exact distance down that ray to the first surface they contact.

The real 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 describing the robot's current sonar readings, all in meters. Also, `sonarInfo` is a static list of tuples where each tuple describes one sonar in the form `(x,y,th)` where `x` and `y` (units in meters) are the location of the sonar on the robot with respect to its center and `th` is the direction it points (units in radians, zero is straight ahead).

Try saving the following code (as say, `"simplebrain.py"`) and running it on both the simulator and the real robot:

```
def setup():
    robot.count = 0

def step():
    print "sonarDistances:"
    distances = sonarDistances()

    #get rid of trailing zeros, so the sonar values are easier to see
    prettydistances = [str(x) for x in distances]
    print prettydistances

    print "pose", pose()

    print robot.count
    robot.count+=1

    print "hit enter to continue"
    raw_input()
```

This brain prints out the current sonar readings and the pose of the robot, then waits for a keyboard press to continue (so that if you hit `start`, the readings don't fly past too quickly to read).

Motion

You can move the robot using the function `motorOutput(`*fvel*`,`*rvel*`)`, where *fvel* is the forward velocity of the robot and *rvel* 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 make the robot stop. Be warned that if you tell the robot to move forward in a step, with no command to stop it, the robot will continue to move forward until it is explicitly told to stop.

A well-formed brain should have *exactly* one call to `motorOutput` on every step. The idea is that the brain will decide, on each step, how fast the robot should be moving.

Checkpoint (should be completed by 2:15 PM)

Demonstrate that you can

- move the robot forward using a brain
- make the robot rotate in both directions using a brain
- read the sonar sensors

2.2 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.

By calling the function `sonarDistances()[0]` (`sonarDistances()` is a function call that returns a list; `sonarDistances()[0]` is the first element of that list) and putting different surfaces in front of that transducer, experiment with a single sonar transducer to discover what its behavior is.

- What happens when something is very close to the sonar?
- What happens when the closest thing is very far away?
- How do the results vary as a function of the angle between the transducer and the surface that it is pointed toward?
- How do the results vary as a function of the material of the surface? Try bubble wrap: what's going on?
- How wide is the beam?

You are eventually going to need to program this robot to decide whether it can move forward some fixed distance without running into something, and to drive through doors. Think about what kind of data would be useful for understanding how to solve the problem. Make the two graphs requested for the checkpoint and explain why they would be useful.

Use your notebooks to record the data you collected (and will collect, throughout this lab). You may need the data again later in the semester.

Checkpoint (by 3:30 PM)

Show two graphs. It's okay just to sketch them by hand.

- Measured versus actual distances to a surface.
- Measured distances versus angle between the transducer and the surface.

Try these for at least two different surface types (shiny, bubble-wrap, human legs). Be ready to explain your results.

Exploration if you have time: Do some more experiments. You might characterize the beam width carefully, or look at the variability of measurements in a fixed configuration.

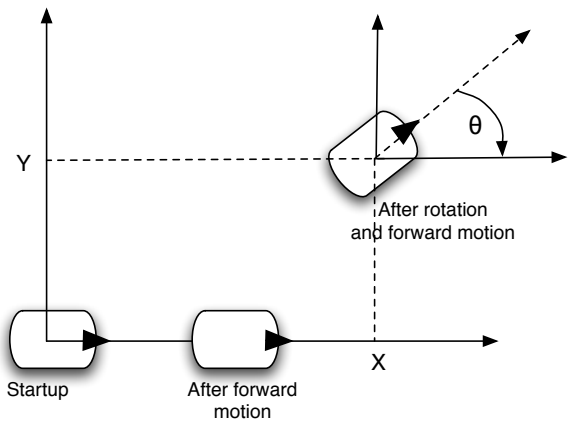

Figure 1: Global odometry reference frame

2.4 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), as demonstrated in `simplebrain.py` above.

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.

Explore the odometry and motion using brains:

- Write a simple 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?
- Write a brain that will move the robot forward one meter (according to the odometry) regardless of the current rotation. Note that if the robot is not at angle 0, the distance moved depends on both x and y coordinates.
- Extend the brain so that it can also do rotations of 90 degrees.
- Measure the accuracy of the odometry by repeatedly moving the robot and comparing the actual change in pose (as you measure it) to the estimated one. Is odometry more reliable for straight-line motion or for pure rotations? Draw the graphs requested in the checkpoint.

Checkpoint (by 5 PM)

Show a staff member:

- Your brain for moving a fixed distance or angle
- Plot showing commanded versus actual forward motions
- Plot showing commanded versus actual rotations

Be ready to discuss the odometry results.

2.5 Putting it together

If you have time after completing the above checkpoints:

Now we'll explore integrating sonar, odometry, and motion to do some interesting things.

- Write a brain that will move the robot forward until it is 30cm from an obstacle in front of it. Use what you learned about sonar behavior in the first section of the lab.
- 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.

3. To do before Sep. 12

Work on the following problem after Thursday's lab. Turn in the written parts at the beginning of lecture on Sep. 12.

3.1. Reflection on lab results

- Suppose you program the robot to circumnavigate a one-meter square—based on the odometry. Given your estimates of the odometry accuracy, how far do you think the robot might end up from its starting point? What if it were a one-hundred-meter square?
- What kinds of terrain would be particularly good or bad for the robot's odometry?
- Imagine that the robot is driving forward toward a wall with a door in it. When might it have trouble seeing the door?
- Imagine that the robot is driving toward a thin pole. When might it have trouble seeing the pole? What can you say about the accuracy with which it could determine the pole's position relative to the robot?

- Suppose your robot is actually 10 meters from a wall. It moves forward 1 meter, according to the odometry. The sonar reading now says it is 8.5 meters from the wall. What might you conclude about the robot's actual position with respect to the wall?
- Can you think of any methods for combining sonar and odometry for dealing with thin poles or surfaces at difficult angles? Could actively moving the robot help solve these problems?

Exploration If the robot is currently at pose $\langle x, y, \theta \rangle$ and the wheels have radius r, and the distance between the wheels is d, what is the robot's pose after the left wheel has made one complete rotation and the right wheel has made two? (No extra credit. Just our admiration.)

3.2. Exercises with the online tutor for Sep. 12

Use the online tutor to complete the problems due for Sep. 12.