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

Assignment 4, Issued: Tuesday, Sept. 25

To do this week

...in Tuesday software lab

1. Start writing code and test cases for the numbered questions in the software lab. Paste all your code, including your test cases, into the box provided in the "Software Lab" (Part 4.1) problem on the on-line Tutor. This will not be graded.

...before the start of lab on Thursday

1. Read the lecture notes.
2. Do the on-line Tutor problems for week 4 that are due on Thursday (Part 4.2).
3. Read through the entire description of Thursday's lab.

...in Thursday robot lab

1. Answer the numbered questions in the robot lab and demonstrate them to your LA.
2. Do the nanoquiz; it will be based on the material in the lecture notes and the on-line Tutor problems due on Thursday.

...before the start of lecture next Tuesday

1. Do the lab writeup, providing written answers (including code and test cases) for **every** numbered question in this handout.

On Athena or the lab laptops make sure you execute:

```
athrun 6.01 update
```

so that you can get the `Desktop/6.01/lab4` directory which has the files mentioned in this handout.

- You need the file `polynomial.pyc` for the software lab.
- You need the file `diffeq.py` for the software lab.
- You need the file `soar-graph.py` for the software lab.

During software lab, if you are using your own laptop, download the files from the course Web site (on the Calendar page). Make sure you are using Python version 2.5, as we are giving you a "compiled" version of the polynomial class.

Tuesday Software Lab 1: Solving second-order difference equations

For this lab, you will be writing a Python program that solves arbitrary second-order homogeneous linear difference equations analytically, by computing natural frequencies. Your program should take as inputs the initial values $x(0)$ and $x(1)$ as well as coefficients a_1 and a_2 for the difference equation in the form

$$y(n) = a_1 y(n-1) + a_2 y(n-2).$$

Your program should print out the solution to the difference equation and also return a procedure that, when called with n, returns $y(n)$. For example, if the difference equation is

$$y(n) = 2y(n-1) - 2y(n-2),$$

and the initial conditions are $y(0) = 0$, $y(1) = 1$, your program should print something equivalent to

```
y(n) = (2.7755575e-017+0.5j)*(1-1j)**n  + (-2.7755575e-017-0.5j)*(1+1j)**n
magnitude of natural frequencies: 1.4142135623730951, 1.4142135623730951
```

and your program should *also* return a function which can be used to evaluate $y(n)$ for any n. **NOTE:** you can assume that the natural frequencies are distinct (but your program should indicate an error when the two natural frequencies are identical). The special case of what to do with repeated natural frequencies is studied in more advanced courses.

Polynomial manipulation

To save time, you should use our implementation of the polynomial manipulation program from the recent post-lab exercises and edit *diffeq.py* to create your second-order difference equation solver. Please be sure *diffeq.py* and *polynomial.pyc* are in the same directory as your difference equation solver. Executing *diffeq.py* in IDLE will import our version of the polynomial manipulation routines implemented in the class Polynomial. You can see the class interface by typing

```
help(Polynomial)
```

at the IDLE command line. Note that the Polynomial class has functions to add, multiply, print and find the roots of polynomials. The following code fragment

```
p = Polynomial([1, -6, 9])
>>> p.roots()
[(3+0j), (3-0j)]
```

generates an instance of the polynomial

$$x^2 - 6x + 9 = 0 \ ,$$

and then computes the polynomial's roots. Our implementation only computes roots correctly for polynomials up to third order (cubics).

Checkpoint: 11:45

- You should understand the polynomial class.

A note on complex numbers

For this problem, your procedure will sometimes need to compute z^n where z is a complex number. Python understands complex numbers to some extent, but you have to write them in the form

$$a + bj$$

where `a` is the real part and `b` is the imaginary part. Note that Python uses the convention that $j = \sqrt{-1}$. As an EECS student, you will have to learn to accept the fact that the variable i must be reserved for electrical current.

You can't raise a negative real number to a fractional power, but you can do so with a complex number. So, for example, the Python command

```
>>> (-4)**0.5
```

will produce an error, but the Python command

```
>>> (-4+0j)**0.5
```

produces `(1.2246063538223773e-016+2j)`. (Why the very small real part?)

Note: use `y**0.5` to compute the square root of a number. The python `sqrt` function does not understand complex numbers. You can get the real and imaginary parts of a complex number `x` with `x.real` and `x.imag`; also, `abs(x)` will return its magnitude.

It will be important to ensure that the function returned by your difference equation solver only returns real (not complex) values. Be sure you understand why, mathematically, the output should always be real if the coefficients and initial values are real.

Demonstrate that your program works

Question 1. Describe how your program works.

Question 2. Test your program by solving the difference equation

$$y(n) = 2y(n-1) - 2y(n-2),$$

with the initial conditions $y(0) = 0$, $y(1) = 1$.

Question 3. Demonstrate that your program returns the correct function. How did you verify your program was correct?

Checkpoint: 12:30 PM

- You should have a working difference equation solving program.

You can also test your program by plotting the values the program generates. Here are the steps:

1. Start SoaR (just type `SoaR` in the terminal; it's best to quit Idle before you do this)
2. Click on the editor window to get an Idle window from inside SoaR
3. Open the file `soar-graph.py` which has the following commands:

   ```
   from diffeq import *
   g1 = GraphingWindow(400, 400, 0, 20, -100, 100, "diffeq")
   f = solveDiffEq([0, 1], [2, -2])
   g1.graphDiscrete(f, 'blue')
   ```

   The first in the above sequence of commands generates a a graphing window that is 400 by 400 pixels, with x coordinates ranging from 0 to 20 and y coordinates from -100 to $+100$, and gives the graph the window title `diffeq`). The second command in the sequence calls a difference equation solver that returns a procedure `f`. The third command causes a blue graph to be generated in the graphing window. The function `graphDiscrete` will call the function `f` with integer arguments from 0 to 20 and plot the result.

   PLEASE NOTE: Your routine for computing difference equation solutions is likely to return complex numbers with very small imaginary parts (due to inexact cancellation). You should therefore force your function to return a real value. If you pass a complex number to graphDiscrete, very strange error messages are generated.

4. Save the file and pick **Run in interpreter** from the SoaR menu. You should see a new graphing window pop up; you can use the controls to rescale the window if you guessed the scales wrong. If the window doesn't pop up, look at the terminal you started SoaR from to see if there are error messages.

Checkpoint: 12:45 PM

- You should be able to plot solutions to difference equations.
- You should understand how natural frequencies are related to the time series plot.

Question 4. How are the natural frequencies are related to the time series plot?

Question 5. Use your program to find a difference equation whose solution oscillates and decays. What must be true about the difference equation's natural frequencies?

Question 6. Use your program to find a difference equation whose solution oscillates and grows. What must be true about the difference equation's natural frequencies?

Go to the on-line Tutor at `http://sicp.csail.mit.edu/6.01/fall07`, choose PS4, and paste the code that you wrote during lab, including your test cases, into the box provided in the "Software Lab" problem. Do this even if you have not finished everything. Your completed answers to these questions are to be handed in with the rest of your writeup for the on Tuesday, October 2nd. **Make sure that you include the test cases and results that you used to conclude that your code was working.**

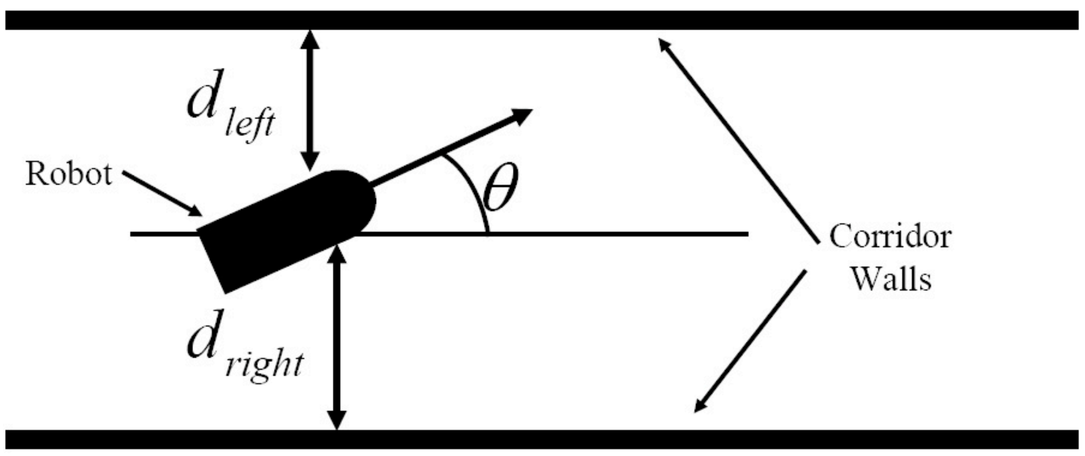

Figure 1: Robot in corridor.

Thursday Design Lab: Difference Equations and Dynamic Feedback Control

Up to this point in the course, you have developed progressively more sophisticated approaches for measuring the robot's environment, as well as progressively more intelligent approaches to controlling the robot's actions based on those measurements. In those labs, the primary goal was to have you use the robot to investigate ideas in software engineering and to expose you to a few subtleties involved in sensing and control. In this lab, you will be using measurements to control the robot in a feedback system, and will be investigating a situation in which detailed analysis is needed to achieve a desired performance. More specifically, we are asking you to investigate how to control the robot so that it drives down the center of a narrow corridor at a constant forward velocity. Such a task is similar to what an automobile driver does when keeping a car on the road. We will suggest a simple feedback scheme to keep the robot in the center of corridor, and ask you to develop and analyze a mathematical model based on difference equations that explains the behavior of the suggested simple feedback system.

In this lab you will be controlling the robot to drive down a narrow corridor as shown in Figure 1.[1] Notice that in the figure, we have denoted the direction of the robot's forward motion, the distance to the left wall, d_{left}, the distance to the right wall, d_{right}, and the angle of the robot with respect to the parallel walls, θ.

Consider the problem of trying to steer the robot down the center of the corridor while keeping the robot moving forward with a constant velocity. For example, if the robot is in the center of the corridor and pointing in a direction parallel to the walls, one could keep the robot moving forward down the center of the corridor with the command

```
motorOutput(0.1, 0.0)
```

which will keep the robot moving forward 0.1 meters per second. If the robot is too close to the right wall, one could issue the command

[1]The optical sensing strategy and software used in this lab was developed by Karim Liman-Tinguiri, a former 6.01 student.

```
motorOutput(0.1, 0.2)
```

which will keep the robot moving 0.1 meters per second but will cause the robot to rotate to the left (increasing θ). If the robot is too close to the left wall, one could issue the command

```
motorOutput(0.1, -0.2)
```

which will keep the robot moving forward at 0.1 meters per second but will cause the robot to rotate to the right.

The above observations suggest a very simple feedback system that keeps the robot moving forward at 0.1 meters per second and tries to keep the robot in the center of the corridor. One could set the forward velocity in the `motorOutput` command to 0.1 and set the rate of rotation in the `motorOutput` command to be proportional to the distance from the center of the corridor.

Measuring Distances

In previous labs, you measured distances to objects using the robot's built-in sonars. The sonars are too slow and noisy to be effective in the kind of dynamic control system being investigated in this lab. Instead, you will be using a set of four optical sensors mounted in a diamond configuration on the top of the robot (near the robot's center of rotation). There are four sensors so that triangulation can be used to determine the shortest distance from the robot center to each of the corridor walls, even if the robot is pointed in a direction that is not perfectly parallel to the wall.

The four optical sensors encode distance information as analog voltages, and these analog voltages are converted to digital numbers that are transmitted to your laptop via a USB connection. A National Instruments digital-to-analog converter (NIDAQ) (the white box on mounted using velcro on top of the robot) performs the conversion, and the NIDAQ is managed by running a server program on your laptop. This server program makes the converted analog voltage data available to Python programs.

In order to use the optical sensors, open a terminal window and change to the `lab4` directory. Then make sure the USB cable from the NIDAQ is plugged into the USB port of your laptop (if so, the green light on the NIDAQ will be blinking). In the terminal window, type

```
./NIDAQserver
```

After a few seconds, the terminal screen will start filling with numbers. Minimize this terminal window, but do **NOT** kill the window.

Open a new terminal window and start the virtual oscilloscope program by typing

```
python v_oscillo.py
```

An oscilloscope window should appear on your laptop screen, and the first channel (Ai0) from the NIDAQ (which is connected to the first optical sensor) should be plotted in the oscilloscope window. Try blocking one of the optical sensors and notice what happens in the oscilloscope window. To show additional NIDAQ channels (corresponding to other optical sensors), right click on the display and toggle the desired channels. You can show multiple channels on the oscilloscope. If the oscilloscope does not seem to be working, close the oscilloscope window, unplug the NIDAQ USB connection, wait three seconds, replug the USB connection and restart the server and oscilloscope. Please ensure that all four distance sensors are functional by waving an object in front of each of them in

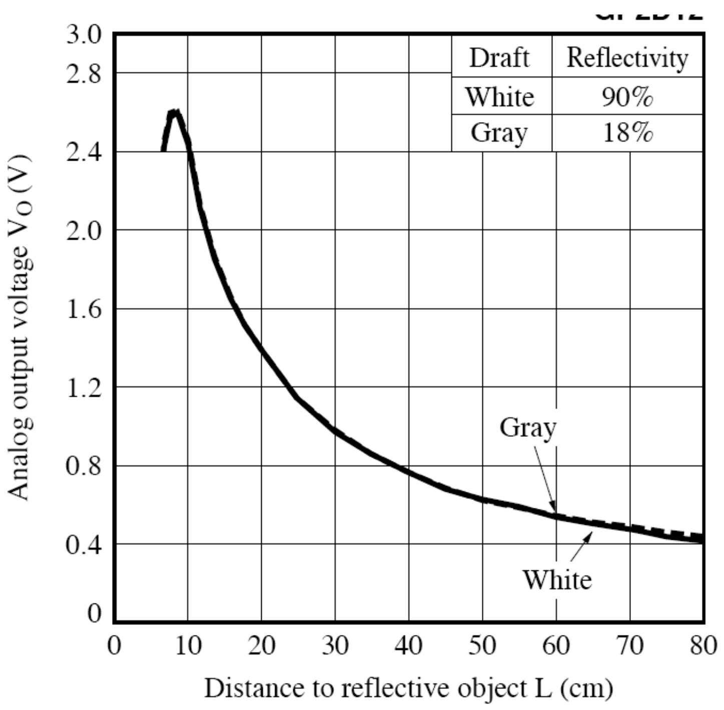

Figure 2: Optical range sensor voltage as a function of distance, from Sharp Corporation's GP2D12/GP2D15 data sheet.

turn. You should observe a spike in voltage readings for that channel. In case you're interested, figure 2 contains a plot of the nominal voltage readings for these sensors, as a function of distance to an obstacle. Note that if you disconnect your laptop from the NIDAQ, you'll have to restart the server, with

```
./NIDAQserver
```

Checkpoint: 10:30

- Demonstrate that your sensors are working.

Trying the robot control system

While still in the `lab4` directory, shut down the virtual oscilloscope, start SoaR and connect to the robot. Make sure to connect the serial cable to both the laptop and the robot (both the USB cable to the NIDAQ and the serial cable to the robot are needed for this lab).

Experiment with the above dynamic feedback control scheme by modifying the SoaR brain we have provided, `feedback_brain.py`. Note that we have provided a function `GetLR`, which returns the distances from the center of the optical sensor mount to the left and right wall, under the assumption that both of the optical sensors on a given side of the robot are "seeing" the same straight wall. Converting the voltages from the sensors to distances requires the use of the function plotted in figure 2, followed by some geometric calculations to recover the shortest distance between wall and robot from two distance readings on a given side of the robot. Writing such a conversion procedure is an interesting problem, but we have provided the program `GetLR` in the `feedback_brain.py` that returns the needed distances in the interest of time.

Please change the program in `feedback_brain.py` so that the rotational velocity is proportional to the robot's signed distance from the corridor center. Try your brain on your robot in one of the narrow corridors we have set up in the lab. Experiment with different constants of proportionality (also known as gains) in your feedback system, and try starting the robot both near and far from the center of the corridor.

PLEASE NOTE: It is very easy to accidently block one of the optical sensors with either of the two cables connecting the robot and the NIDAQ to the laptop. Typically, a single sensor will be blocked, and then very odd things happen. This problem is best avoided by carrying the laptop behind the robot and making sure the cables do not cross the line of sight of the optical sensors.

Checkpoint:11:30

- Demonstrate your robot moving in the corridor with several gains and initial positions.

Question 7. What is the effect on robot behavior of changing the feedback gain and the initial position? Try several gains and several initial positions for the robot.

Difference Equation Derivation

When the SoaR system is communicating with the robot and running your `brain`, it executes the step function many times every second. If you wrote your brain correctly, this means that many times every second, your brain is taking sensor readings to determine the offset from center, and then updating the robot's rate of rotation. Suppose we let $d(n)$ be the signed displacement from center due to the n^{th} execution of the step function. Then, your feedback control algorithm will be setting the rate of rotation at the n^{th} step to be $Kd(n)$, were K is the feedback gain. Can you derive a second-order difference equation that can be used to solve for $d(n)$?

To help get you started, suppose that at step n, the robot is displaced from the center by an amount $d(n)$, and is moving with a speed V in a direction that makes an angle $\theta(n)$ as shown in Figure 1. In addition, suppose that the step function is executed every δt seconds. Then

$$\begin{aligned} d(n) &= d(n-1) + V\delta t \sin\theta(n-1) \\ &\approx d(n-1) + V\delta t\theta(n-1) \end{aligned}$$

where the approximation holds if the angle $\theta(n-1)$ is small.

Think about how to write a difference equation for $\theta(n-1)$, and how to combine the two difference equations in to a single equation for $d(n)$.

Question 8. How did you derive a difference equation for $d(n)$?

Question 9. How are the difference equation natural frequencies related to your feedback gain? Draw a plot of natural frequency locations in the complex plane for many values of feedback gain. Be sure to denote the location of the unit circle.

Question 10. What does your answer to the above question tell you about the solution to the difference equation model of the robot feedback system?

Question 11. Show plots of the difference equation model solution for different feedback gains and a nonzero initial offset from the corridor center. Why are your plots consistent with the natural frequency locations?

Question 12. What does the model tell you about your ability to find a feedback gain for which the robot will not eventual hit the wall of the corridor?

Checkpoint: 12:30

- Show your difference equation model for the robot displacement from center to an LA.
- Describe your answers to the above questions to an LA.

Exploration: Validating your model

Determine and execute experiments to validate your difference equation model. That is, show that your model predicts the actual robot's behavior. What does the model predict about how the robot will behave as you change the feedback gain? Is the model accurate?

There are a number of constants to determine, and units must be verified as consistent. Do the optical sensors return distances in meters? Does the `motorOutput` command set the forward velocity in meters per second and set the rotational velocity in radians per second? What is the sample rate of your feedback system?

There are many ways to determine the sampling rate being used by your robot. One direct way to measure the sample rate is to use the `time()` command, which you can use if you include the line:

```
from time import *
```

at the top of your brain. Please keep in mind that if you print something every time the brain executes a step, you will slow down the step function and change the sample rate!

To gather data from your robot, you can use the brain in `feedback_brain2.py`. You'll need to add your control algorithm to this second brain. The program in `feedback_brain2.py` will run the robot for a fixed number of steps (currently set to be 200, but you can change it by changing the value of `robot.nValues`), storing the left distance values. After 200 steps have been reached, the values are written out to a file called `leftValues.py`, and a graph of the series is drawn.

If you want to read the values in from the file (for example, to plot them on the same axes as another graph), then you can use the following commands in `IDLE`:

```
import pickle
f = open("foo.py", "r")
data = pickle.load(f)
f.close()
```

Note that the variable `data` will have the list of reading values.

Post-Lab Writeup for Tuesday and Thursday's labs: Due before lecture on October 2

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 **all** of the numbered questions in this lab handout (both Tuesday's software and Thursday's design lab).

We also want the code you wrote for Tuesday's or Thursday's lab (though there will not be much code to hand in for Thursday's lab).

Concepts covered in this assignment

Here are the important points covered in this assignment:

- Learn about difference equations and how to solve them.
- Learn about the connection between natural frequencies and the solutions of difference equations.
- Learn to reason about systems using difference equations.
- Begin to learn about the issue of stability in dynamic feedback systems.