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

Assignment for Week 7

- Issued: Tuesday, October 17th
- Includes homework and preparation due before class on Thursday, October 19th
- Also includes post-lab homework due before class on Tuesday, October 24th
- Readings:
  - Review 10/17 Lecture Notes

Dynamic Feedback Control

In many of the labs so far, you 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, our primary goal was to have you learn a little about the robot as well as have you investigate some ideas in software engineering and circuit design. In this lab we will also be using measurements to control the robot, also known as feedback control, but we will be investigating a situation in which detailed analysis is needed to achieve a desired performance. The problem we are asking you to investigate is 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 that simple feedback system. In next week's lab, you will use more sophisticated mathematical tools to develop improvements to the simple feedback scheme suggested in this lab.

The summary of tasks for this week are:

- BEFORE LAB: Tutorial exercises to practice solving difference equations
- BEFORE LAB: Write a program you can call from SOAR's interpreter to solve and plot difference equation solutions
- IN LAB: Implement the suggested simple feedback scheme and run experiments
- IN LAB: Derive a difference equation model of the robot behavior
- IN LAB: Conduct additional experiments to calibrate and validate the model
- POST LAB: Write a program to solve second order difference equations.

1. Homework in preparation for lab

This section includes problems to complete with the online tutor. These are due before lab on Thursday. The examples are to help you solidify your understanding of difference equations.

1.0 Read this whole document!

Please read the entire section on the lab before coming to lab.

1.1. Exercises with the online tutor

Use the online tutor to complete the problems due Thursday, October 19th.

1.2. Difference equation program from PS2 (revisited)

Problem set 2 asked you to write a program that generated the solutions to linear difference equations. Here's the sort of program you should have written (notice the use of list comprehension):

```
def diffeq(inits, coeffs):
    def add(x,y): return x+y
    order = len(coeffs)
    assert order == len(inits),\
    'Number of initial values must equal number of coefficients'
    def val(n):
        if n < order:
            return inits[n]
        else:
            return reduce(add,[coeffs[k]*val(n-k-1) for k in range(order)])
    return val
```

You would use this as follows:

```
>>>  fib=diffeq([0,1],[1,1])
>>>  fib(20)
6765
```

As we saw, this program generates a tree recursive process that has exponential order of growth in time. We also saw that we can speed it up by using memoization:

```
def memoize(f):
    storedResults={}
    def doit(n):
        if storedResults.has_key(n):
            return storedResults[n]
        else:
            value = f(n)
            storedResults[n] = value
            return value
    return doit
```

Adding memoization to `diffeq` requires inserting only a single (very tricky) line to the procedure:

```
def diffeq(inits, coeffs):
    def add(x,y): return x+y
    order = len(coeffs)
    assert order == len(inits),\
    'Number of initial values must equal number of coefficients'
    def val(n):
        if n < order:
            return inits[n]
        else:
            return reduce(add,[coeffs[k]*val(n-k-1) for k in range(order)])
    val=memoize(val) #****
    return val
```

Type in the definitions of `memoize` and `diffeq` and test them by computing Fibonacci numbers (again), as well as checking the analytic solutions to the difference equations derived in the tutor exercises.

Save your program in a place where you can get to it from Lab on thursday.

2. To do in lab

As we are sure you have come to expect, today's lab will begin with a mini-quiz that is based on the tutorial problems due for today, and on material covered this week. It should be no great surprise that you will be asked a question about the solution to a difference equation. Please keep in mind that the point of the quizzes is as a diagnostic for us (did we succeed in teaching you?) and to encourage you to do the tutorial problems before coming to lab (did you prepare?). After the quiz, you should work as part of a group with a single computer and robot.

2.1 Graphing Difference Equations

Start with the difference equation solver that you wrote for the prelab, be sure to put the program in the SOAR directory. Then, start the SOAR program and invoke the SOAR interpreter. Then you can open a graphing window.

GraphingWindows are used for graphing data and visually displaying mathematical information to the user. You can open as many GraphingWindows as you want. Open one with something like:

```
graph = GraphingWindow(400,400,0,10,-10,20,"MyGraph")
```

This will make a new window with a drawable area of 400x400, and set the coordinate frame of the window to run from x=0 to x=10 and y=-10 to y=20. It will also title the window MyGraph.

Given a python procedure f which takes an integer argument and returns a numerical value, the function

```
graph.graphDiscrete(f)
```

will plot $y = f(n)$ for $n = 0, 1, ...9, 10$. If you want to plot f for more values of n, you must create a new graphing window.

Demonstrate that you can use your differential equation solver and the SOAR graphing commands to graph solutions to difference equations. While testing your program, be sure to try some of the difference equations you investigated in the tutor problems.

Checkpoint: 2:00 PM

- Plots of solutions to the tutor difference equations.

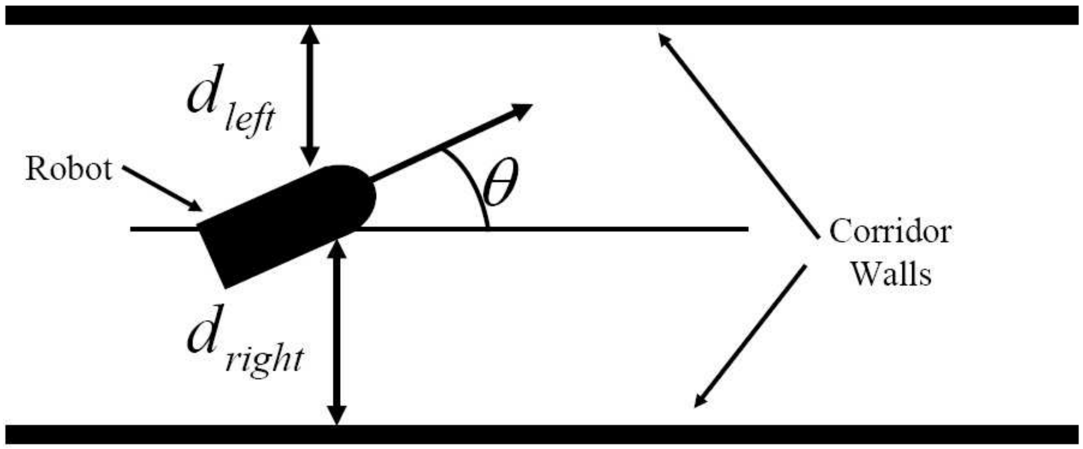

Figure 1: Robot in corridor.

2.2 Robot Feedback System

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

Consider the problem of trying to steer the robot down the center of the corridor while keeping it 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 with the command

```
motorOutput(0.2, 0.0),
```

which will keep the robot moving forward 0.2 robot-lengths per second. If the robot is too close to the right wall, one could issue the command

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

which will keep the robot moving 0.2 robot-lengths per second but will cause the robot to rotate to the left. If the robot is too close to the left wall, one could issue the command

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

which will keep the robot moving 0.2 robot-lengths per second but will cause the robot to rotate to the right.

The above suggests a very simple feedback system which keeps the robot moving forward at 0.2 robot-lengths 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 0.2 and set the rate of rotation in the motorOutput() command to be proportional to the distance from the center of the corridor.

Experiment with the above dynamic feedback control scheme by writing a SoaR brain that on every step, sets the robot velocity and updates the robot rotation so that it is proportional to the robot's signed distance from the corridor center ($d_{left} - d_{right}$). You could use one of the virtual sensors you developed in an earlier lab, though for ease of analysis later, we suggest you create virtual sensors that determine the distance to the left wall by computing the minimum of the 0^{th} and 1^{st} and the distance to the right wall by computing the minimum of the 6^{th} and 7^{st} sensor. Try your brain on your robot in one of our specially designed corridors. 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.

Checkpoint: 3:15 PM

- Code for the dynamic feedback control brain
- Results on robot motion in the corridor for several gains and initial positions

2.3 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 rate of rotation. Suppose we denote $d[n]$ to 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 $K * d[n]$, 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 center by an amount $d[n]$, and is moving with a velocity 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

$$d[n+1] = d[n] + V\delta t \sin\theta[n] \approx d[n] + V\delta t\theta[n]$$

where the approximation holds if the angle $\theta[n]$ is small.

Now think about how to write a difference equation for $\theta[n]$, and how to combine the two difference equations into a single equation for $d[n]$.

Checkpoint: 4:15 PM

- Difference equation model for the robot displacement from center

2.4 Validating your model

Determine and execute experiments to validate your 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 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 put 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!!!

Checkpoint: 5 PM

- Model validation (show computer plots and robot behavior)

3. To do before October 24th

Work on the following problems after Thursday's lab and turn in the written parts by October 24th at the beginning of class.

3.1. Lab writeup

Write a brief description of how you derived your difference equation model for the simple robot dynamic control system and explain how you determined the parameters in the difference equation. In addition, describe what experimental measurements you made to validate your model.

3.2. Automate solving second-order difference equations

Write a python program that solves arbitrary second-order homogeneous difference equations analytically, by computing the 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

$$x[n] = a_1 * x[n-1] + a_2 * x[n-2].$$

Your program should return a procedure that, when called with n, returns $x[n]$.

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 $j = \sqrt{-1}$. This should not surprise you, as entering EECS students, you already appreciate that the variable i must be reserved for electrical current.

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 square root function does not understand complex numbers.

What to turn in

For 3.1, hand in a brief (few page) lab report. For 3.2, hand in a copy of your carefully commented python code with an explanation, *written in actual English sentences*, of how and why it works. In addition, demonstrate that your function works for a number of test cases, including those examples from the prelab.

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