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

Assignment 6, Issued: Tuesday, Mar. 11

Overview of this week's work

In software lab

- Work through the software lab.
- Paste your solution to questions 1 and 4 into the PS.7.1 in the tutor.

Before the start of your design lab on Mar 13 or 14

- Read the class notes and review the lecture handout.
- Do the on-line tutor problems in section PS.7.2.
- 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

- Take the nanoquiz in the first 15 minutes; don't be late.
- Work through the design lab with a partner, and take good notes on the results of your work.

At the beginning of your next software lab on Mar 18 or 19

- Submit written solutions to all the questions in this handout. All written work must conform to the homework guidelines on the web page.

An *ex camera* midterm will be given on Wednesday 19 March:

- **You may pick up a copy in 34-501 any time between 8:30AM and 7PM on Wednesday 19 May.**
- **You must return your answers by 3 hours after the time you picked up the exam.**
- **You must do the exam by yourself; you may read anything (including written notes, or the web) but you may not communicate with anyone.**
- **We will have lecture, software lab, and design lab that week; attendance at lab is mandatory as usual, but there will be no lab report on those labs.**
- **If you do not have a three-hour block of time available on 19 May, send email to `lpk@csail.mit.edu`, and we'll schedule a conflict exam time.**

Software Lab: Combining System Functions

- Get the lab6 files via `athrun 6.01 update` or from the home page.
- If you're on an Athena machine, do `add -f 6.01`.

In this software lab, we will continue to build up the functionality of our `SystemFunction` class, by adding methods to create new system functions out of old ones. Then, we'll use the combination methods to build new system functions and try to build intuition for what is happening.

System Function combinators

To get started, we have to understand something about how system functions are represented inside the `SystemFunction` class, and what arguments its initializer needs.

Internally, we represent system functions using a pair of polynomials in R; to create a new instance of the system function class, we need to pass in two instances of the `Polynomial` class. Here is the definition of the helper function we used in last week's lab:

```
def systemFunctionFromDifferenceEquation(outputCoeffs, inputCoeffs):
    return SystemFunction(Polynomial(reverseCopy(inputCoeffs)),
                          Polynomial(reverseCopy(outputCoeffs)))
```

Let's look at what's going on here, in detail. First, as you know, the arguments to `systemFunctionFromDifferenceEquation` are a list of coefficients of the y terms, starting with $y[n]$ and going back to $y[n-k]$, and a list of the coefficients of the x terms, starting with $x[n]$ and going back to $x[n-j]$. That is, if the difference equation is

$$a_0 y[n] + a_1 y[n-1] + ... + a_k y[n-k] = b_0 x[n] + b_1 x[n-1] + ... + b_j x[n-j] \quad ,$$

then `outputCoeffs` is $[a_0, a_1, \ldots, a_k]$ and inputCoeffs is $[b_0, b1, \ldots, b_k]$.

If we convert this difference equation into an operator equation, then we get

$$Y(a_0 + a_1 R + a_2 R^2 + ... + a_k R^k) = X(b_0 + b_1 R + b_2 R^2 + ... + b_j R^j) \quad .$$

So, our system function is going to be described by the two polynomials in R. Remembering that we have to specify polynomials with the high-order coefficients first, that means that the arguments to `SystemFunction` need to be $[b_j, b_{j-1}, \ldots, b_0]$ and $[a_k, a_{k-1}, \ldots, a_0]$. This makes it clear that we need to reverse the order of the coefficients when we create the operator polynomials in the system function. We wrote the utility function `reverseCopy` that takes a list as an argument and returns a list that contains the elements of the original list in reverse order (but unlike the Python reverse operator, doesn't change the original list).

So, we make a system function by putting the coefficients in the right order for the operator polynomials and call the `SystemFunction` constructor on them.

Cascade When we connect two system with system functions H_1 and H_2 together, we get a new system with system function $H_1 \cdot H_2$. In the file `SystemFunctionToBeEdited.py`, you'll find the `cascade` method defined, but just containing a `pass` statement.

To test your systems' behavior more easily, we have provided a method of the `SystemFunction` class called `transduceF`. It takes the same arguments as `plotSequence`: a list of j previous values of x ($x[-j], \ldots, x[-1]$), a list of k previous values of y ($y[-k], \ldots, y[-1]$), a function mapping $n \geq 0$ into $x[n]$, and a number of points to be generated (defaults to 30).

Question 1: Fill in the `cascade` method, so that it returns a new system function that is equivalent to the cascade of the system functions it is given as arguments. Be sure that neither of the original system functions is modified.

Question 2: Make a cascade of two systems each of which delays the input by two. Look at the system function of the resulting system. Does it do what you expect? (If `new` is your new system function, you should try

```
new.transduceF([10, 20, 30, 40], [], lambda n: n)
```

Be sure you know what the answer should be before you try it.)

Question 3: Write the difference equation associated with the resulting system function.

Feedback Figure 1 shows a feedback configuration that is frequently used: the output, Y, of a system H_1 is "fed back" through a system H_2 to yield signal W. This signal is subtracted from the input X to yield an "error" signal E, which is the input to H_1. We can describe this system algebraically:

$$\begin{aligned} Y &= H_1 E \\ E &= X - W \\ W &= H_2 Y \end{aligned}$$

Now, we can solve that system of equations, to find that

$$Y = \frac{H_1}{1 + H_1 H_2} X \ .$$

This is known as Black's formula, and it gives us a characterization of the composite system, H_B, that we can think of as mapping from X to Y.

In the file `SystemFunctionToBeEdited.py`, you'll find the `feedback` method defined. The body curently contains:

```
if h2 == None:
    h2 = SystemFunction(Polynomial([1]), Polynomial([1]))
pass
```

The first two lines say that if the system function H_2 is unspecified, to simply use one with 1 in the numerator and denominator, which passes the input straight through to the output with no delay.

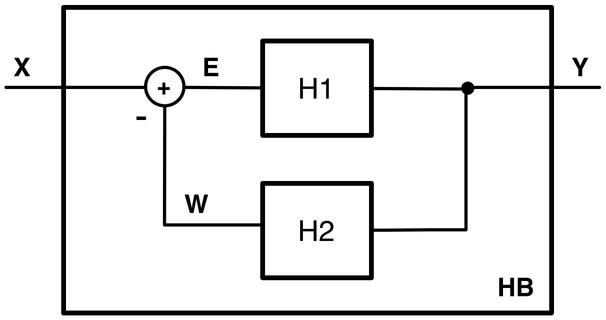

Figure 1: Feedback system

Question 4: Fill in the `feedback` method, so that if `h1` and `h2` are `SystemFunctions`, then

```
h1.feedback(h2)
```

is the system function for the system H_B in the figure above. Be sure that neither of the original system functions is modified. Also, specify the numerator and denominator of the new system function as simply as possible.

Question 5: Let H be the system function for a system that delays its input by one time step and doubles it. Apply the feedback operation to it. As a test, if your new system is called `new`, try:

```
new.transduceF([1], [0], lambda n: 0)
```

Be sure you understand what the arguments mean, and explain what the output should be, and why.

Robot meets walls, both ways

Now, go back to the problem from software lab 5, where we drove the robot toward the wall and tried to stop $D_{desired}$ from the wall. If you think of $D_{desired} - D = E$ as the error signal above (it might help to draw an instance of the feedback figure, and to remember that the signal W is subtracted from X to get E), then you can construct the system function of the whole system by defining system functions for the components, and composing them with `cascade` and `feedback`.

You should put your system function definitions for this lab in the file `lab6/ps6work.py`. You can use Run Module in Idle to evaluate these definitions in the Python interpreter.

Question 6: Define (in Python) the system function that maps E into V.

Question 7: Define the system function that maps V into D.

Question 8: Cascade those two systems to get one that maps E into D.

Question 9: Use feedback to make a system that maps $D_{desired}$ into D.

Question 10: Look at the system function. Is it the same or different from the one that you derived by hand in lab 5?

We'll do this again, with the problem from design lab 5, where we tried to drive the robot stably down a corridor. Remember that:

- $d[n]$ is the distance of the robot to the left of the centerline of the hallway
- $d[n] \approx d[n-1] + V\delta_T\theta[n-1]$
- $\Omega[n]$ is the robot's rotational velocity at time n
- $\theta[n]$ is the robot's angle with respect to the centerline of the hallway at time n
- We're assuming that $\Omega[n]$ is instantaneously dependent on $d[n]$ and $d_{Desired}[n]$

Question 11: Define the system function that maps E into Ω.

Question 12: Define the system function that maps Ω into Θ.

Question 13: Define the system function that maps Θ into D.

Question 14: Cascade those three systems to get one that maps E into D.

Question 15: Use feedback to make a system that maps D_{desired} into D.

Question 16: Look at the system function. Is it the same or different from the one that you derived by hand in lab 5?

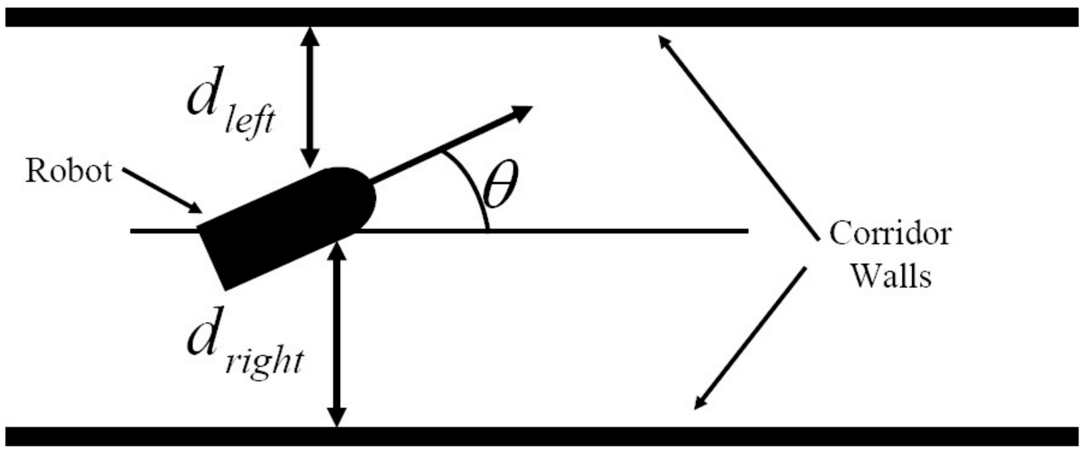

Figure 2: Robot in corridor.

Design Lab: Better Driving

- Get the lab6 files via `athrun 6.01 update` or from the home page.
- Get a lab laptop, a robot, a serial cable and a USB cable.

In the design lab of last week, you wrote a simple controller to steer your robot down a narrow corridor as shown in Figure 2. The controller set the forward velocity V to a constant (0.1 m/s) and the angular velocity Ω to be proportional to the error signal $e[n]$ which is the difference between the desired distance $d_{desired}$ and the current position $d = (d_{right} - d_{left})/2$.

As you recall from last week, we found that there was no good value for the constant of proportionality K between error and angular velocity. For small values of K the robot is sluggish and barely corrects errors and, for higher values, the robot moves with growing oscillations.

Designing a better controller

In order to design a better controller, we can try to process the error $e[n]$ in a more sophisticated way. For example, one could adjust the angular speed using some combination of the present and previous values of the displacement error,

$$\Omega[n] = K_1 e[n] + K_2 e[n-1]. \tag{1}$$

The robot angle, as before, satisfies the difference equation

$$\theta[n] = \theta[n-1] + \delta_T \Omega[n-1]. \tag{2}$$

Recall also that the position of the robot d can be computed from the angle θ and velocity V as

$$d[n] = d[n-1] + \delta_T V \theta[n-1]. \tag{3}$$

Finally, the error $e[n]$ is the difference between the desired and actual positions,

$$e[n] = d_{desired}[n] - d[n]. \tag{4}$$

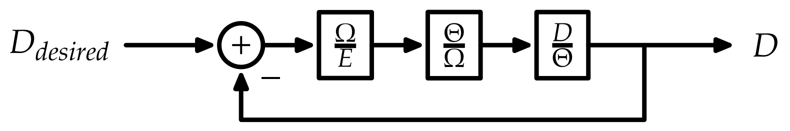

Figure 3: Block diagram of system described by equations 1-4.

Equations 1-3 can be represented by system functions that can be combined with a feedback loop (equation 4) as shown in Figure 3.

You should put your system function definitions for this lab in the file `lab6/ps6work.py`. You can use Run Module in Idle to evaluate these definitions in the Python interpreter.

Question 17: Use `systemFunctionFromDifferenceEquation` to represent each of equations 1-3 as system functions. Assume that the parameters `deltaT`, `V`, `K1`, and `K2` are available as variables.

Question 18: Use the `cascade` method of `SystemFunction` to combine the previous three system functions into one.

Question 19: Use the `feedback` method to convert the previous system function into one that maps $D_{desired}$ to D.

Question 20: Write a function called `makePositionController` that takes numerical values for δ_T, V, K_1, and K_2 and returns a `SystemFunction` that represents the transformation from $D_{desired}$ to D.

Check your answer by using `makePositionController` to determine the system function when $\delta_T = 0.2$, $V = 0.1$, $K_1 = 10$, and $K_2 = -1$. Your answer should have the form

```
SF(-0.004 R**3 + 0.040 R**2/-0.004 R**3 + 1.040 R**2 + -2.000R + 1.000)
```

The system function for the new position controller depends on two parameters that we will take as constants ($\delta_T = 0.2$ and $V = 0.1$) as well as two gains, K_1 and K_2, that we can adjust to optimize performance.

Characterizing Performance

We can characterize the performance of a system by evaluating its poles. If the magnitude of any pole is bigger than one, then the response of the system will grow without bound (i.e., it will be "unstable"). If a system has multiple poles, then the one with the biggest magnitude will most limit stability.

Question 21: Determine the poles of the system with new position control for $K_1 = 10$ and $K_2 = -1$ by running

```
makePositionController(0.2,0.1,10,-1).poles()
```

Briefly describe how the values of these poles will affect performance of the system.

Question 22: Write a function `maxMagnitude` that takes K_1 and K_2 as inputs and returns the maximum of the magnitudes of the poles of the system described by equations 1-4.

Optimizing Performance

Our goal is to find the values of K_1 and K_2 that give the most stable system performance. Thus we need to find the values of K_1 and K_2 for which `maxMagnitude(K1,K1)` is *smallest*.

Question 23: Find the best values of K_1 and K_2 by exhaustively searching the two-dimensional parameter space consisting of all integer values of K_1 and K_2 between -100 and 100. [Hint: we have provided a function `minOverGrid` that can be used to apply `minMagnitude` iteratively over a two-dimensional grid of numbers that correspond to K_1 and K_2.]

Checkpoint: 60 minutes

- Demonstrate your optimized new position controller by substituting your best gains into `makePositionController` and evaluating its poles.

Implementing the new position Controller

Modify `feedbackBrain.py` to implement the new position controller. You can start with the gains K_1 and K_2 that worked best in the model. However, there are many reasons why those gains may be too large. For example, there may be delays in the physical robot that are not explicitly accounted for in equations 1-4, and adding delays tends to destabilize feedback loops. Physical systems are also subject to random disturbances that we call noise. Noise can cause erratic behaviors, especially when feedback systems employ large gains.

Question 24: Test your `feedbackBrain.py` using the values of K_1 and K_2 that you found to have the smallest `maxMagnitude`. Briefly describe the resulting performance of the robot. Try several different starting positions and angles.

Question 25: Try to make the robot steering more stable by reducing both gains by factors of 1/2, 1/4, 1/10, ... Choose values of K_1 and K_2 that work well, and make plots of the robot's position as a function of time. What new problem is introduced when you reduce the gains?

Checkpoint: 120 minutes

- Demonstrate the new position controller with best performance on the robot. Compare the values of K_1 and K_2 that work best on the robot to those that worked best when you did the grid search. Explain any differences.

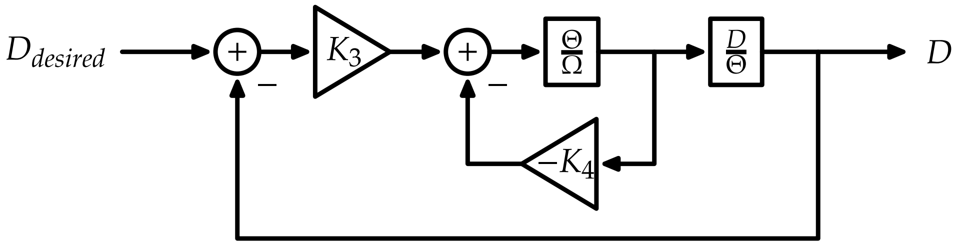

Figure 4: Block diagram of system with position-plus-angle steering.

Going Around in Circles

One way to think about the new position controller is that it uses not only position information (through $e[n]$, provided $d_{desired}$ is known) but also information about how fast position is changing (through $e[n] - e[n-1]$, provided $d_{desired}$ is constant). We know from our model that the rate of change of d is proportional to the angle of the robot, θ. Thus, the new position controller has access to both position and angle information when computing the next angular velocity. To make this idea more concrete, let's consider the case where $d_{desired} = 0$ for simplicity, and let $\Omega[n]$ represent the angular velocity commanded on step n

$$\Omega[n] = K_1 d[n] + K_2 d[n-1] = \alpha(d[n] + d[n-1])/2 + \beta(d[n] - d[n-1])/2 \approx \alpha d + \beta\theta$$

where $\alpha = K_1 + K_2$ and $\beta = K_1 - K_2$ are the gains to position and angle, respectively. Best performance was predicted for $K_2 \approx -K_1$, which corresponds to a relatively large dependence on angle compared to position.

There are two problems with calculating angle from differences in position. First, it introduces an additional delay in the feedback loop, and delays tend to destabilize feedback. The second is that relatively large values of gain are required to extract and emphasize angle. Both of these problems can be avoided by estimating angle as well as position from the optical sensors.

Analyze Position-plus-Angle Steering

Position-plus-angle steering can be implemented by combining the following equation

$$\Omega[n] = K_3 e[n] + K_4 \theta[n] \tag{5}$$

with equations 2-4. Figure 4 illustrates the relation between these equations using a block diagram.

Question 26: Write a function called `makePositionPlusAngleSteering` that takes numerical values for δ_T, V, K_3, and K_4 and returns a `SystemFunction` that represents the transformation from $D_{desired}$ to D for position-plus-angle steering.

Question 27: Find the best values of K_3 and K_4 by exhaustively searching the two-dimensional parameter space consisting of all integer values of K_3 and K_4 between -100 and 100.

Using Position-plus-Angle Steering

Substitute the function `sensors.getLRT` for `sensors.getLR` in `feedbackBrain.py`. Like `sensors.getLR`, `sensors.getLRT` returns l and r (the distances (in meters) to the left and right wall, respectively). However, `sensors.getLRT` also returns the angle of the robot (in radians), with respect to the walls (we assume that the robot is between straight, parallel walls). Implement a new controller that depends on $d[n]$ as well as the angle $\theta[n]$.

Question 28: Demonstrate your best position-plus-angle controller on the robot. Compare the values of K_3 and K_4 that work best on the robot with those that worked best in simulation. Explain any differences.

Checkpoint: 180 minutes

- Demonstrate the best position-plus-angle controller on the robot. Compare the values of K_3 and K_4 that work best on the robot to those that worked best when you did the grid search. Explain any differences.

Exploration: Will be issued separately

Concepts covered in this lab

- Learn about system functions and how they relate to difference equations.
- Learn how to combine system functions.
- Learn how to analyze feedback systems.
- Learn how to design controllers for feedback systems.
- Learn to characterize feedback systems by their stability and responsiveness.