Design Lab 4 *6.01 – Fall 2011*

Hitting the Wall

Goals: In **Design Lab 2**, you developed a “distance-keeping” brain to position a robot a fixed distance in front of a wall, even when the wall moved. In this lab, we develop a signals and systems model of that system, and use the model to understand the performance of the brain/robot system. With this model, we will be able to answer key performance questions, such as: was the response fast or slow? Did the system overshoot or oscillate?
This will be studied in three steps:

- Develop a difference equation model of the wall finder system
- Build a state machine model of the feedback system
- Implement a robot brain realizing the state machine controller

1 Introduction

Resources: This lab should be done with a partner. Each partnership should have a lab laptop or a personal laptop that reliably runs soar.

- `designLab04Work.py`: Template with appropriate imports for making state machines and plotting their outputs.
- `smBrainPlotDistSkeleton.py`: Template simple brain with a place for you to write the state machine for the controller.
- **Chapter 5** of the course notes.

Be sure to mail all of your code and plots to your partner. You will both need to bring it with you to your first interview.

Consider a brain/robot system which senses the distance the robot is away from a wall, and actuates the robot motors to move the robot to keep it a certain distance from the wall. This is an example of a simple control system with a single input (the desired distance to the wall) and a single output (the actual distance to the wall). We can think of the control system as having three subsystems: the “controller,” the “plant,” and the “sensor.”

Think of the input as being set by a user (e.g. you) or by some planning system that the robot uses to navigate in the world (e.g. a state machine that moves the robot from one target point to the next in some sequence). Thus, the input might stay constant for some period of time, and then change to a new value for some period of time, and so on.

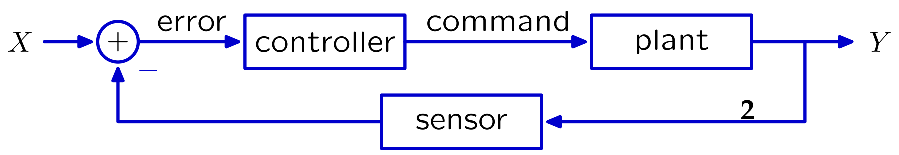


Figure 1 Structure of Control System. For today's lab, the controller is the robot's brain, the plant is the part of the system that is being controlled (i.e., the robot's locomotion system, whose input is an `io.Action` and whose output is the robot's position), and the sensor is the robot's sonars.

Generally, we design the controller to *command* the plant so that its (the plant's) output Y tracks some desired value X. For example, in the case of asking a robot to stay a particular distance from a wall, we want the output Y to be as close as possible to the input X. Since the plant is typically a physical system, the output of the plant (e.g., the position of the robot) is not easily compared directly with X. Rather, the physical output is measured by the sensor, whose output (which is typically electronic) can be subtracted from X to determine the *error*. Ideally, the output of the sensor is proportional to the output of the plant. More generally, however, the sensor introduces its own distortions, delays, and noise.

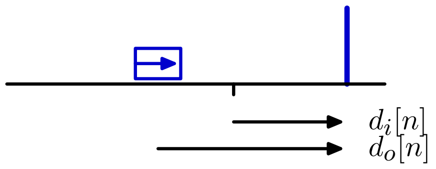
Wk.4.2.1 **Do tutor problem Wk.4.2.1 if you have not already done so.**

2 Difference equations for wall finder

Objective: Develop a difference equation model for the wall finder system.

Detailed guidance:

Make a simple model of the brain/robot system, as follows. Let $d_o[n]$ (the 'o' stands for *output*) represent the current distance from the robot to the wall, and let $d_i[n]$ (the 'i' stands for *input*) represent the (current) desired distance to the wall. Also let v represent the forward velocity of the robot. Let $T = 0.1$ seconds represent the time between steps.



When the robot receives a new command, we assume that the robot **immediately** changes its velocity and then holds the new velocity constant until it recieves the next command (i.e., the robot accelerates so fast that we can ignore the acceleration time).

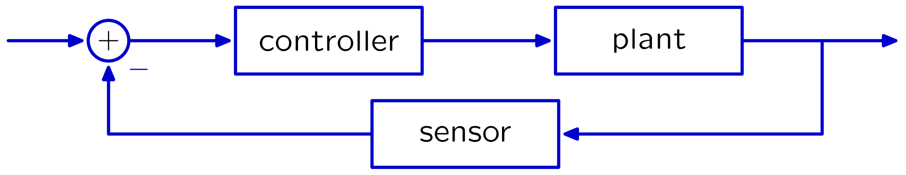
Check Yourself 1. Given the following conditions, what is the distance to the wall on step 1?

$v[0] = 1$	$d_o[0] = 3$
$v[1] = 2$	$d_o[1] =$ $\boxed{}$

Wk.4.3.1 **Enter your answer into part 1 of Wk.4.3.1 and check it.**

Assume the system has the structure shown in **Figure 1**. Assume that the sensor measures the current distance $d_o[n]$ and generates the sensed distance $d_s[n]$, which is equal to the current distance **delayed** by one step time. Let $e[n]$ represent the error signal, which is the difference between the input distance $d_i[n]$ and the sensed distance $d_s[n]$. On each step, the controller commands a forward velocity $v[n]$ in proportion to the error so that $v[n] = ke[n]$. Choose k so that the velocity is $5\,\mathrm{m/s}$ when the desired location is $1\,\mathrm{m}$ in front of the robot (think about the previous figure showing the position of the robot in order to help frame this calculation for k).

Check Yourself 2. Fully label the following system diagram. Include d_o, d_i, d_s, v, e.



Determine difference equations (using constants T and k) to relate the input and output signals of the following system components.

- the controller
- the model of the plant
- the model of the sensor

Wk.4.3.1 Enter these equations into parts 2, 3, and 4 of tutor problem Wk.4.3.1.

Combine these equations to derive a difference equation that relates d_o to d_i, by:

1. Converting the difference equations to operator equations in $\mathcal{R}$,
2. Solving for D_o in terms of D_i, and
3. Converting the result back to a difference equation.

Wk.4.3.1 Enter these equations into part 5 of tutor problem Wk.4.3.1.

Checkoff 1. **Wk.4.3.2**: Explain your results to a staff member.

3 State machines primitives and combinators

Objective: Build a state machine model of the wall finder system, based on its system diagram representation.

The wall finder system, as represented by the system diagram of **Figure 1**, can be modeled as a combination of primitive state machines, using two basic constructs: `sm.Gain` and `sm.R`. This is true because the wall finder system is a **"linear time-invariant" (LTI) system**.

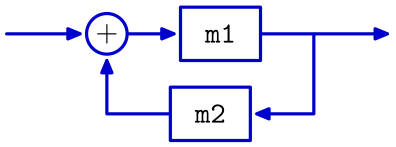
The `sm.Gain` state machine is really just a pure function: the output at step n is the input at step n, times a constant, k. The state is irrelevant. The reason we create this as a type of state machine is that we want to use the principles of PCAP to be able to combine it with other state machines to create new kinds of state machines.

```
class Gain(SM):
    def __init__(self, k):
        self.k = k
    def getNextValues(self, state, inp):
        return (state, self.k*inp)
```

The `sm.R` state machine is a renamed version of the `Delay` state machine. It takes a value at initialization time which specifies the initial output of the machine; thereafter, the output at step n is the input at step $n - 1$.

```
class R(SM):
    def __init__(self, v0 = 0):
        self.startState = v0
    def getNextValues(self, state, inp):
        return (inp, state)
```

For the purposes of building LTI systems, the *feedback addition* composition will be useful. It takes two machines and connects them like this (note that we are using generic boxes here, those boxes would be a triangle if the state machine were simply a gain, or would be labeled with an R if the state machine were a delay, or could be some more complicated feedback loop):



If `m1` and `m2` are state machines, then you can create their feedback addition composition with

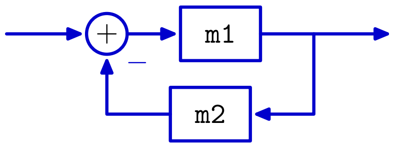
```
newM = sm.FeedbackAdd(m1, m2)
```

Now `newM` is itself a state machine. So, for example, `newM = sm.FeedbackAdd(sm.R(0), sm.Gain(1))` makes a machine whose output is the sum of the inputs from step 0 up to but not including the present step. You can test it by feeding it a sequence of inputs; in the example below, it is the numbers 0 through 9:

```
>>> newM.transduce(range(10))
[0, 0, 1, 3, 6, 10, 15, 21, 28, 36]
```

Feedback subtraction composition is the same, except the output of `m2` is *subtracted* from the input, to get the input to `m1`. (Note the minus sign next to the output of `m2` as it is fed into the adder.) You can use it like this:

```
newM = sm.FeedbackSubtract(m1, m2)
```



Note that if you want to apply one of the feedback operators in a situation where there is only one machine, you can use `sm.Gain(1.0)` or `sm.Wire()` as the other argument.

Check Yourself 3. Use gains, delays, and adders to draw a system diagram for the first system in tutor problem **Wk.4.2.1**. (That is, the tutor problem that you did before coming to lab).

Check Yourself 4. Use gains, delays, and adders to draw a system diagram for the second system in tutor problem **Wk.4.2.1**.

Check Yourself 5. Use gains, delays, and adders to draw a system diagram for the third system in tutor problem **Wk.4.2.1**.

Wk.4.3.3 Do tutor problem Wk.4.3.3 (State machine composition)

Check Yourself 6. Use gains, delays, and adders to draw a system diagram for the **controller** in the wall-finder system.

Check Yourself 7. Use gains, delays, and adders to draw a system diagram for the **plant** in the wall-finder system.

Check Yourself 8. Use gains, delays, and adders to draw a system diagram for the **sensor** in the wall-finder system.

Check Yourself 9. Connect the previous three component systems to make a diagram of the wall-finder system. Label all the wires. Draw boxes around the controller, plant, and sensor components.

Checkoff 2. **Wk.4.3.4** Explain your system diagrams to a staff member. Identify instances of cascade and feedback composition for the wall-finder system.

Wk.4.3.5 Do tutor problem Wk.4.3.5.
Write your code in `designLab04Work.py`. Each function should return an instance of state machine of the corresponding block. Test your code with
> idle -n
Then submit your code in the Tutor.

With `T=0.1` and an initial distance to the wall of 1.5 meters, experiment with different values of the gain. You can do this using the `plotD` procedure, defined in `designLab04Work.py`. Use idle -n. For a given gain value, `k`, it will make a plot of the sequence of distances to the wall.

Check Yourself 10. Find three different values of `k`, one for which the distance converges monotonically, one for which it oscillates and converges, and one for which it oscillates and diverges. Make plots for each of these `k` values. Save screen shots (see Reference tab of the 6.01 website) for each of these plots.

Wk.4.3.6 Enter the gains you found into the tutor.

4 On the simulated robot

Objective: Implement a brain for the wall-finder problem using a state machine, as described in **Section 2**.

Recall that the robot itself is the plant and so we do not need to write any code for that. We have already implemented a `Sensor` state machine which outputs a delayed version of the values of sonar sensor 3 (the robot actually has relatively little delay in its sensing).

Your job is to implement the `Controller` state machine, which takes as input the output of the sensor state machine, and generates as output instances of `io.Action` with 0 rotational velocity and an appropriate forward velocity. It should depend on `dDesired`, which is the desired distance to the wall. Do this by editing the `getNextValues` method of the `Controller` class in `Desktop/6.01/designLab04/smBrainPlotDistSkeleton.py`. Your controller should attempt to make the output of sonar sensor 3 be equal to 0.7, even though sensor 3 doesn't point straight forward.

Check Yourself 11. For each of the three gains you found **Check Yourself 10**, run the simulated robot in the `wallFinderWorld.py` world, and save the plots.

Checkoff 3. **Wk.4.3.7** Compare the plots from **Check Yourself 10** and **11**. Explain how they differ, and speculate about why. Email your code and plots to your partner.