

Staggering Proportions

Goals:

In this lab, we will program the robot to move along a wall to its side, maintaining a constant, desired distance from the wall. We will use a simple *proportional controller*, and model it as a system with the same structure that we used for the *wall finder* system of [Design Lab 4](#).

This will be studied in three steps:

- Build a proportional controller for a robot and test it in simulation with different gains.
- Build an analytical model of the controller-plant-sensor system, both by hand and using the `SystemFunction` class.
- Use the model to gain understanding about the best gain to use for the controller and what kind of behavior to expect from the system.

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.

- `propWallFollowBrainSkeleton.py`: Template brain with a place for you to write the proportional controller.
- `designLab05Work.py`: Template python script with appropriate imports for making system functions and finding their properties.
- [Chapter 5](#) of the course notes.

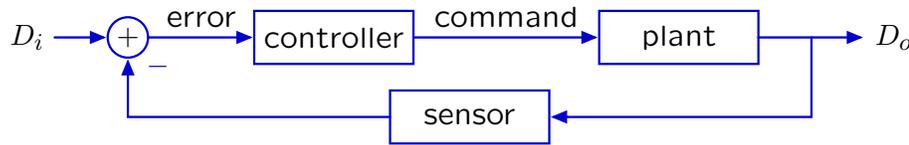
Be sure to mail all of your code and plots to your partner. Each of you will need to bring copies with you to your first interview.

We have been studying a basic feedback system, composed of three parts: a controller, a plant, and a sensor. In last week's lab, this system was modeled as a difference equation, in which the input was the desired distance to the wall, the output was the actual distance to the wall, and the sonars provided feedback on the sensed distance to the wall.

This week, we generalize from the simple one-dimensional model, to a two-dimensional world, and seek to guide a robot such that it moves parallel to a wall, staying a specific distance away. In

addition, we generalize from difference equations to a model using abstract signals, and system equations.

The basic structure of the system we build will remain the same:



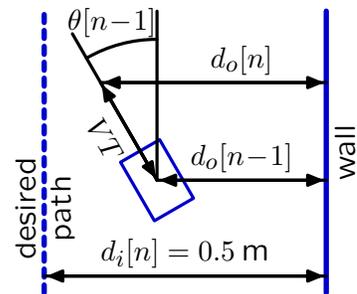
However, the symbols attached to the diagram are now signals, and not just samples. Specifically, here are definitions of various symbols to be used in this lab:

- k : gain of the controller, a constant number
- V : forward velocity of the robot, a constant number
- T : the amount of time between discrete samples of the signals, a constant number
- D_i : desired distance of the robot to the wall, a signal whose samples are $d_i[n]$
- D_o : actual distance of the robot to the wall, a signal whose samples are $d_o[n]$
- E : error, equal to $D_i - D_o$, a signal whose samples are $e[n]$
- Θ : robot's angle with respect to the wall, a signal whose samples are $\theta[n]$
- Ω : robot's angular velocity, a signal whose samples are $\omega[n]$

2 Proportional wall-follower

Objective: Build a proportional controller for a robot and test it in simulation with different gains .

The figure to the right illustrates a robot in a hallway, with its desired path a fixed distance from the right wall. We can build a controller with a fixed forward velocity of $V = 0.1$ m/s, and adjust the rotational velocity $\omega[n]$ (not shown) to be proportional to the *error*, which is the difference between the desired distance $d_i[n] = 0.5$ m to the right wall and the actual distance $d_o[n]$. The constant of proportionality between the error and the rotational velocity is called the *gain*, and we will write it as k . Notice that when the rotational velocity $\omega[n]$ of the robot is positive the robot turns towards the left, thus increasing its angle $\theta[n]$.



Look in the file `propWallFollowBrainSkeleton.py`. The brain has two state machines connected in cascade. The first component is an instance of the `Sensor` class, which implements a state machine whose input is of type `io.SensorInput` and whose output is the perpendicular distance to the wall on the right. The perpendicular distance is calculated by `getDistanceRight` in the `sonarDist` module by using triangulation (assuming the wall is locally straight). All of the code for the `Sensor` class is provided.

The second component of the brain is an instance of the `WallFollower` class. Your job is to provide code so that the `WallFollower` class implements a proportional controller.

Check Yourself 1. What should be the types of the input to and the output from a state machine of the WallFollower class? What should be the sign of k ?

Detailed guidance:

Step 1. Implement the proportional controller by editing the brain. Then use soar to run your brain in the world `wallTestWorld.py`. The brain is set up to issue a command during the setup to rotate the robot, so it starts at a small angle with respect to the wall.

Step 2. Experiment with a few values of the gain k to determine how k affects the resulting behavior.

Generate plots to illustrate the trends that you found.

Save screen shots of your plots and email them to your partner.

Checkoff 1. **Wk.5.3.1:** Show your plots to a staff member.
Describe how the gain k affects behavior of the wall-follower.
For what value of k (if any) does the distance to the wall converge most rapidly to the desired value of 0.5 m?
Compare the effect of k in this lab and in the lab 4 (wall finder).

3 Mathematical model

Objective: Construct an analytical, mathematical model of the proportional wall-follower system.

We will now construct a model of the system so that we can determine analytically how the performance of the system depends on gain. The **analytic approach** is less tedious and more insightful than trying to exhaustively search the parameter space using soar.

You may find it helpful to read **Section 5.8.3** of the readings, which illustrates a similar modeling effort for the wall-finder system.

Step 3. Controller model: Assume that the controller can instantly set the rotational velocity $\omega[n]$ to be proportional to the error $e[n]$. Express this relation as a difference equation.

Step 4. Plant model: Write difference equations describing the “plant,” which relates the input (angular velocity) to the output (distance to the wall). It is useful to break this problem into two parts:

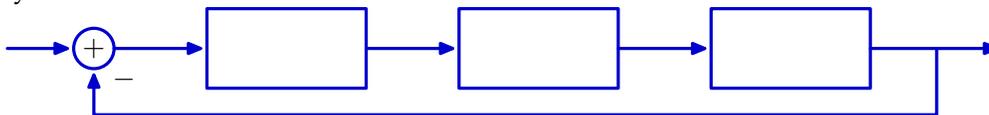
- **Plant 1:** Write an expression for $\theta[n]$, the robot’s angular orientation with respect to the wall, that depends on its rotational velocity. Assume that the rotational velocity at time $n - 1$ is $\omega[n - 1]$, and that this rotational velocity is constant until time n .

- **Plant 2:** Write an expression for $d_o[n]$, that depends on angular displacement. Assume that the angular displacement at time $n - 1$ is $\theta[n - 1]$, and that this angular displacement is constant until time n .

Linearize the resulting difference equation using the small angle approximation (i.e., if θ is small, then $\sin \theta \approx \theta$). This approximation makes our model linear, allowing us to analyze it easily. It may be useful, however, to think about the consequences of this approximation when trying to account for behaviors in subsequent sections.

Sensor model: To keep things simple, we will model the sensor as a wire: that is, assume it introduces no delay.

The subsystems represented by the three difference equations above connect together to form a system of the following form. Label each wire in the block diagram with the name of the corresponding signal. You may also find it useful to label each of the boxes in this diagram with the element of the model (controller, plant1, plant2) corresponding to each of the three difference equations you derived.



Step 5. Convert your difference equations into operator (\mathcal{R}) equations, and find the system function.

$$H = \frac{D_o}{D_i} =$$

Wk.5.3.2

Enter the system function into the tutor by entering the numerator and denominator polynomials.

Step 6. Find an algebraic expression for the poles p of the system.

$p =$

Step 7. Choose one of the poles p , and make two sketches (on the same axis) of the real and imaginary parts of p^n when $k = 1$, $T = 0.1$ second, and $V = 0.1$ m/s. What does this plot tell you about the **response of the system**?



Checkoff 2.

Wk.5.3.3: Describe the locations of the poles in **Step 6**. What are the implications of those locations for system behavior? Compare your plot in **Step 7** with the results from **Checkoff 1**. Explain similarities and differences. Are the periods the same?

4 Software model

Objective: Construct a model of the proportional wall-follower system using the **SystemFunction class**.

We now develop a software representation of system functions to automate much of the preceding algebra. The software solution allows analysis of large systems that would be difficult to analyze by hand. However, our software solution requires numerical values of parameters such as k and T . (You can learn about systems for symbolic manipulation of such parameters in more advanced programming subjects. Also compare and contrast with the **software representation for signals**.)

The **SystemFunction class** provides two primitive kinds of system functions: gain (`sf.Gain`) and delay (`sf.R`). They are implemented as Python procedures, but named with uppercase variables by analogy with the `sm.Gain` and `sm.R` state machines.

```
def Gain(k):
    return sf.SystemFunction(poly.Polynomial([k]), poly.Polynomial([1]))
def R():
    return sf.SystemFunction(poly.Polynomial([1, 0]), poly.Polynomial([1]))
```

These can be combined, using `sf.Cascade`, `sf.FeedbackSubtract`, and `sf.FeedbackAdd` to make any possible system function; and the structure of the combination will be the same as it would have been to build up the analogous state machine.

Note that while the internal definitions of `Gain` and `R` use **polynomials** to construct them, by abstraction you can use these system functions, together with our means of combination without ever having to utilize these internal details.

Check Yourself 2. Use gains, delays, and adders to draw system diagrams representing the controller, plant 1, and plant 2 from the previous section.

Controller:

Plant 1:

Plant 2:

- Step 8.** Edit the `designLab05Work.py` file to implement Python procedures called `controller`, `plant1`, and `plant2` that use `sf.Gain`, `sf.R`, `sf.Cascade`, `sf.FeedbackSubtract`, and `sf.FeedbackAdd` to construct and return instances of the `SystemFunction` class that represent the three blocks in the mathematical model. Pass the important parameters for each block (e.g., `k`) as inputs to the corresponding procedure.
- Step 9.** Write a Python procedure `wallFollowerModel(k, T, V)` that calls the previous Python procedures to make system functions for the components and composes them into a single `SystemFunction` that describes the system with `desiredRight` as input and `distanceRight` as output.

Wk.5.3.4

Enter the definition for `wallFollowerModel` and any procedures it calls into the tutor. Do not enter any `import` statements.

- Step 10.** Use the dominantPole method of your system model to determine the period that will result for $T = 0.1$ seconds, $V = 0.1$ m/s, and the values of k you used in **Checkoff 1**.

k period _____

Checkoff 3. **Wk.5.3.5:** Show your results to a staff member. Explain similarities and differences between your results in this part and your results in **Checkoff 1**.

MIT OpenCourseWare
<http://ocw.mit.edu>

6.01SC Introduction to Electrical Engineering and Computer Science
Spring 2011

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.