# CSCI 445 Lab 5 — Wall Following

Prof. Ayanian, University of Southern California Spring 2017

In this lab we will learn how to use a PD-Controller to let our robot follow a wall.

## 1 Prerequisites

- Review feedback control (i.e. Lecture 6).
- Bring your laptop.

## 2 Controller Architecture

Your P(D)-controller requires the robots' current state and goal state in order to produce an output value. In our example, we would like the robot to follow a wall using only sensors which we already learned about. What is the current state in this case and how can you estimate it? What is the goal state?

The robot moves if we provide two values to the `drive_direct` function: the velocities for each wheel individually. However, our P(D)-controller will only compute a single scalar output value. Find two mathematical functions, one for each wheel, which compute the desired velocity for the left and right wheel respectively, based on the output of your P(D)-controller. Specifically, find $f_{left}(\cdot)$ and $f_{right}(\cdot)$, with $v_{left} = f_{left}(pdoutput)$ and $v_{right} = f_{right}(pdoutput)$.

## 3 P Controller

Now we would like to implement a P-Controller which will help us with the wall-following task.

### 3.1 Controller Implementation

Since we need controllers later in the lab, it is best if we create generic, re-usable code. Create a new file `p_controller.py` containing a class `PController`. The class has just two functions: the constructor taking all the required tuning parameters, and an `update` function, which has to be called with arguments for the current state and desired state.

Furthermore, your `update` function should clamp your output value before it returns it. Clamping restricts the output value to be within a specified range (i.e. it saturates the signal, see Figure 1). Hence, this range should be a parameter for the `PController` class constructor as well. Clamping is useful because you know that the robot will only be able to execute velocities between -500 to $500\,\mathrm{mm/s}$. If you send values which are out of this range, the behavior is unpredictable (in the worst case such out-of-range values can even cause damages to the hardware.) You need to adjust the clamping range of your controller such that $f_{left}(\cdot)$ and $f_{right}(\cdot)$ never produce invalid velocities.

### 3.2 Wall Following in Simulation

Based on the previous sections you can now build a simple wall-following robot. Create a new file, e.g. `lab5.py` which uses your `PController` to follow a wall. Test it in simulation using V-REP with `Lab5.ttt`.

Change the controller gain and provide plots of the measured distance for three different gains. The V-REP scene already contains a view which will plot the data in real-time for you. Which gain works best for you?

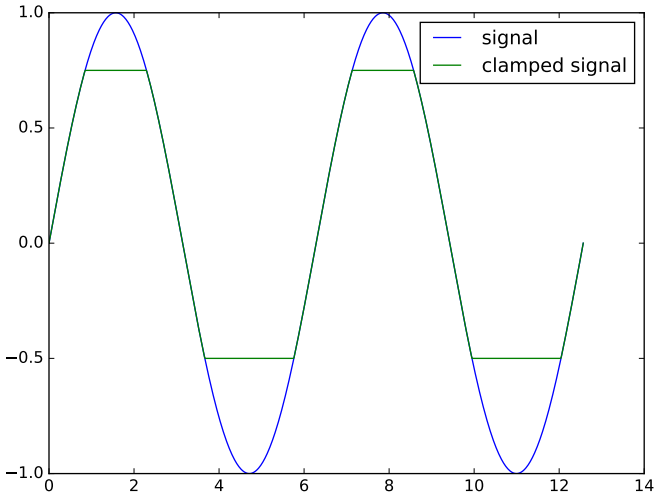


Figure 1: Sinus-wave signal and its clamped version to be within the bound$[-0.5, 0.75]$.

### 3.3 Wall Following on the Robot

Test your code on the robot. Re-tune the gain if necessary. Does it behave the same as in simulation?

## 4 PD Controller

Now we would like to extend our controller to include a damping (D controller) term and compare it to the P Controller.

### 4.1 Controller Implementation

Copy your existing `PController` into a new file called `pd_controller.py`. Change the class name to `PDController` and update the code such that your controller includes a damping term.

**Hint:** You will need to include the current time in your `update` function.

### 4.2 Wall Following in Simulation

Repeat your experiment from section 3.2. Now you have two gains to tune. Provide plots of at least three different sets of gains and report a good set of PD gains. How does the PD controller compare to the P Controller?

### 4.3 Wall Following on the Robot

Test your code on the robot. Re-tune the gain if necessary. Does it behave the same as in simulation?