**PID Controller**

The governing equation of this system is

Taking the Laplace transform of the governing equation, we get

The transfer function between the input force  and the output displacement  then becomes

To find the open-loop step response we use the following code:

close all

clear

% Define System Parameters

m = 1; %kg

b = 10; %N s/m

k = 20; %N/m

F = 1; %N

% Define the Transfer Function

num = [1];

den = [m b k];

sys = tf(num,den);

% Step Response of Uncontrolled Open Loop System

step(sys)

The DC gain of the plant transfer function is 1/20, so 0.05 is the final value of the output to a unit step input. This corresponds to a steady-state error of 0.95, which is quite large. Furthermore, the rise time is about one second, and the settling time is about 1.5 seconds. Let's design a controller that will reduce the rise time, reduce the settling time, and eliminate the steady-state error.

The proportional controller () reduces the rise time, increases the overshoot, and reduces the steady-state error.

The closed-loop transfer function of our unity-feedback system with a proportional controller is the following, where  is our output and our reference  is the input:

Let the proportional gain () equal 300 and add the following lines to your code:

% Adding a Proportional Controller

Kp = 300;

C = pid(Kp);

sys\_closed1 = feedback(sys\*C,1);

figure

step(sys\_closed1)

The resulting plot shows that the proportional controller reduced both the rise time and the steady-state error, increased the overshoot, and decreased the settling time by a small amount.

The addition of derivative control () tends to reduce both the overshoot and the settling time. The closed-loop transfer function of the given system with a PD controller is:

Let the proportional gain () equal 300 and the derivative gain () equal 10 and add the following lines to your code:

% Adding a Proportional-Derivative Controller

Kp = 300;

Kd = 10;

C = pid(Kp,0,Kd);

sys\_closed2 = feedback(sys\*C,1);

figure

step(sys\_closed2)

The plot shows that the addition of the derivative term reduced both the overshoot and the settling time, and had a negligible effect on the rise time and the steady-state error.

Before proceeding to PID control, let's investigate PI control. The addition of integral control () tends to decrease the rise time, increase both the overshoot and the settling time, and reduces the steady-state error. For the given system, the closed-loop transfer function with a PI controller is:

Let's reduce to 30, and let equal 70. Add the following lines to your code.

We have reduced the proportional gain () because the integral controller also reduces the rise time and increases the overshoot as the proportional controller does (double effect). The above response shows that the integral controller eliminated the steady-state error in this case.

Now, let's examine PID control. The closed-loop transfer function of the given system with a PID controller is:

After several iterations of tuning, the gains = 350, = 300, and = 50 provided the desired response.

**Automatic PID Tuning:**

MATLAB provides tools for automatically choosing optimal PID gains which makes the trial and error process described above unnecessary. You can access the tuning algorithm directly using pidtune or through a nice graphical user interface (GUI) using pidTuner.

The MATLAB automated tuning algorithm chooses PID gains to balance performance (response time, bandwidth) and robustness (stability margins). By default, the algorithm designs for a 60-degree phase margin.

Let's explore these automated tools by first generating a proportional controller for the mass-spring-damper system by entering the command shown below. In the shown syntax, sys is the previously generated plant model, and 'p' specifies that the tuner employ a proportional controller.

pidTuner(sys,'p')

We can now interactively tune the controller parameters and immediately see the resulting response in the GUI window. Try dragging the Response Time slider to the right to 0.14 s. This causes the response to indeed speed up, and we can see is now closer to the manually chosen value. We can also see other performance and robustness parameters for the system. Note that before we adjusted the slider, the target phase margin was 60 degrees. This is the default for the pidTuner and generally provides a good balance between robustness and performance.

Now let's try designing a PID controller for our system.

pidTuner(sys)