rldesign.PID Prop−integral−derivative controller design

We have designed P, PI, and PD controllers. Now we include all three terms in a single PID controller. With this, we can design for both steady-state and transient response.
A PID controller transfer function will have one pole and two zeros. One zero z_I and the pole will be specified by an integral compensator and the other zero z_D will be specified by a derivative compensator.
Our design process will yield a PID controller with transfer function

$$\underbrace{K_1}_{\text{P design}} \cdot \underbrace{K_2(s - z_D)}_{\text{D compensation}} \cdot \underbrace{K_3 \frac{(s - z_I)}{s}}_{\text{I compensation}} = K_P + K_I/s + K_D s, \tag{1}$$

where the named gains are called proportional K_P, integral K_I, and derivative K_D. The design procedure below will yield numbered gains K_1 (P design), K_2 (D compensation), and K_3 (I compensation). They are related as follows:

$$K_P = -K_1 K_2 K_3 (z_D + z_I) \tag{2}$$
$$K_I = K_1 K_2 K_3 z_I z_D \tag{3}$$
$$K_D = K_1 K_2 K_3. \tag{4}$$

Our design procedure is as follows.

1. Check that the integral compensation of a PID controller is necessary and sufficient to meet the steady-state performance criteria.
2. From the transient performance criteria and using the second-order approximation, determine the region of the s-plane in which the dominant closed-loop poles of the root locus should appear.
3. Design a P controller and evaluate its transient response performance.

4. Apply derivative (D) compensation to improve the transient response. Simulate to verify the transient response performance.
5. Apply integrator (I) compensation to improve the steady-state error performance.
6. Check all performance criteria and adjust gains and zero locations, as-needed.
7. Determine gains: proportional K_P, integral K_I, and derivative K_D.

A design example

Let a system have plant transfer function

$$\frac{s+40}{s^2+10s+200}. \quad (5)$$

Design a PID controller with unity feedback such that the closed-loop rise time is about 0.05 seconds, the overshoot is less than 5%, and the steady-state error is zero for a step command.

Determining ψ

We use Matlab for the design.[9] First, we see that the plant is Type 0, so integral compensation is required to yield zero steady-state error for a step command and therefore a PID controller is a good choice. Second, we must determine what the specified transient response criteria imply for the locations of our closed-loop poles. Let one of these desired pole locations be called ψ. The transient response performance criteria are as follows.

9. See ricopic.one/control/source/pid_controller_design_example_01.m for the source.

```
Tr = .05; % sec ... spec rise time
OS = 5; % percent ... spec overshoot max
```

The second-order approximation from Chapter trans tells us, via Fig. exact.2, that the rise time specification implies a specific ratio between ω_n and the implicit function $f(\zeta)$ defined in Fig. exact.2:

$$\begin{aligned} T_r\omega_n &= f(\zeta) \Rightarrow && (6a)\\ T_r &= \frac{f(\zeta)}{\omega_n} && (6b)\\ &= 0.05. && (spec)\end{aligned}$$

The minimum angle is determined from the overshoot specification via the relations

$$\angle\psi = \pi - \arccos\zeta \quad \text{and} \qquad (7)$$

$$\zeta = \frac{-\ln(\%OS/100)}{\sqrt{\pi^2 + \ln^2(\%OS/100)}}. \qquad (8)$$

```
zeta = -log(OS/100)/sqrt(pi^2+log(OS/100)^2)
psi_angle_min = pi - acos(zeta)
```

```
zeta =

    0.6901


psi_angle_min =

    2.3324
```

With ζ in-hand, we use Fig. exact.2 to determine $f(\zeta)$ and apply Eq. 6a to determine $|\psi| = \omega_n$:

```
psi_mag = 2.1/Tr % also = omega_n
```

```
psi_mag =

    42
```

So the target magnitude $|\psi|$ and minimum angle $\angle\psi$ are determined. Let's convert this to rectangular coordinates:

```
psi_real = psi_mag*cos(psi_angle_min);
psi_imag = psi_mag*sin(psi_angle_min);
psi = psi_real+i*psi_imag
```

```
psi =

 -28.9845 +30.3957i
```

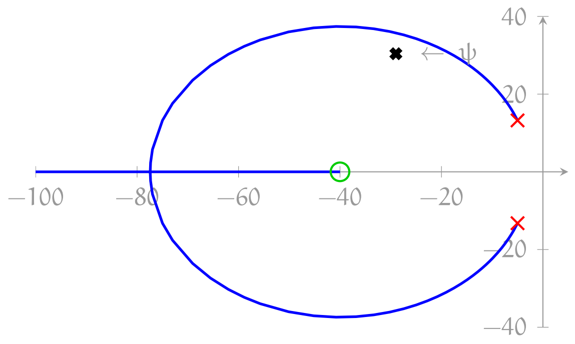

Figure PID.1: root locus without compensation.

So this is our design target for the dominant closed-loop poles. As usual, it depends on the second-order approximation, so we will need to simulate to determine the actual performance.

P control

We design a proportional controller that gets us as close as possible to ψ. The root locus is shown in Figure PID.1.

```
G = tf([1,40],[1,10,200]);
figure
rlocus(G);hold on
plot(psi,'kx','MarkerSize',5,'LineWidth',2)
text(real(psi),imag(psi),' \leftarrow \psi')
```

Although we cannot get quite to ψ on the root locus, we can at least try to meet our %OS specification by choosing a conservative gain of about

$$K_1 = 64. \tag{9}$$

Let's construct the compensator and corresponding closed-loop transfer function G_P for gain control.

```
K1 = 64;
G_P = feedback(K1*G,1); % closed loop transfer func
```

Derivative compensation

Now, we try cascade derivative compensation with compensator

$$K_2(s - z_c). \tag{10}$$

For now, we set $K_2 = 1$. From Equation 4, we compute the compensator zero angle contribution

$$\theta_c = \pi - \angle G(\psi).$$

```
theta_c = pi - angle(evalfr(G,psi));
disp(sprintf('theta_c = %0.3g deg',rad2deg(theta_c)))
```

```
theta_c = 13.1 deg
```

We try using the zero compensator:

$$K_2(s - z_c). \tag{11}$$

where

$$z_c = \mathrm{Re}(\psi) - |\mathrm{Im}(\psi)|/\tan(\theta_c) \tag{12}$$

```
z_c = real(psi) - abs(imag(psi))/tan(theta_c);
disp(sprintf('z_c = %0.3g',z_c))
```

```
z_c = -159
```

Let's construct the compensator sans tuned gain K_2 and construct the corresponding root locus.

```
C_sans = zpk(z_c,[],1);
figure
rlocus(K1*C_sans*G);hold on
plot(psi,'kx','MarkerSize',5,'LineWidth',2)
text(real(psi),imag(psi),' \leftarrow \psi')
```

By construction, the resulting root locus of Fig. PID.2 intersects ψ. The corresponding gain is, from Eq. 2 (or we could use the data cursor),

$$K_2 = \frac{1}{|K_1(\psi - z_c)G(\psi)|}. \tag{13}$$

Let's compute it, the controller C_{PD}, and the closed-loop transfer function G_{PD}.

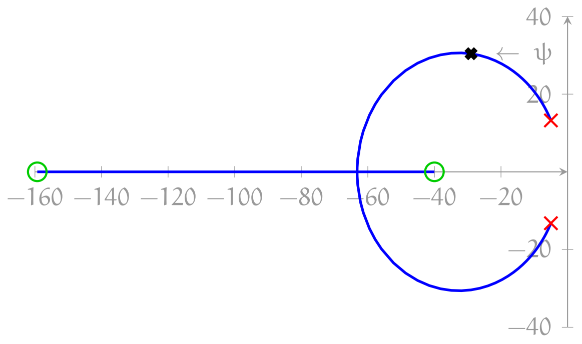

Figure PID.2: root locus with derivative compensation.

```
K2 = 1/abs(evalfr(K1*C_sans*G,psi))
C = K1*K2*C_sans;
G_PD = feedback(C*G,1);
```

```
K2 =

    0.0053
```

Simulate Our placement of the ψ depended on the second-order approximation's accuracy, which in this case is questionable, due to the proximity of a third closed-loop pole. In any case, we simulate the step response to test the efficacy of the PD controller design and to compare it with the P controller.

```
t_a = linspace(0,.7,200); % s ... sim time
y_P = step(G_P,t_a); % P controlled step response
y_PD = step(G_PD,t_a); % PD controlled step response
```

```
figure
plot(t_a,y_P);
hold on;
plot(t_a,y_PD);
xlabel('time (s)');
ylabel('step response');
grid on
legend('P control','PD control','location','southeast');
```

The responses, shown in Figure PID.3, suggest the PD controller is probably not meeting the transient performance specifications. Let's use `stepinfo` to compute more accurate transient

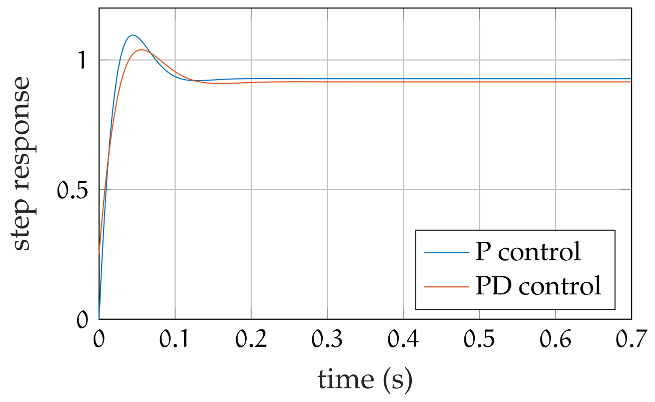

Figure PID.3: step responses for proportional and proportional-derivative controllers.

response characteristics of the PD-controlled system.

```
si_PD = stepinfo(y_PD,t_a);
disp(sprintf('rise time: %0.3g',si_PD.RiseTime))
disp(sprintf('percent overshoot: %0.3g',si_PD.Overshoot))
```

```
rise time: 0.022
percent overshoot: 13.6
```

It's too fast and overshoots too much. Our second-order approximation that led to this design is not very accurate. Before we start tuning this design, let's fix the steady-state error by including an integral compensator. Perhaps this compensator's zero can "help" us with our fix.

Integral compensation

The integral compensator has its usual form

$$K_3 \frac{s - z_I}{s}. \tag{14}$$

We're less concerned than usual about affecting our transient response with this compensator because we need some help doing so in any case. Let's start with $z_I = -5$.

```
z_I = -5;
C_I_sans = zpk(z_I,0,1);
```

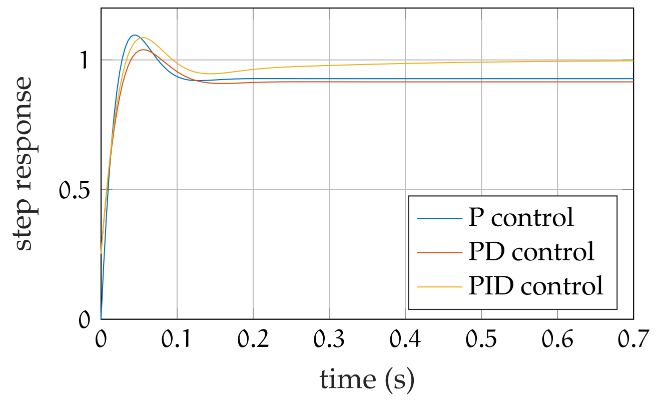

Figure PID.4: step responses for proportional and proportional-derivative controllers.

Now, a root locus wouldn't be particularly helpful here, since our second-order approximation is poor and getting worse by the minute. Instead, we proceed directly to simulation.

```
G_PID1 = feedback(C_I_sans*C*G,1);
y_PID1 = step(G_PID1,t_a); % PID controlled step response
```

The responses, shown in Figure PID.4, show that the steady-state error has improved with integral compensation, and so has the transient response, but not enough.

```
figure
plot(t_a,y_P);
hold on;
plot(t_a,y_PD);
hold on;
plot(t_a,y_PID1);
xlabel('time (s)');
ylabel('step response');
grid on
legend('P control','PD control','PID control',...
  'location','southeast');
```

Let's take a look at the `stepinfo`.

```
si_PID = stepinfo(y_PID1,t_a);
disp(sprintf('rise time: %0.3g',si_PID.RiseTime))
disp(sprintf('percent overshoot: %0.3g',si_PID.Overshoot))
```

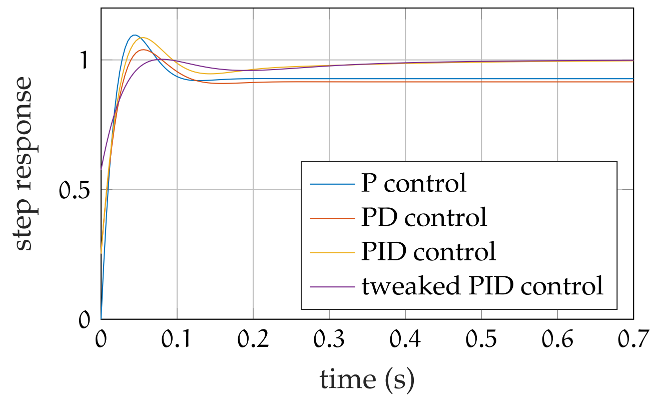

Figure PID.5: step responses for proportional and proportional-derivative controllers.

```
rise time: 0.0246
percent overshoot: 8.98
```

It's still too fast and overshoots too much. At this point we can directly tweak our compensator zeros and the overall gain to try to meet our specifications.

```
K3 = 4;
z_D = -25;
z_I = -8;
C_D_sans = zpk(z_D,[],1);
C_I_sans = zpk(z_I,0,1);
G_PID2 = feedback(K1*K2*K3*C_I_sans*C_D_sans*G,1);
y_PID2 = step(G_PID2,t_a); % PID controlled step response
```

```
figure
plot(t_a,y_P);hold on;
plot(t_a,y_PD);hold on;
plot(t_a,y_PID1);hold on;
plot(t_a,y_PID2);
xlabel('time (s)');
ylabel('step response');
grid on
legend('P control','PD control','PID control',...
  'tweaked PID control','location','southeast');
```

```
si_PID = stepinfo(y_PID2,t_a);
disp(sprintf('rise time: %0.3g',si_PID.RiseTime))
disp(sprintf('percent overshoot: %0.3g',si_PID.Overshoot))
```

```
rise time: 0.0422
percent overshoot: 0.391
```

It turns out to be difficult to meet both specifications, even with the massively tweaked controller design. Whenever one attempts to increase the rise time, the overshoot also increases. However, we've done a serviceable job, considering.
Compute the PID gains.

```
KP = -K1*K2*K3*(z_D+z_I)
KI = K1*K2*K3*z_D*z_I
KD = K1*K2*K3
```

```
KP =

   44.8013


KI =

  271.5228


KD =

    1.3576
```