

Christopher Lum
lum@uw.edu

Lecture 08c

Designing a PID Controller Using the Root Locus Method



Lecture is on YouTube

YouTube video entitled 'Designing a PID Controller Using the Root Locus Method' covering this is located at https://youtu.be/Hk6YBO_A_PU.

Outline

- Introduction
 - PI Controller
 - PID Controller
 - P, I, pseudo-D Controller
-

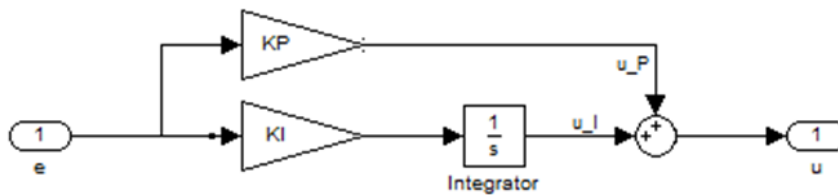
Introduction

In a previous lecture, we discussed using iterative, online methods such as Ziegler-Nichols (https://youtu.be/n829SwSUZ_c) to tune a PID controller. A more mathematical approach to designing a PID controller is to use the root locus method.

PI Controller

We saw previously (Using the Control System Designer in Matlab - <https://youtu.be/RPzFLzKkQGs>) that a proportional controller can easily be designed using the root locus technique. The Control System Designer can help rapidly iterate on the design until satisfactory performance is achieved. The root locus technique can also be used to design more complicated controllers such as a PI controller. We will later investigate methods to choose all the gains simultaneously, but for now, let us concentrate on a PI controller.

Consider a PI controller



The goal is to now choose K_P and K_I so that the overall system has desired performance. Consider the control law

$$u(t) = u_P(t) + u_I(t)$$

$$= K_P e(t) + K_I \int_0^t e(\tau) d\tau$$

$$U(s) = K_P E(s) + K_I \frac{1}{s} E(s)$$

$$= \left(K_P + K_I \frac{1}{s} \right) E(s)$$

$$= \left(\frac{K_P s}{s} + \frac{K_I}{s} \right) E(s)$$

$$= \left(\frac{K_P s + K_I}{s} \right) E(s)$$

$$C(s) = \frac{U(s)}{E(s)} = \frac{K_P s + K_I}{s} = \frac{K_P(s + K_I/K_P)}{s} \quad (\text{Eq.2})$$

So we see that a PI controller is nothing more than a real zero and a pole at the origin. The proportional gain is the coefficient of the s in the numerator and the integral gain is the constant in the numerator. Recall that the original design question was how to choose K_P and K_I . We now see that the equivalent question is how to choose the location of the real zero and the overall gain in the zpk transfer function. As a corollary, the original problem had 2 degrees of freedom (K_P and K_I) and this equivalent problem also has 2 degrees of freedom (location of real zero and overall gain).

Note: It may be desirable to create a Matlab script which can be passed a transfer function $C(s)$ and it will automatically extract the K_P and K_I gains (we will do this later when the controller structure is more complicated).

We can use root locus techniques to place the closed loop poles of the system in desirable locations. We need to investigate where is “desirable”. We can use the metrics we discussed previously to choose “desirable” locations.

Example: Designing a PI Controller to Meet Specified Performance Requirements

If you have not viewed the ‘Using the Control System Designer in Matlab’ video you should do this now before proceeding further.

Suppose that we would like to control the position of the DC motor. Recall that the transfer function for position is given as

$$G_P(s) = \frac{\theta(s)}{V_o(s)} = G_V(s) \frac{1}{s} = \frac{46163}{(s^2 + 1021s + 4845)s} = \frac{46163}{s^3 + 1021s^2 + 4845s}$$

Recall that in the previous example where we used proportional control only, the requirements were

- Less than 40% overshoot
- Rise time of less than 1 second
- Settling time of less than 3 seconds
- Gain margin of 20 dB, Phase margin of 30 deg
- Bandwidth of greater than 5 rad/s

In the 'Using the Control System Designer in Matlab' video we showed how to synthesize a controller that met these requirements only using proportional control with $K_P = 0.55$, the system was subject to steady state error if there were any nonlinearities such as friction or disturbance weights (**show video**).

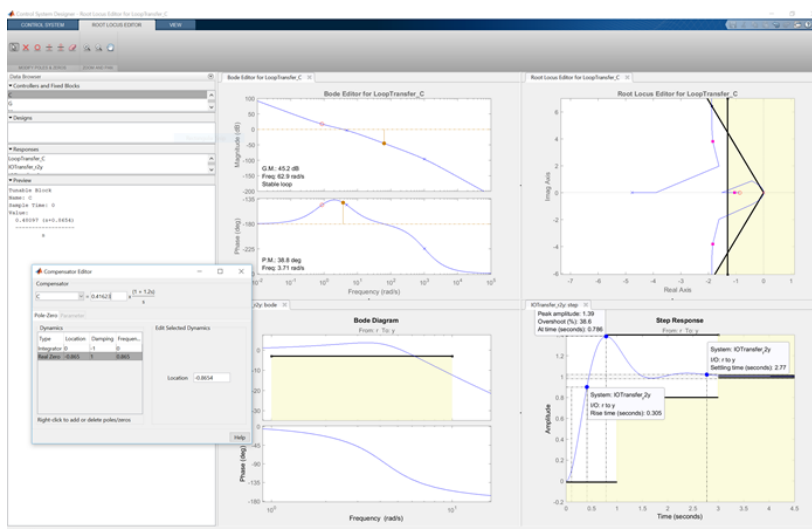
To combat this problem, we now add the requirement that the system be robust to disturbances and have no steady state error.

- Zero steady state error
- Robust to external disturbances

Suppose that we want to meet some requirements using a PI controller.

The design process was covered in the video 'Using the Control System Designer in Matlab'.

The Control System Designer for this scenario is shown below (Note the drawing/rendering error/artifact in the root locus, it should be symmetrical but this is merely due to how Matlab decides to connect discrete points on the root locus)



Note that the percent overshoot and settling time constraints in the root locus are only valid for 2nd order systems. Since this system has multiple poles that are relatively close together, it is not guaranteed to meet these requirements if we design using the root locus only. It is therefore advisable to also investigate the step response and use this to validate that we meet the percent overshoot and settling

time requirements.

We can export our designed compensator to the Matlab workspace

$$C(s) = \frac{0.41623(s+0.8654)}{s} = \frac{0.41623s+0.360205}{s}$$

Comparing with the previous expression for $C(s)$ (Eq.2), we see that

$$K_P = 0.41623$$

$$K_I = 0.360205$$

PID Controller

Extending this concept, we see that we can write the control law for a PID controller as

$$u(t) = u_P(t) + u_I(t) + u_D(t)$$

$$= K_P e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{de(t)}{dt}$$

$$U(s) = K_P E(s) + K_I \frac{1}{s} E(s) + K_D s E(s)$$

$$= \left(K_P + K_I \frac{1}{s} + K_D s \right) E(s)$$

$$= \left(\frac{K_P s}{s} + \frac{K_I}{s} + \frac{K_D s^2}{s} \right) E(s)$$

$$= \left(\frac{K_P s + K_I + K_D s^2}{s} \right) E(s)$$

$$C(s) = \frac{U(s)}{E(s)} = \frac{K_P s + K_I + K_D s^2}{s} \quad \text{(Eq.3)}$$

So we see that a pure PID controller is nothing more than

1. pole at origin
2. pair of zeros (potentially complex)

Note that as we discussed previously in our lecture on ‘Practical Implementation Issues with a PID Controller’, this is an improper (non-causal) transfer function due to the pure derivative. Nevertheless, the we can use the root locus technique to design a PID controller using similar techniques as above. Note that if $G(s)$ is strictly proper (has strictly more poles than zeros), then the loop transfer function $G(s) C(s)$ will be at least proper (same number of poles and zeros) so the root locus plotting/behaviour techniques we’ve developed in the past will still apply.

Further note that the Control System Designer tool in Matlab will allow you to design an improper

compensator. However since a pure PID controller is not practical/implementable, we will instead focus on designing a PID w/ pseudo-derivative controller

P, I, pseudo-D Controller

We can now implement a P, I, pseudo-D controller using the Control System Designer. Recall that the control law for this is simply

$$C(s) = \frac{U(s)}{E(s)}$$

$$= K_P + K_D \frac{as}{s+a} + K_I \frac{1}{s}$$

$$Cs[s_] = KP + KD \frac{a s}{s + a} + \frac{KI}{s};$$

$$\text{num} = \text{Collect}[\text{Numerator}[\text{Together}[Cs[s]]], \{s, s^2\}];$$

$$\text{den} = \text{Denominator}[\text{Together}[Cs[s]]];$$

$$\text{Calt}[s_] = \frac{\text{num}}{\text{den}}$$

$$\frac{a KI + (KI + a KP) s + (a KD + KP) s^2}{s (a + s)}$$

So we see that we can write this as

$$C(s) = \frac{(a K_D + K_P) s^2 + (a K_P + K_I) s + a K_I}{s(s+a)}$$

So we see that a P, I, pseudo-D controller is nothing more than

1. pole at origin
2. pole at $-a$
3. pair of zeros (potentially complex)

We can use the Control System Designer to design a compensator. This will have the form of

$$C(s) = \frac{\alpha s^2 + \beta s + \gamma}{s(s+\delta)}$$

So we see that we can simultaneously solve for K_P , K_I , K_D , and a as a function of α , β , γ , and δ

1. $a K_D + K_P = \alpha$ (coefficient of s^2 in numerator must match)
2. $a K_P + K_I = \beta$ (coefficient of s in numerator must match)
3. $a K_I = \gamma$ (constant in numerator must match)
4. $a = \delta$ (real pole must match)

`Solve[{a KD + KP == α, a KP + KI == β, a KI == γ, a == δ}, {KP, KI, KD, a}]`

$$\left\{ \left\{ KP \rightarrow \frac{-\gamma + \beta \delta}{\delta^2}, KI \rightarrow \frac{\gamma}{\delta}, KD \rightarrow \frac{\gamma - \beta \delta + \alpha \delta^2}{\delta^3}, a \rightarrow \delta \right\} \right\}$$

So we have

$$K_P = \frac{-\gamma + \beta \delta}{\delta^2}$$

$$K_I = \frac{\gamma}{\delta}$$

$$K_D = \frac{\gamma - \beta \delta + \alpha \delta^2}{\delta^3}$$

$$a = \delta$$

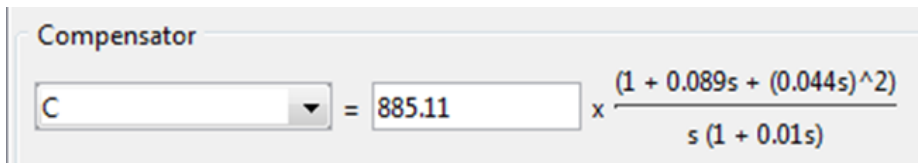
To assist in extracting the PID gains, I provide the function `ExtractPIDGainsFromSecondOrderTransferFunction.m` which can be downloaded from https://github.com/clum/UWMatlab/blob/master/UWMatlab/Controls_Functions/ExtractPIDGainsFromSecondOrderTransferFunction.m.

Example

P, I, Pseudo-D controller via the Control System Designer

-Note that the zeros act like magnets.

Suppose that we obtain from the Control System Designer a compensator as shown below



$$C = 885.11 \times \frac{(1 + 0.089s + (0.044s)^2)}{s(1 + 0.01s)}$$

$$C_{sisotool}[s_] = 885.11 \frac{(0.044 s)^2 + 0.089 s + 1}{s (1 + 0.01 s)}$$

$$\frac{885.11 \times (1 + 0.089 s + 0.001936 s^2)}{(1 + 0.01 s) s}$$

We can manipulate this to the form of

$$C(s) = \frac{\alpha s^2 + \beta s + \gamma}{s(s + \delta)}$$

Using our numerical example

$$\begin{aligned} C(s) &= \frac{885.11 \times (1 + 0.089 s + 0.001926 s^2)}{s(1 + 0.01 s)} \\ &= \frac{885.11 \times (1 + 0.089 s + 0.001926 s^2)}{0.01 s \left(\frac{1}{0.01} + \frac{0.01}{0.01} s \right)} \end{aligned}$$

$$= \frac{88511 \cdot (1 + 0.089s + 0.001926s^2)}{s(s+100)}$$

$$= \frac{(88511 + 7877.48s + 170.472s^2)}{s(s+100)}$$

$$= \frac{170.472s^2 + 7877.48s + 88511}{s(s+100)}$$

$$= \frac{\alpha s^2 + \beta s + \gamma}{s(s+\delta)}$$

where $\alpha = 170.472$

$\beta = 7877.48$

$\gamma = 88511$

$\delta = 100$

Using our previous expressions

$\alpha = 170.472;$

$\beta = 7877.48;$

$\gamma = 88511;$

$\delta = 100;$

$$K_P = \frac{-\gamma + \beta \delta}{\delta^2} \quad // \quad N$$

$$K_I = \frac{\gamma}{\delta} \quad // \quad N$$

$$K_D = \frac{\gamma - \beta \delta + \alpha \delta^2}{\delta^3} \quad // \quad N$$

$a = \delta \quad // \quad N$

69.9237

885.11

1.00548

100.

Workflow

So the procedure for designing a PID controller via the Control System Designer is

1. Create mathematical model of system.
2. Obtain transfer function between desired input and desired output, $G(s)$
3. Load this into the Control System Designer.
4. Add a compensator of the form $C(s) = \frac{\alpha s^2 + \beta s + \gamma}{s(s+\delta)}$

5. Modify parameters α , β , γ , and δ until desirable root locus is achieved.
6. Export this compensator to the Matlab workspace.
7. Use your function `ExtractPIDGainsFromSecondOrderTransferFunction.m` to obtain PID gains.
8. Add linear bells and whistles (ie additional derivative filtering). Note that this will change the actual root locus from the root locus derived in step 5. Further note that if you wanted to, you could have accounted for this in your `sisotool` analysis.
9. Add non-linear bells and whistles (ie integral anti-wind up to your controller). Note that you cannot account for this behavior in `sisotool` because it is a nonlinear phenomenon.
10. Test and verify controller performance.