PI Controller for OPV Costas Loop

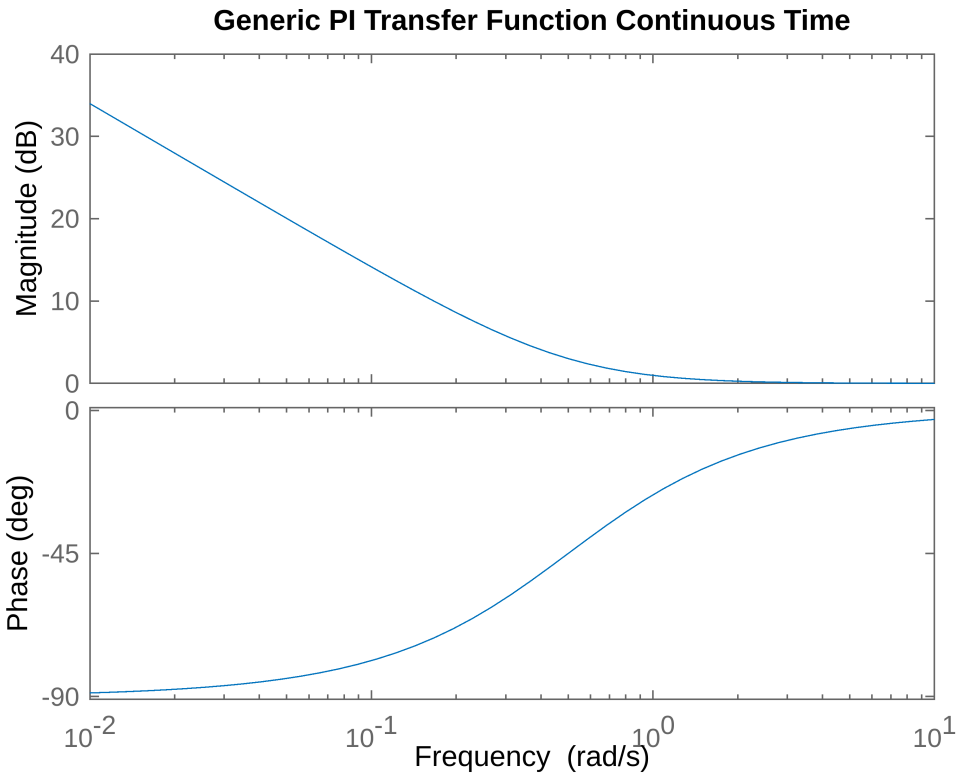
Version 2.0 Abraxas3d 18 November 2024

First, some examples from Mathworks and a brief explanation of what we are doing.

First, let's draw a Bode Diagram and Frequency Plot for a continuous time PI Controller.

```
Kp = 1;
Ki = 0.5;
pi_tf = tf([Kp, Ki], [1, 0]);

bode(pi_tf);
title('Generic PI Transfer Function Continuous Time')
```



Next, let's set up and draw a Bode Diagram and Frequency Plot for Discrete time PI Controller running at our sample rate. Our design is running at Fs of 61.44 MHz, and we discard 24 out of 25 samples to reduce the rate for the controller to 2,457,600 Hz. Our bitrate is 54,200 Hz. 433,600 Hz is our IF. F2 is IF + bitrate/4.

```
discard_ratio = 25;
Fs = 61.44e6/discard_ratio;
bitrate = 54200;
Fc = (bitrate/4)*32;
F2 = Fc + bitrate/4;
```

Loop bandwidth is the range that our control system can respond to the signal we're trying to follow. The recommendation is from frequency/20 to frequency/200. The larger the loop bandwidth, the wider range of frequencies that we can respond to, but the more noise we allow in. The smaller the loop bandwidth, the smaller the range that we can respond to, but we reduce the noise and increase performance within the loop. Let's start out with a large loop bandwidth. We can get fancy later and reduce the loop bandwidth after we pull in our signal, if it turns out that we need to or want to. So, Bn is set to the fraction of the frequency or rate that we're using, which is F2. LB is Bn of F2. We write down the period of F2 and Kv, which is used to calculate one of the time constants. We set Kd to equal pi. Kd is our phase error detector gain.

```
Bn = 1/20; %fraction of loop bandwidth
LB = F2*Bn; %rule of thumb estimate for loop bandwidth in Hz
T = 1/F2; %period of F2 and used in transfer function creation
Kv = 1/T; %used in NCO transfer function
Kd = pi; %phase detector gain
Kd = (Fs/bitrate)*pi; %updated phase detector gain (matthew)
```

Now we're going to hit the textbooks and calculate zeta, alpha, our 3dB loop bandwidth in radians/secon, our natural frequency, and two time constants.

```
zeta = 1/sqrt(2);
alpha = 1 - 2*power(zeta,2);
w3db = 2*pi*F2*Bn; %target 3dB loop bandwidth rads/sec
w3db_kHz = w3db/(2*pi*1000); %target 3dB loop bandwidth kHz
wn = w3db/(sqrt(alpha + sqrt(power(alpha,2) + 1)));
tau1 = (Kv*Kd)/(power(wn,2));
tau2 = (2*zeta)/wn;
```

With this we are going to create several transfer functions.

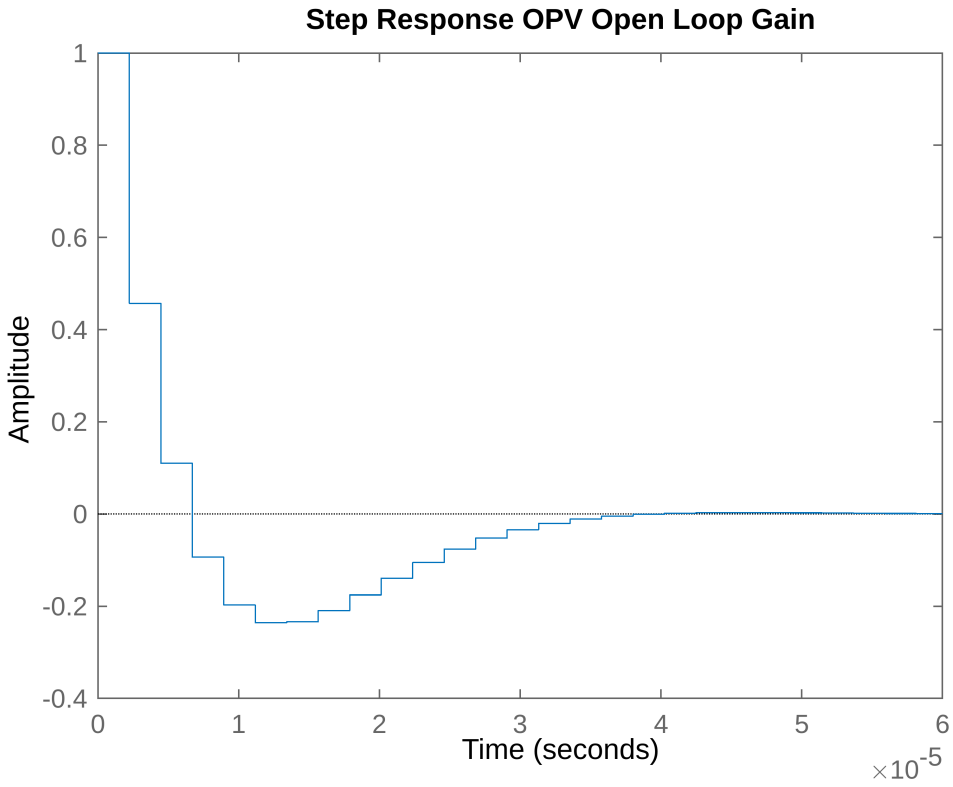
```
H_lf = tf([(T+tau2)/tau1, -tau2/tau1],[1,-1],T); %PI loop filter
H_ped = Kd; %just the phase detector gain
H_nco = tf([Kv*T],[1,-1],T); %NCO transfer function
```

Now we combine them like a pokemon team!

```
open_loop_gain = H_ped*H_nco*H_lf;
```

Now we see what the step response is for the whole thing.

```
step(1/(1+open_loop_gain));
title('Step Response OPV Open Loop Gain')
```



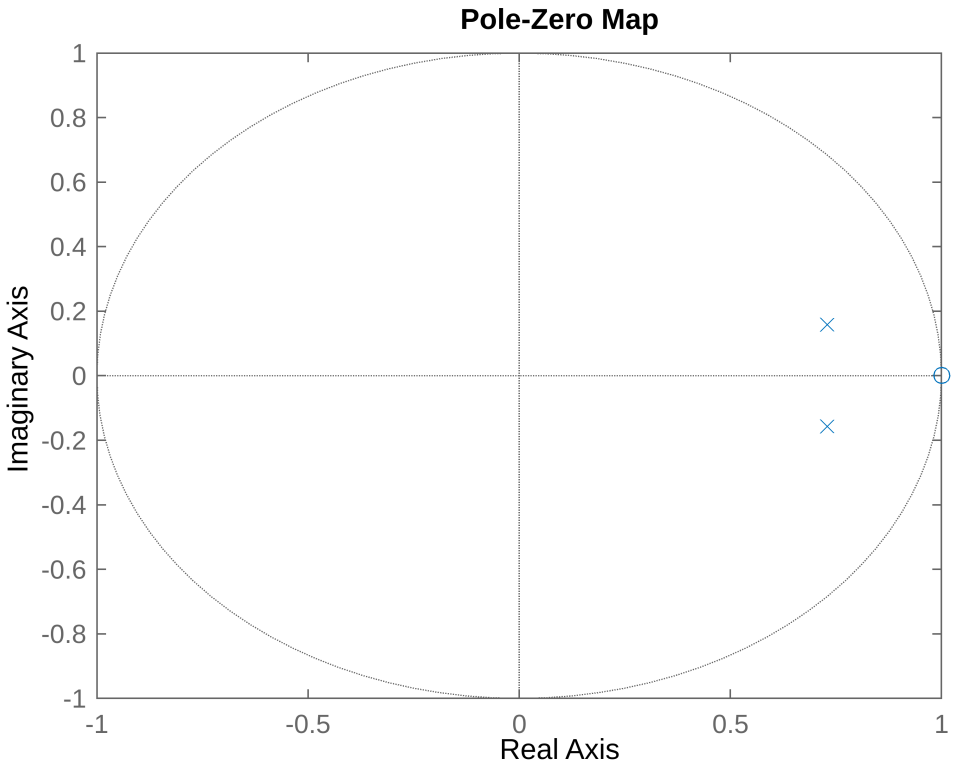
How long does it take the loop to respond to an impulse? Here is an estimate.

```
number_symbols = (0.35*F2)/Bn
```

```
number_symbols = 3130050
```

We can show the locations of the poles!

```
pzmap(1/(1+open_loop_gain));
```



And now we can calculate Kp and Ki.

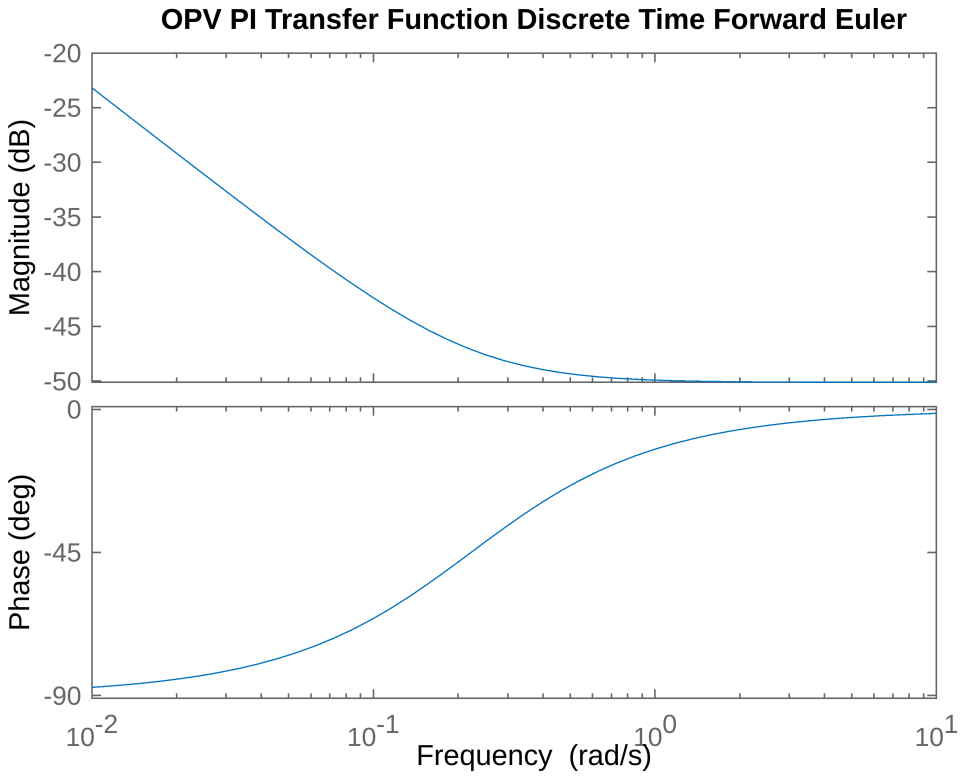
```
Kp = tau2/tau1
```

```
Kp = 0.0031
```

```
Ki = T/tau1
```

```
Ki = 6.9285e-04
```

```
OPV_pi_tf = pid(Kp, Ki, 'Ts', T, 'Iformula','ForwardEuler');
bode(OPV_pi_tf);
title('OPV PI Transfer Function Discrete Time Forward Euler')
```


OPV PI Transfer Function Discrete Time Forward Euler
Magnitude (dB)
-20
-25
-30
-35
-40
-45
-50
Phase (deg)
0
-45
-90
10^-2
10^-1
10^0
10^1
Frequency (rad/s)