Joshua Andrews -
ECE414 - PID Tuner HW

Table of Contents

Part 1, P control

Testing the PID tuner and PID tune function for a P controller

```
clear all; clc;

G = tf(40, [1 30 200])
```

```
G =

         40
  ----------------
  s^2 + 30 s + 200

Continuous-time transfer function.
```

Find the baseline matlab generated P controller

```
Cp = pidtune(G, 'P');
```

Run PID Tuner with plant tf. and the baseline Cp

Using the GUI the response time can be changed to alter the Kp parameter. Because we are looking at minimizing %OS and controller effort, the contoller effort can be plotted alongside the step response of

the system using the tools at the top of the GUI. The controller parameters is displayed by clicking the button at the top and the controller can be exported to the workspace.

Finding the minimum controller effort

Using the slider, the lowest %OS and controller effort occured when the response time was as slow as possible. The lowest %OS was found by watching the parameters while the lowest controller effort was found by displaying the peak by right clicking on the controller effort graph. After tuning export it to the workspace as C.

```
Tune = pidTuner(G, Cp);
waitfor(Tune);   %Wait here until controller is exported and pidtune
 GUI closed
```

```
Warning: Error occurred while executing the listener callback for
 event
ActionPerformed defined for class toolpack.component.TSButton:
Error using pidtool.desktop.pidtuner.gc.RichSlider/atomicSet (line
 558)
Assertion failed.

Error in pidtool.desktop.pidtuner.gc.RichSlider/shiftLogRange (line
 720)
            this.atomicSet(min_,max_,val_,[],[],[],[]);

Error in pidtool.desktop.pidtuner.gc.RichSlider/
rangeDownButtonCallback (line
692)
                outfactor = this.shiftLogRange(factor);

Error in
pidtool.desktop.pidtuner.gc.RichSlider>@(varargin)this.rangeDownButtonCallback(var
(line 119)
            addlistener(this.RangeDownTPComponent, 'ActionPerformed',
            @this.rangeDownButtonCallback);

Error in toolpack.component.TSButton>LocalActionPerformed (line 141)
obj.notify('ActionPerformed')

Error in hgfeval (line 62)
        feval(fcn{1},varargin{:},fcn{2:end});

Error in javaaddlistener>cbBridge (line 52)
    hgfeval(response, java(o), e.JavaEvent)

Error in javaaddlistener>@(o,e)cbBridge(o,e,response) (line 47)
                         @(o,e) cbBridge(o,e,response));
```

Export the tuned controller

The best value of Kp was found to be 5 as it minimzed both parameters. The controller effort had a peak of 5 and the %OS was 2.84%

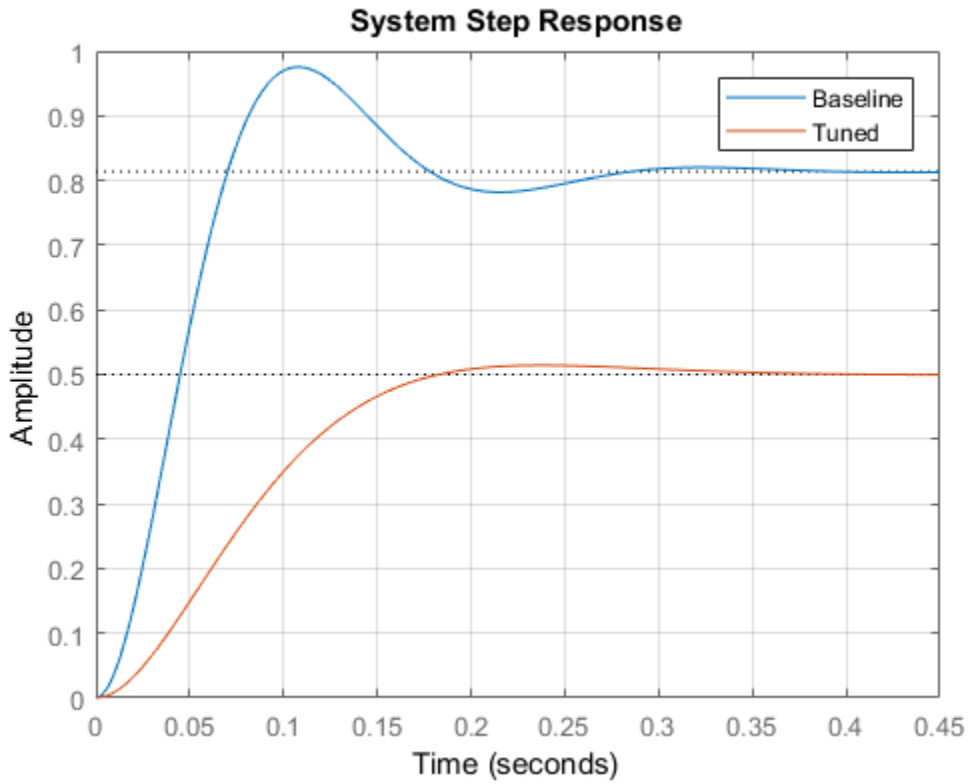
```
% Baseline controller effort and system transfer functions
```

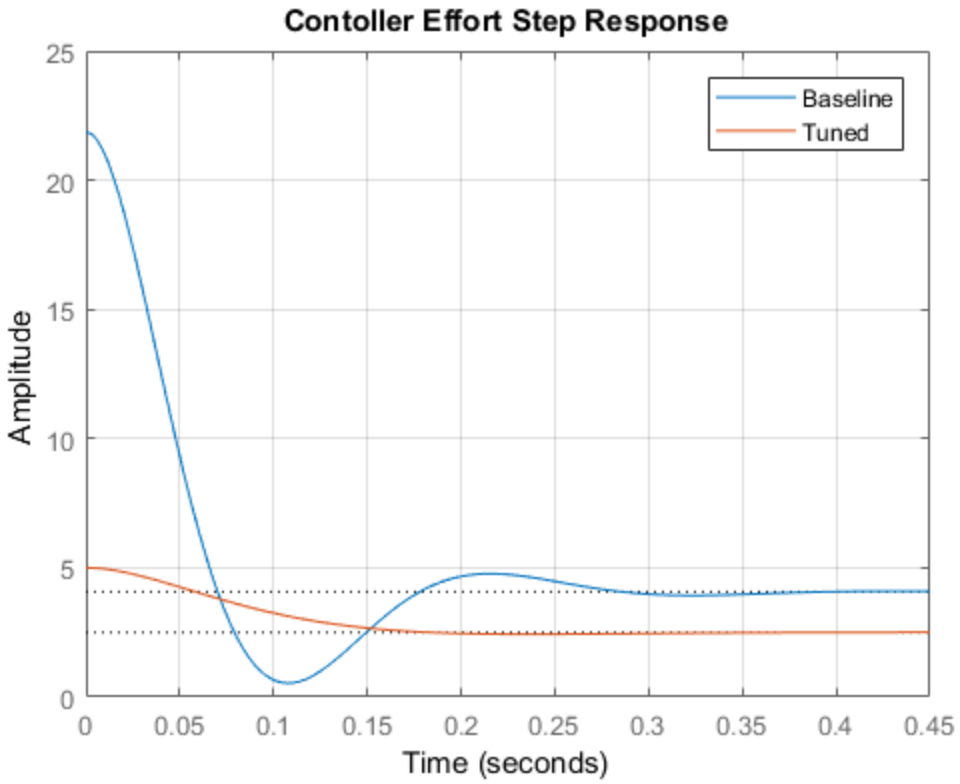
```
Up = Cp/(1+(Cp*G));
Tp = (Cp*G)/(1+(G*Cp));

% Tuned controller effort and system transfer functions
Utp = C/(1+(C*G));
Ttp = (C*G)/(1+(G*C));
```

Ploting the step responses

Using the transfer functins above the step responses we plotted for each

```
figure(1); clf;
step(Tp);
hold on;
step(Ttp);
grid on;
legend('Baseline', 'Tuned');
title('System Step Response');

figure(2); clf;
step(Up);
hold on;
step(Utp);
grid on;
legend('Baseline', 'Tuned');
title('Contoller Effort Step Response');
```





In the graphs above you can see that by only focusing on those two parameters we got a much better response with a longer rise time, which is still fairly low, with the tuned controller. The peak controller effort also drops from 21 to 5

Part 2 - PD Controller

By using the steps above we can do the same process for the P controller

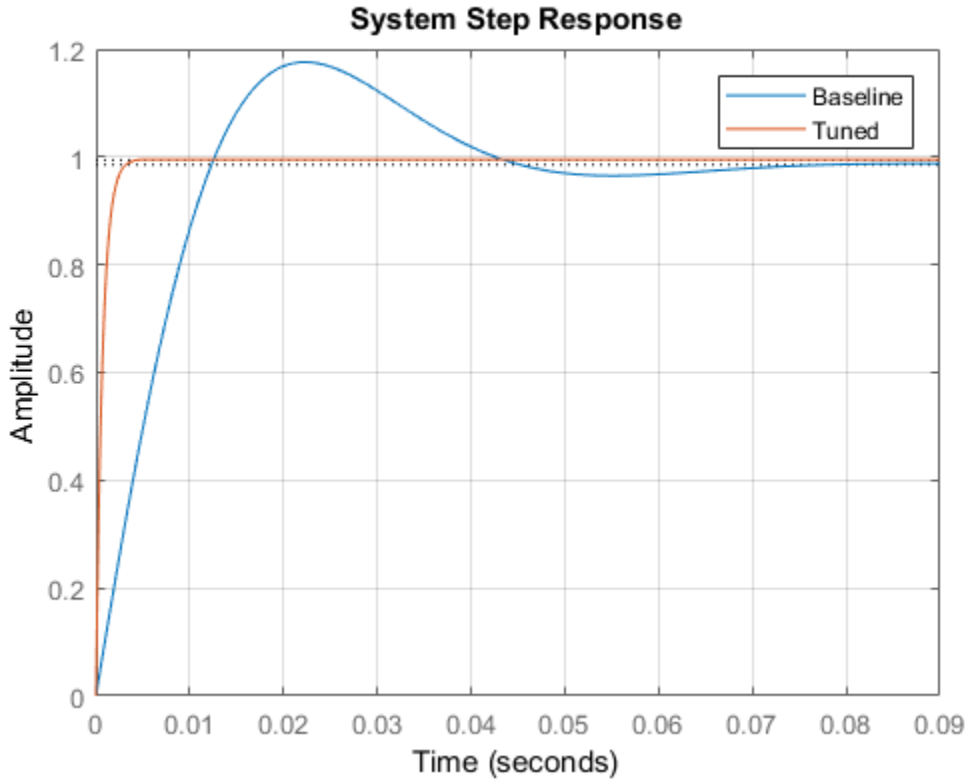
The controller effort transfer function is improper for any value of Kd other than 0. This results in an ideal controller that is not implementable in the real world. The controller would have to supply impluse outputs and the plant would have to handle impulse inputs. But in order to get values for Kp and Kd, the controller was tuned based on %OS alone. This occured when both sliders were taken all the way to the right. (Fast response, robust transient).

```
Cpd = pidtune(G, 'PD');
Tune = pidTuner(G, Cpd);
waitfor(Tune);
disp('Parameters from PID Tune');
disp('Kp = ');
disp(C.Kp);
disp('Kd = ');
disp(C.Kd);

% Baseline controller effort and system transfer functions
disp('Baseline Controller Effort Transfer Function');
Upd = Cpd/(1+(Cpd*G))
```

```
disp('Baseline System Transfer Function');
Tpd = (Cpd*G)/(1+(G*Cpd))

% Tuned controller effort and system transfer functions
disp('Tuned Controller Effort Transfer Function');
Utpd = C/(1+(C*G))
disp('Tuned System Transfer Function');
Ttpd = (C*G)/(1+(G*C))


figure(3); clf;
step(Tpd);
hold on;
step(Ttpd);
grid on;
legend('Baseline', 'Tuned');
title('System Step Response');
```

```
Parameters from PID Tune
Kp =
  875.0647

Kd =
   35.3543

Baseline Controller Effort Transfer Function

Upd =

  2.656 s^3 + 419 s^2 + 1.071e04 s + 6.787e04
  -------------------------------------------
           s^2 + 136.2 s + 1.377e04

Continuous-time transfer function.

Baseline System Transfer Function

Tpd =

     106.2 s^3 + 1.676e04 s^2 + 4.285e05 s + 2.715e06
  -----------------------------------------------------
  s^4 + 166.2 s^3 + 1.806e04 s^2 + 4.405e05 s + 2.755e06

Continuous-time transfer function.

Tuned Controller Effort Transfer Function

Utpd =

  35.35 s^3 + 1936 s^2 + 3.332e04 s + 1.75e05
  -------------------------------------------
            s^2 + 1444 s + 3.52e04

Continuous-time transfer function.
```

```
Tuned System Transfer Function

Ttpd =

     1414 s^3 + 7.743e04 s^2 + 1.333e06 s + 7.001e06
  ----------------------------------------------------
  s^4 + 1474 s^3 + 7.873e04 s^2 + 1.345e06 s + 7.041e06

Continuous-time transfer function.
```



Using the parameters above the %OS was 0.07%. While the rise time could have been decreased much further by making the response time even faster, this resulted in a slightly larger overshoot and much more impractical values for Kd and Kp.

Part 3 PID Controller

By using the steps above we can do the same process for the PD controller

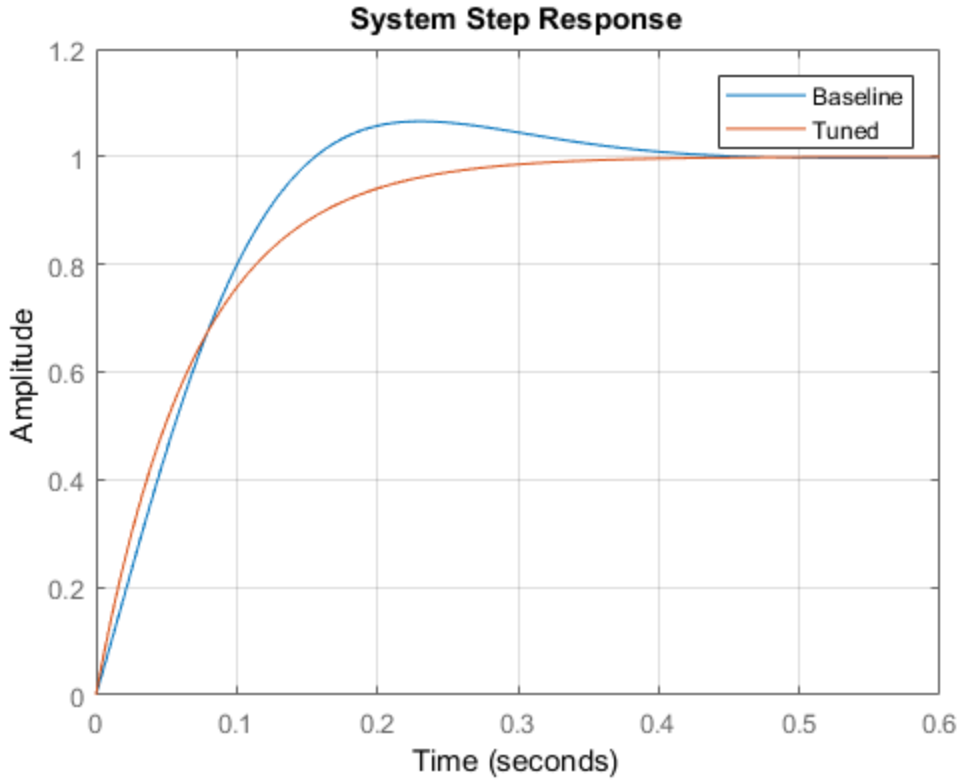
The controller effort transfer function is improper for any value of Kd or Ki other than 0. This results in an ideal controller that is not implementable in the real world. But in order to get values for Kp, Ki and Kd, the controller was tuned based on %OS alone and to have values for each paramter.

```
Cpid = pidtune(G, 'PID');
Tune = pidTuner(G, Cpid);
waitfor(Tune);
```

```
disp('Parameters from PID Tune');
disp('Kp = ');
disp(C.Kp);
disp('Ki = ');
disp(C.Ki);
disp('Kd = ');
disp(C.Kd);

% Baseline controller effort and system transfer functions
disp('Baseline Controller Effort Transfer Function');
Upid = Cpid/(1+(Cpid*G))
disp('Baseline System Transfer Function');
Tpid = (Cpid*G)/(1+(G*Cpid))

% Tuned controller effort and system transfer functions
disp('Tuned Controller Effort Transfer Function');
Utpid = C/(1+(C*G))
disp('Tuned System Transfer Function');
Ttpid = (C*G)/(1+(G*C))


figure(4); clf;
step(Tpid);
hold on;
step(Ttpid);
grid on;
legend('Baseline', 'Tuned');
title('System Step Response');
```

```
Parameters from PID Tune
Kp =
   10.6066

Ki =
   70.6743

Kd =
    0.3534

Baseline Controller Effort Transfer Function

Upid =

  0.221 s^5 + 16.56 s^4 + 439 s^3 + 4893 s^2 + 1.938e04 s
  ---------------------------------------------------------
           s^4 + 38.84 s^3 + 597.2 s^2 + 3876 s

Continuous-time transfer function.

Baseline System Transfer Function

Tpid =

    8.842 s^5 + 662.5 s^4 + 1.756e04 s^3 + 1.957e05 s^2 + 7.751e05 s
```

```
 ------------------------------------------------------------------
  s^6 + 68.84 s^5 + 1962 s^4 + 2.956e04 s^3 + 2.357e05 s^2 + 7.751e05
 s

Continuous-time transfer function.

Tuned Controller Effort Transfer Function

Utpid =

  0.3534 s^5 + 21.21 s^4 + 459.5 s^3 + 4242 s^2 + 1.413e04 s
  ---------------------------------------------------------
             s^4 + 44.13 s^3 + 624.3 s^2 + 2827 s

Continuous-time transfer function.

Tuned System Transfer Function

Ttpid =

    14.13 s^5 + 848.3 s^4 + 1.838e04 s^3 + 1.697e05 s^2 + 5.654e05 s

 ------------------------------------------------------------------
  s^6 + 74.13 s^5 + 2148 s^4 + 3.038e04 s^3 + 2.097e05 s^2 + 5.654e05
 s

Continuous-time transfer function.
```



Using the parameters above the %OS was 0%. The turned controller also had a similar rise time to the baseline but a much better %OS.

Part 4 PDF Controller

By using the steps above we can do the same process for the PD controller

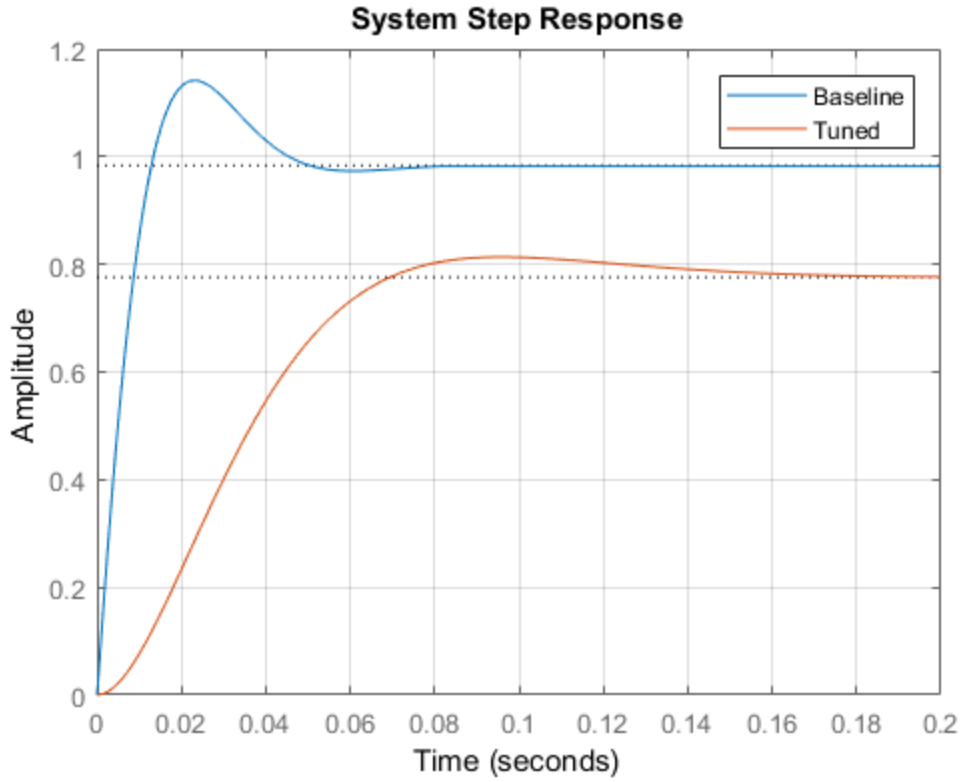
The controller effort transfer function is proper due to the introduced Tf term. Now the controller can be tuned with both constraints in mind. This controller is significantly harder to tune over the other though and so a design guideline I followed was to keep %OS under 5% and Peak controller effort under 50. This was found by slowing down the response time while increasing the robustness of the transient. The response time was set at 0.0763 and transient was set to 0.84. This resulted in a %0s = 4.85% and a peak controller effort of 48.9. The PID parameters are shown below.

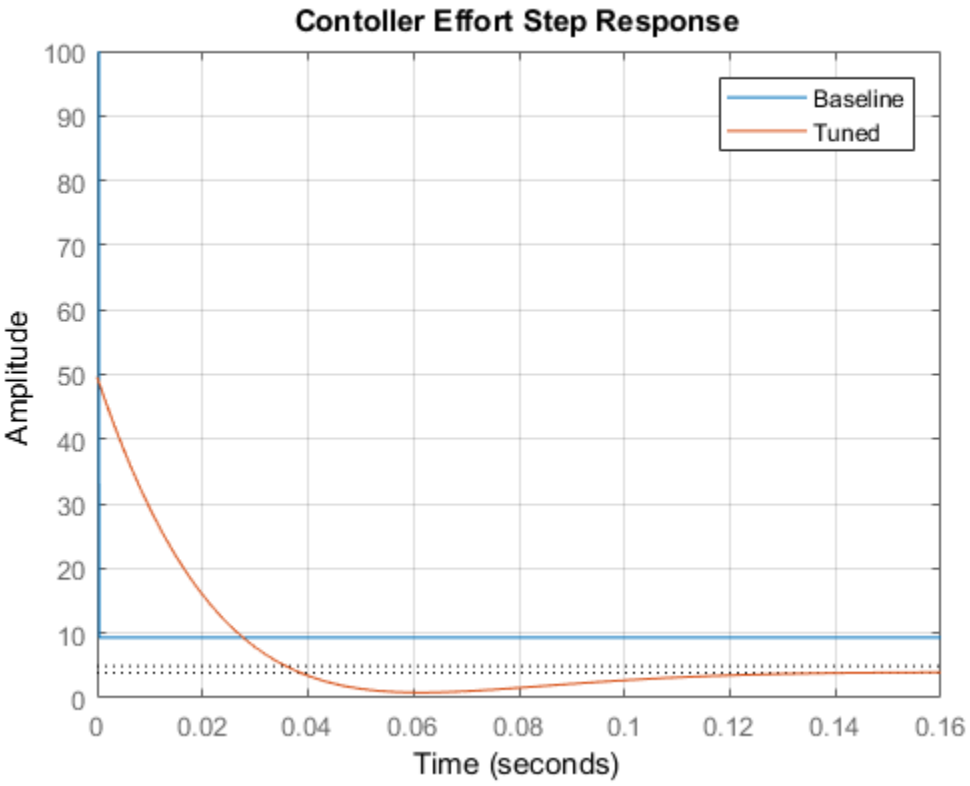
```
Cpdf = pidtune(G, 'PDF');
Tune = pidTuner(G, Cpdf);
waitfor(Tune);
disp('Parameters from PID Tune');
disp('Kp = ');
disp(C.Kp);
disp('Kd = ');
disp(C.Kd);
disp('Tf = ');
disp(C.Tf);

% Baseline controller effort and system transfer functions
```

```
disp('Baseline Controller Effort Transfer Function');
Updf = Cpdf/(1+(Cpdf*G))
disp('Baseline System Transfer Function');
Tpdf = (Cpdf*G)/(1+(G*Cpdf))

% Tuned controller effort and system transfer functions
disp('Tuned Controller Effort Transfer Function');
Utpdf = C/(1+(C*G))
disp('Tuned System Transfer Function');
Ttpdf = (C*G)/(1+(G*C))


figure(5); clf;
step(Tpdf);
hold on;
step(Ttpdf);
grid on;
legend('Baseline', 'Tuned');
title('System Step Response');

figure(6); clf;
step(Updf);
hold on;
step(Utpdf);
grid on;
legend('Baseline', 'Tuned');
ylim([0 100]);
title('Contoller Effort Step Response');
```

```
Parameters from PID Tune
Kp =
   17.3177

Kd =
    0.4303

Tf =
    0.0133

Baseline Controller Effort Transfer Function

Updf =

  4.653e04 s^4 + 7.583e08 s^3 + 1.013e11 s^2 + 2.509e12 s + 1.572e13
  ------------------------------------------------------------------
       s^4 + 3.235e04 s^3 + 2.64e08 s^2 + 3.812e10 s + 3.195e12

Continuous-time transfer function.

Baseline System Transfer Function

Tpdf =
```

```
    1.861e06 s^4 + 3.033e10 s^3 + 4.052e12 s^2 + 1.003e14 s +
 6.286e14

 ------------------------------------------------------------------------

  s^6 + 3.238e04 s^5 + 2.65e08 s^4 + 4.605e10 s^3 + 4.392e12 s^2

                                                      + 1.035e14 s +
 6.391e14

Continuous-time transfer function.

Tuned Controller Effort Transfer Function

Utpdf =

  49.66 s^4 + 6524 s^3 + 2.588e05 s^2 + 3.942e06 s + 1.957e07
  ------------------------------------------------------------
    s^4 + 180.3 s^3 + 1.235e04 s^2 + 4.009e05 s + 5.043e06

Continuous-time transfer function.

Tuned System Transfer Function

Ttpdf =

        1986 s^4 + 2.61e05 s^3 + 1.035e07 s^2 + 1.577e08 s + 7.827e08

 ----------------------------------------------------------------------------

  s^6 + 210.3 s^5 + 1.796e04 s^4 + 8.074e05 s^3 + 1.954e07 s^2 +
 2.315e08 s

                                                                  +
 1.009e09

Continuous-time transfer function.
```


System Step Response
Baseline
Tuned
Amplitude
Time (seconds)
0
0.2
0.4
0.6
0.8
1
1.2
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18


Contoller Effort Step Response
Baseline
Tuned
Amplitude
Time (seconds)
0
10
20
30
40
50
60
70
80
90
100
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16

There may be a better solution to this problem as to minizing both constraints however the tuned controller had a much lower controller effort over the baseline, pratical values of Kp, Kd, and Tf, and a low peak control effort.

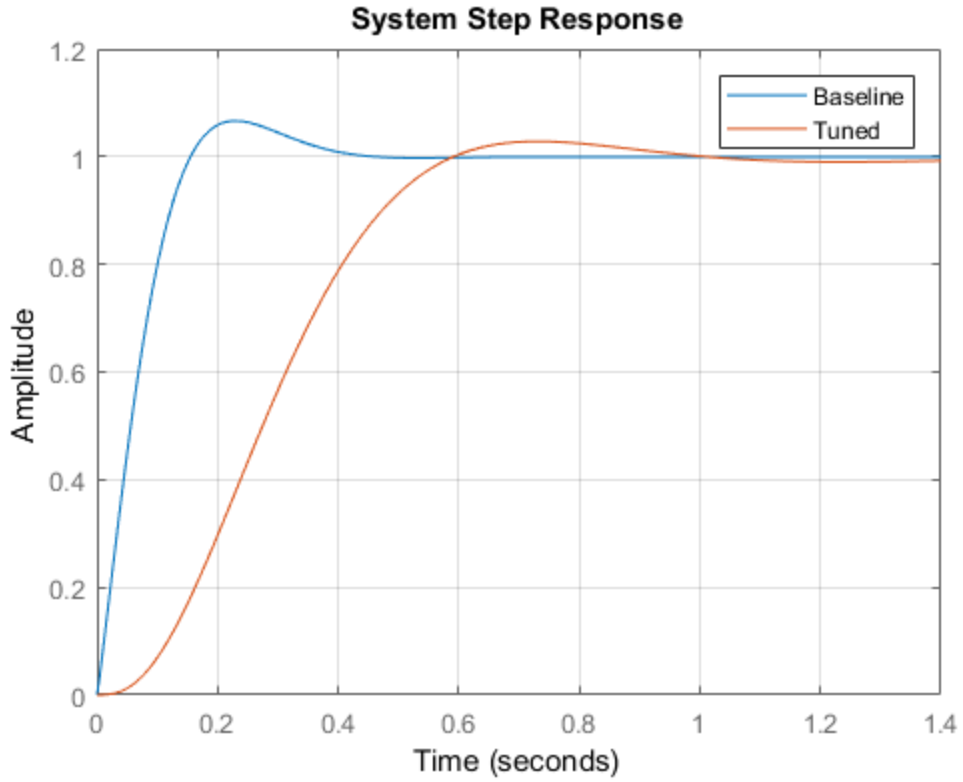
Part 5 PIDF Controller

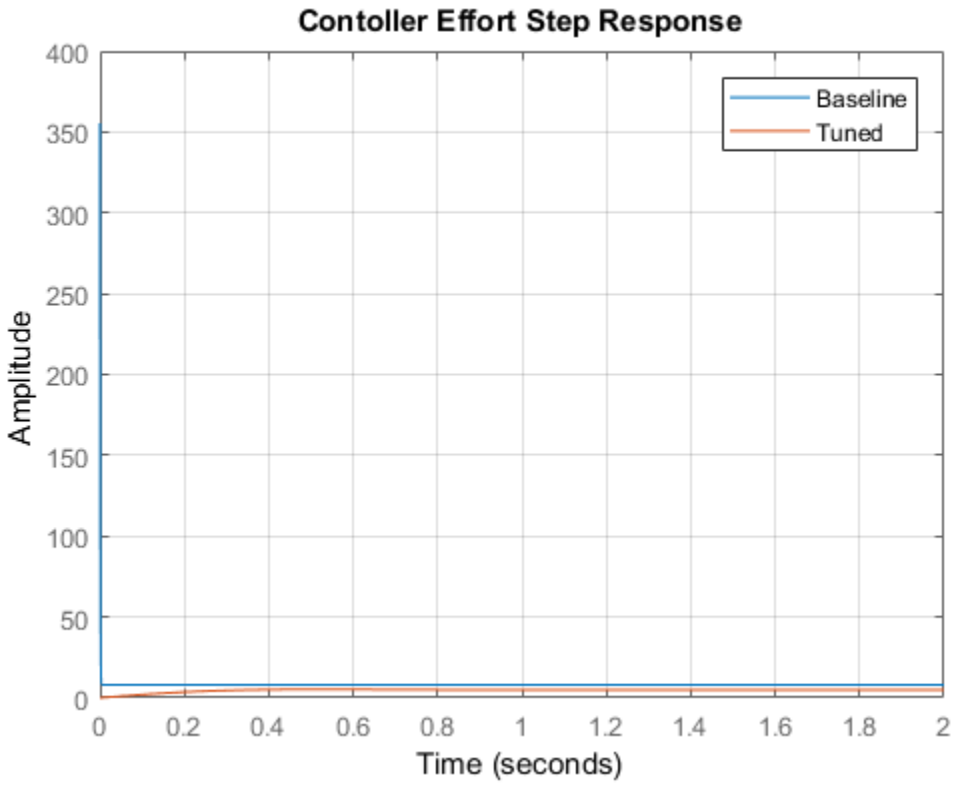
The controller effort transfer function is proper due to the introduced Tf term. Now the controller can be tuned with both constraints in mind. This controller is significantly harder to tune over all the others and to keep all for Kpid and Tf present in the controller. The value for response time found was 0.563, and the value for the transient response (though is didnt matter is it was any value below this) was 0.52. This resulted in a %OS of 2.83% and a peak controller effort of 5.27.

```
Cpidf = pidtune(G, 'PIDF');
Tune = pidTuner(G, Cpidf);
waitfor(Tune);
disp('Parameters from PID Tune');
disp('Kp = ');
disp(C.Kp);
disp('Ki = ');
disp(C.Kp);
disp('Kd = ');
disp(C.Kd);
disp('Tf = ');
disp(C.Tf);

% Baseline controller effort and system transfer functions
disp('Baseline Controller Effort Transfer Function');
Upidf = Cpidf/(1+(Cpidf*G))
disp('Baseline System Transfer Function');
Tpidf = (Cpidf*G)/(1+(G*Cpidf))

% Tuned controller effort and system transfer functions
disp('Tuned Controller Effort Transfer Function');
Utpidf = C/(1+(C*G))
disp('Tuned System Transfer Function');
Ttpidf = (C*G)/(1+(G*C))


figure(7); clf;
step(Tpidf);
hold on;
step(Ttpidf);
grid on;
legend('Baseline', 'Tuned');
title('System Step Response');

figure(8); clf;
step(Upidf);
hold on;
step(Utpidf);
grid on;
legend('Baseline', 'Tuned');
title('Contoller Effort Step Response');
```

```
Parameters from PID Tune
Kp =
    0.7928

Ki =
    0.7928

Kd =
   -0.1876

Tf =
    0.2367

Baseline Controller Effort Transfer Function

Upidf =


  355.4 s^6 + 6.011e05 s^5 + 4.389e07 s^4 + 1.153e09 s^3 + 1.279e10
 s^2

                                                                    +
 5.043e10 s

 ---------------------------------------------------------------------


  s^6 + 3262 s^5 + 2.724e06 s^4 + 1.026e08 s^3 + 1.567e09 s^2 +
 1.009e10 s


Continuous-time transfer function.

Baseline System Transfer Function

Tpidf =


  1.422e04 s^6 + 2.405e07 s^5 + 1.756e09 s^4 + 4.614e10 s^3 + 5.114e11
 s^2

                                                                    +
 2.017e12 s

 ---------------------------------------------------------------------
```

```
  s^8 + 3292 s^7 + 2.822e06 s^6 + 1.85e08 s^5 + 5.19e09 s^4 + 7.761e10
 s^3

                                                    + 6.159e11 s^2 +
 2.017e12 s

Continuous-time transfer function.

Tuned Controller Effort Transfer Function

Utpidf =

     21.04 s^5 + 794.7 s^4 + 9431 s^3 + 4.219e04 s^2 + 6.315e04 s
  ------------------------------------------------------------------
  s^6 + 38.45 s^5 + 471.4 s^4 + 3067 s^3 + 1.011e04 s^2 + 1.263e04 s

Continuous-time transfer function.

Tuned System Transfer Function

Ttpidf =

    841.4 s^5 + 3.179e04 s^4 + 3.772e05 s^3 + 1.688e06 s^2 + 2.526e06
 s

 ------------------------------------------------------------------------

  s^8 + 68.45 s^7 + 1825 s^6 + 2.49e04 s^5 + 1.964e05 s^4 + 9.295e05
 s^3

                                                    + 2.402e06 s^2 +
 2.526e06 s

Continuous-time transfer function.
```


System Step Response
Baseline
Tuned
Amplitude
Time (seconds)
0
0.2
0.4
0.6
0.8
1
1.2
1.4


Contoller Effort Step Response
Baseline
Tuned
Amplitude
Time (seconds)
0
50
100
150
200
250
300
350
400
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

Which one is Best

The best controller for this plant based on the contraints would be the PIDF controller. It has the lowest %OS of the implementable controllers and the peak control effort is also very low. A strong case can be made for the P controller as well as it has similar %OS and peak effort but much easier to implement.

Published with MATLAB® R2017a