Daftar Isi

Daftar Isi 1
Modul 3 Kontroler PID 2
BAB 1 Pendahuluan Kontroler PID 2
1.1 Definisi 2
1.2 Karakteristik Performa 3
1.3 Contoh Permasalahan 3
1.4 Proportional (P) Control 5
1.5 Proportional-Derivative (PD) Control 8
1.6 Proportional-Integral (PI) Control 11
1.7 Proportional-Integral-Derivative Control 14
BAB 2 Desain Kontroler PID Metode Ziegler Nichols 15
2.1 Metode Ziegler Nichols Pertama 16
2.1.1 Contoh Soal (1) 16
2.2 Metode Ziegler Nichols Kedua 19
2.2.1 Contoh Soal (2) 20
2.2.2 Contoh Soal (3) 28
BAB 3 Desain Kontroler PID Metode Analitik 30
3.1 Contoh Soal (4) 31

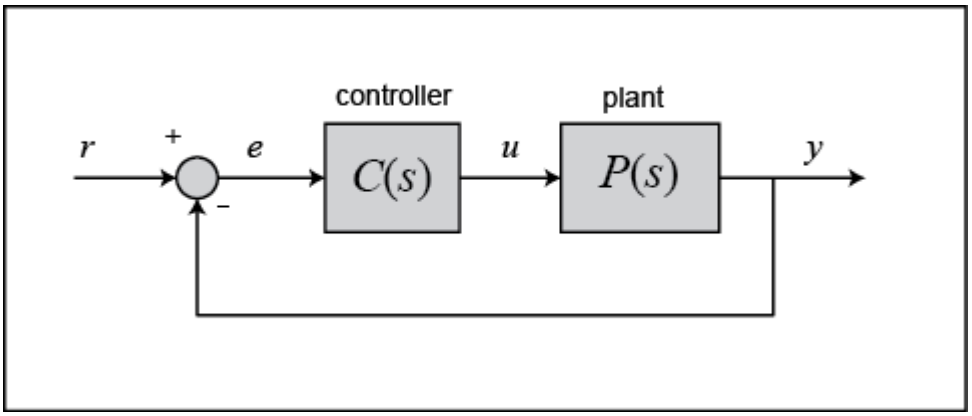
Modul 3 Kontroler PID

Pada modul ini akan dibahas mengenai kontroler PID atau *Proportional-Integral-Derivative*. Dimulai dari pembahasan terkait kontroler ini termasuk jenis-jenisnya hingga desain kontroler PID dalam mengatasi permasalahan pada suatu sistem. Desain dibagi menjadi 2 yaitu berdasarkan metode Ziegler Nichols maupun analitik.

BAB 1 Pendahuluan Kontroler PID

1.1 Definisi

Berikut adalah sistem *closed-loop unity-feedback* dengan *plant* sistem adalah $P(s)$ dan kontroler $C(s)$.



Sinyal $\boldsymbol{u}$ merupakan output dari kontroler PID yang berperan sebagai input kontrol bagi *plant*. Persamaan input kontrol adalah sebagai berikut:

$$u(t) = K_p e(t) + K_i \int_{t_0}^{t} e(t) + K_d \frac{de(t)}{dt}$$

Variabel $\boldsymbol{e}$ adalah sinyal error pada sist3em yang merupakan selisih antara input/referensi $\boldsymbol{r}$ dan output $\boldsymbol{y}$. Sinyal error $\boldsymbol{e}$ diinputkan ke kontroler PID. Di dalam kontroler PID, sinyal kontrol $\boldsymbol{u}$ didapat dengan menggunakan persamaan di atas dimana terdapat 3 gain yaitu K_p atau gain proporsional, K_i atau gain integral, dan K_d atau gain derivatif.

Sinyal kontrol $\boldsymbol{u}$ yang dihasilkan akan masuk ke *plant* dan didapatkan output $\boldsymbol{y}$. Output $\boldsymbol{y}$ di-umpan balik dan dibandingkan lagi dengan sinyal referensi $\boldsymbol{r}$ untuk mendapatkan sinyal error $\boldsymbol{e}$ yang baru. Jadi pada intinya, kontroler PID bekerja berdasarkan sinyal error yang dihasilkan pada *plant* atau sistem.

Fungsi alih atau *transfer function* dari kontroler PID dalam domain laplace yaitu

$$\frac{U(s)}{E(s)} = K_p + K_i \frac{1}{s} + K_d s$$

Kita dapat meng*enerate* fungsi alih tersebut pada *script* MATLAB dengan menggunakan *syntax* `pid` sebagai berikut

```
Kp = 1; %gain Kp
```

```
Ki = 1; %gain Ki
Kd = 1; %gain Kd


C = pid(Kp,Ki,Kd) %kontroler PID
```

```
C =

             1
  Kp + Ki * --- + Kd * s
             s

  with Kp = 1, Ki = 1, Kd = 1

Continuous-time PID controller in parallel form.
Model Properties
```

1.2 Karakteristik Performa

Karakteristik performa dari masing-masing jenis *gain* pada PID dapat dijelaskan melalui tabel berikut

Gain\Respon	*Rise Time*	*Overshoot*	*Settling Time*	Error SS
Kp	Menurun	Meningkat	Perubahan Sedikit	Menurun
Ki	Menurun	Meningkat	Meningkat	Mengeliminasi
Kd	Perubahan Sedikit	Menurun	Menurun	Tanpa Perubahan

Berdasarkan tabel tersebut, kita dapat menentukan *gain* mana yang perlu digunakan dalam desain kontroler PID. Karakteristik dari sistem atau *plant* dapat dianalisis menggunakan sinyal uji *step* seperti pada bab sebelumnya. Sehingga nantinya kita dapat menentukan karakteristik mana yang perlu diperbaiki apakah karakteristik transien ataupun *steady-state* (SS).

1.3 Contoh Permasalahan

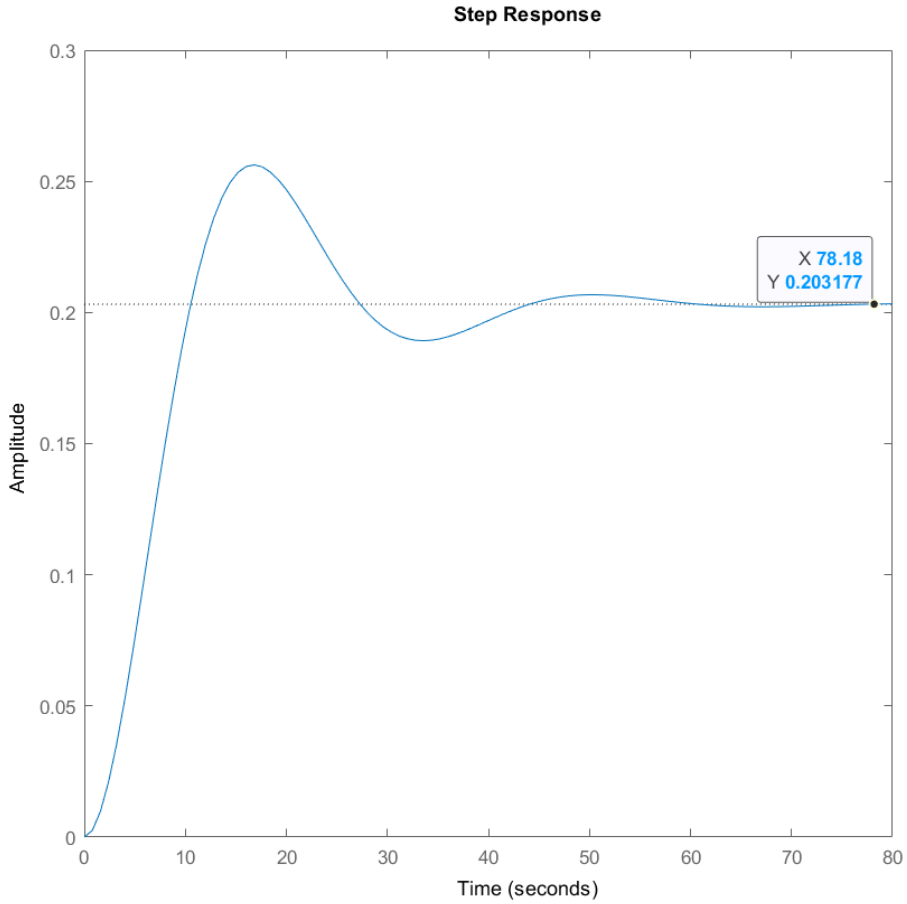
Pada bagian ini akan diberikan sebuah contoh permasalahan dari suatu *plant*. Sehingga nantinya kita dapat menentukan jenis kontroler PID mana yang perlu digunakan berdasarkan karakteristik yang dihasilkan. *Transfer function plant* yang digunakan yaitu

$$P(s) = \frac{0.008434}{s^2 + 0.16s + 0.04152}$$

Respon *open loop* dari sistem dengan menggunakan sinyal uji *step* dapat ditentukan menggunakan *script* berikut

```
s = tf('s');
P = 0.008434/(s^2 + 0.16*s + 0.04152);
```

```
step(P,80) %respon step
```




```
stepinfo(P) %informasi respon step
```

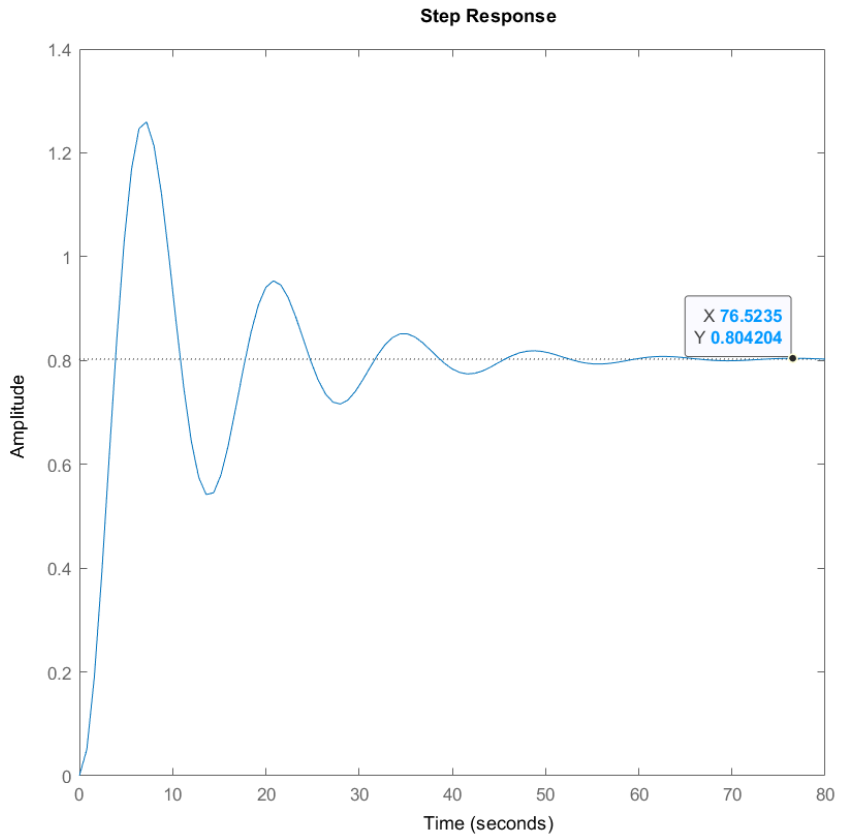
```
ans = struct with fields:
         RiseTime: 7.1346
    TransientTime: 41.2731
     SettlingTime: 41.2731
      SettlingMin: 0.1890
      SettlingMax: 0.2563
        Overshoot: 26.1530
       Undershoot: 0
             Peak: 0.2563
         PeakTime: 16.6937
```

Nilai steady state dari step response diatas adalah sekitar 0.203 menunjukkan error *steady-state* atau ε_{ss} sekitar 80% (didapat dari $\frac{y(\infty)}{r(\infty)} = \frac{1-0.203}{1} = 79.70\%$). Selain itu settling time (±2%) disekitar 41 detik dan rise time (10%-90%) disekitar 7 detik. Selanjutnya akan dipelajari pengaruh masing-masing parameter kombinasi atau jenis kontroler P, PD, PI, PID pada sistem di atas.

1.4 Proportional (P) Control

Berdasarkan tabel sebelumnya, parameter kontroler proporsional K_p akan mengurangi *rise time* dan error *steady-state* serta meningkatkan *overshoot*. *Transfer function loop* tertutup untuk sistem *unity-feedback* dengan kontroler proporsional adalah sebagai berikut:

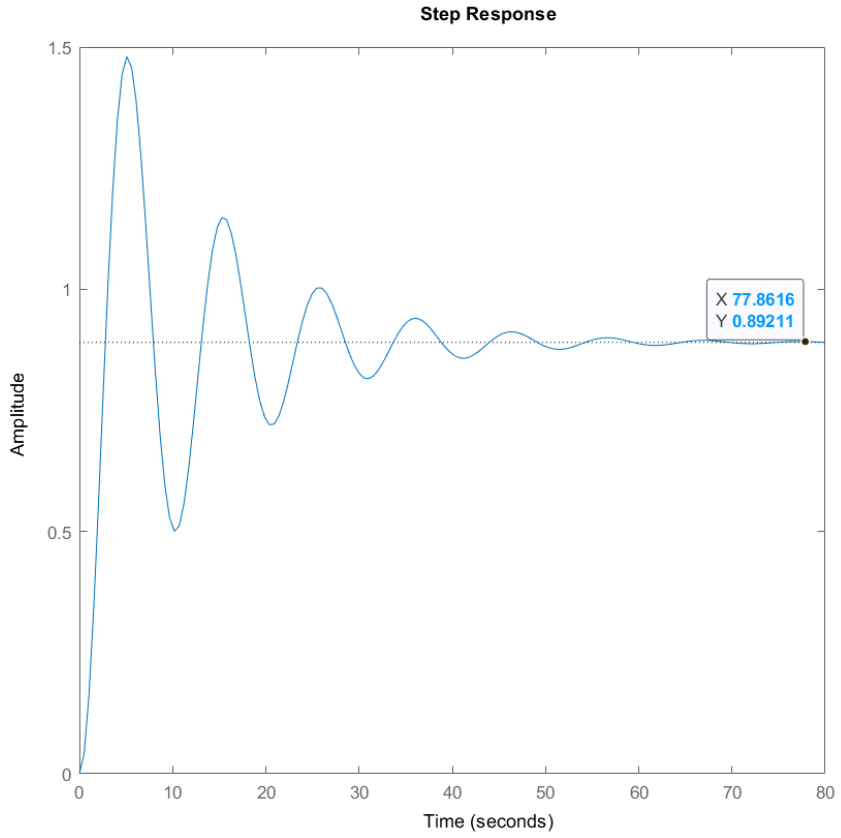
```
Kp = 20; %proportional gain
C = pid(Kp); %kontroller PID
T = feedback(C*P,1); %transfer function loop tertutup

step(T,80) %respon step
stepinfo(T) %informasi respon
```




```
ans = struct with fields:
          RiseTime: 2.6003
     TransientTime: 49.0295
      SettlingTime: 49.0295
       SettlingMin: 0.5392
       SettlingMax: 1.2622
         Overshoot: 57.2926
        Undershoot: 0
```

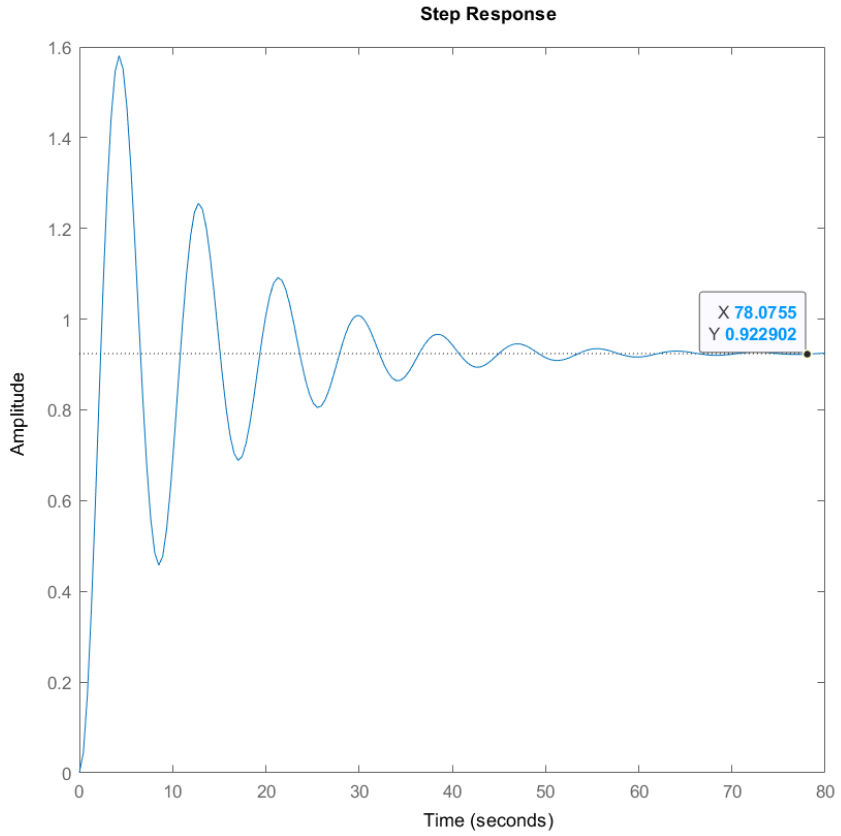
```
            Peak: 1.2622
        PeakTime: 6.9078
```

```
Kp = 40;
C = pid(Kp);
T = feedback(C*P,1);

step(T,80)
stepinfo(T)
```




```
ans = struct with fields:
          RiseTime: 1.8775
    TransientTime: 47.3440
     SettlingTime: 47.3440
      SettlingMin: 0.5002
      SettlingMax: 1.4801
        Overshoot: 66.2216
       Undershoot: 0
             Peak: 1.4801
         PeakTime: 5.1039
```

```
Kp = 60;
C = pid(Kp);
T = feedback(C*P,1);

step(T,80)
stepinfo(T)
```



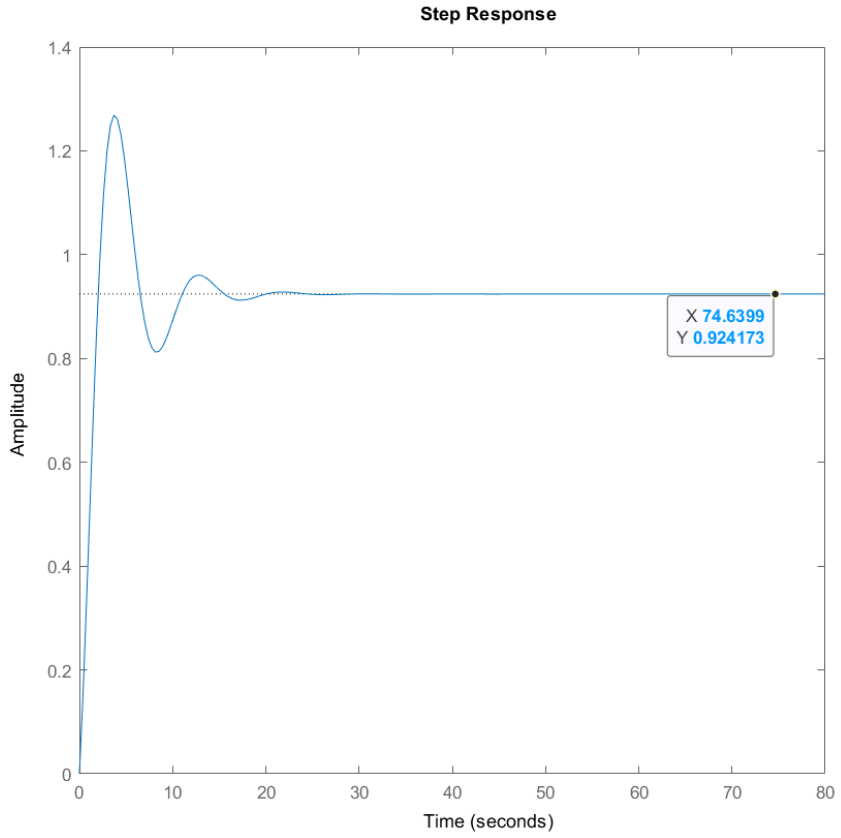

```
ans = struct with fields:
         RiseTime: 1.5333
    TransientTime: 47.6989
     SettlingTime: 47.6989
      SettlingMin: 0.4578
      SettlingMax: 1.5808
        Overshoot: 71.0477
       Undershoot: 0
             Peak: 1.5808
         PeakTime: 4.2456
```

Dapat dilihat dari respon di atas bahwa kenaikan nilai *gain* K_p mengurangi *rise time* dan error *steady-state*, serta meningkatkan *overshoot*.

1.5 Proportional-Derivative (PD) Control

Berdasarkan tabel sebelumnya, penambahan parameter kontroler derivatif K_d akan mengurangi *overshoot* dan *settling time*. *Transfer function loop* tertutup untuk sistem *unity-feedback* dengan kontroler proporsional-derivatif yaitu

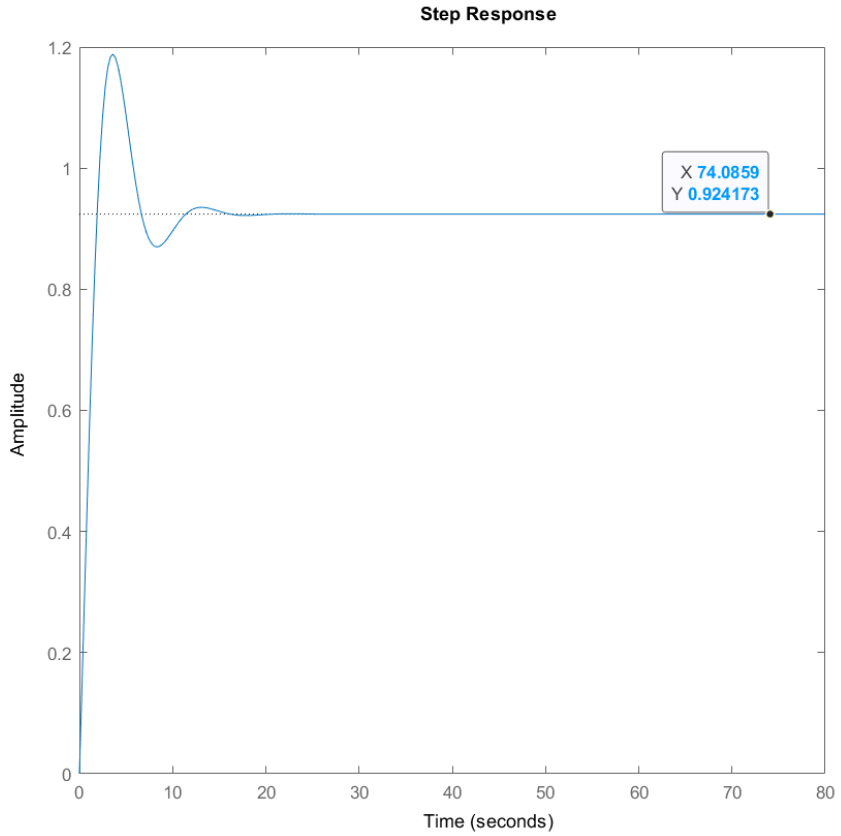
```
Kp = 60;
Kd = 40; %derivative gain
C = pid(Kp,0,Kd);
T = feedback(C*P,1);

step(T,80)
stepinfo(T)
```




```
ans = struct with fields:
         RiseTime: 1.5519
    TransientTime: 14.4072
     SettlingTime: 14.4072
      SettlingMin: 0.8118
      SettlingMax: 1.2685
        Overshoot: 37.2549
       Undershoot: 0
             Peak: 1.2685
```

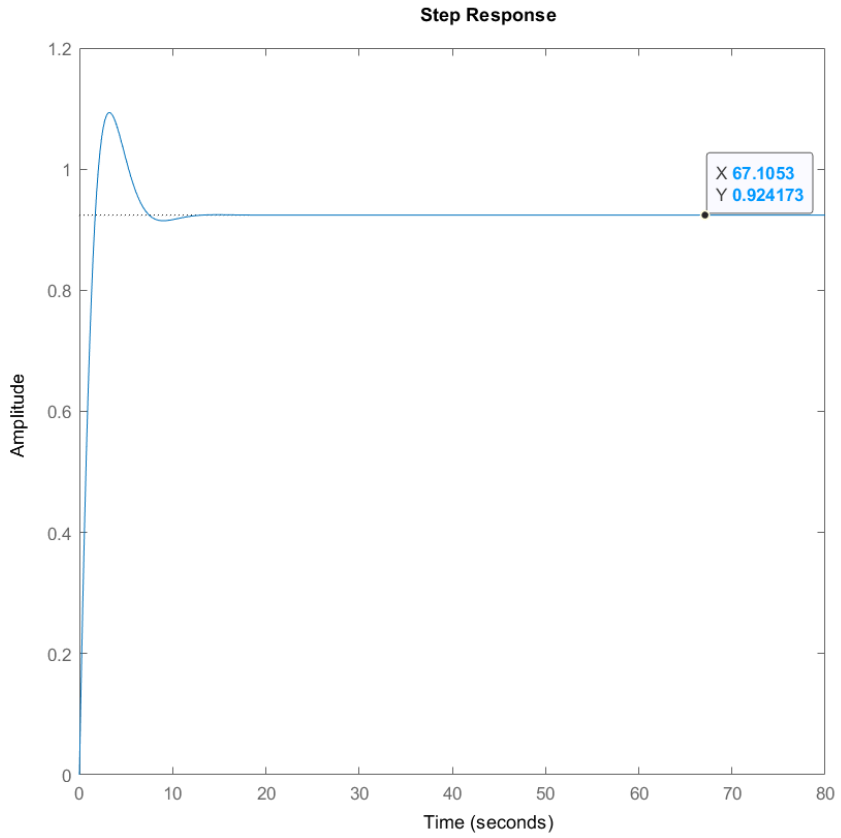
```
        PeakTime: 3.7037
```

```
Kp = 60;
Kd = 60;
C = pid(Kp,0,Kd);
T = feedback(C*P,1);

step(T,80)
stepinfo(T)
```




```
ans = struct with fields:
         RiseTime: 1.4740
    TransientTime: 10.4169
     SettlingTime: 10.4169
      SettlingMin: 0.8349
      SettlingMax: 1.1882
        Overshoot: 28.5653
       Undershoot: 0
             Peak: 1.1882
         PeakTime: 3.5954
```

```
Kp = 60;
Kd = 100;
C = pid(Kp,0,Kd);
T = feedback(C*P,1);

step(T,80)
stepinfo(T)
```



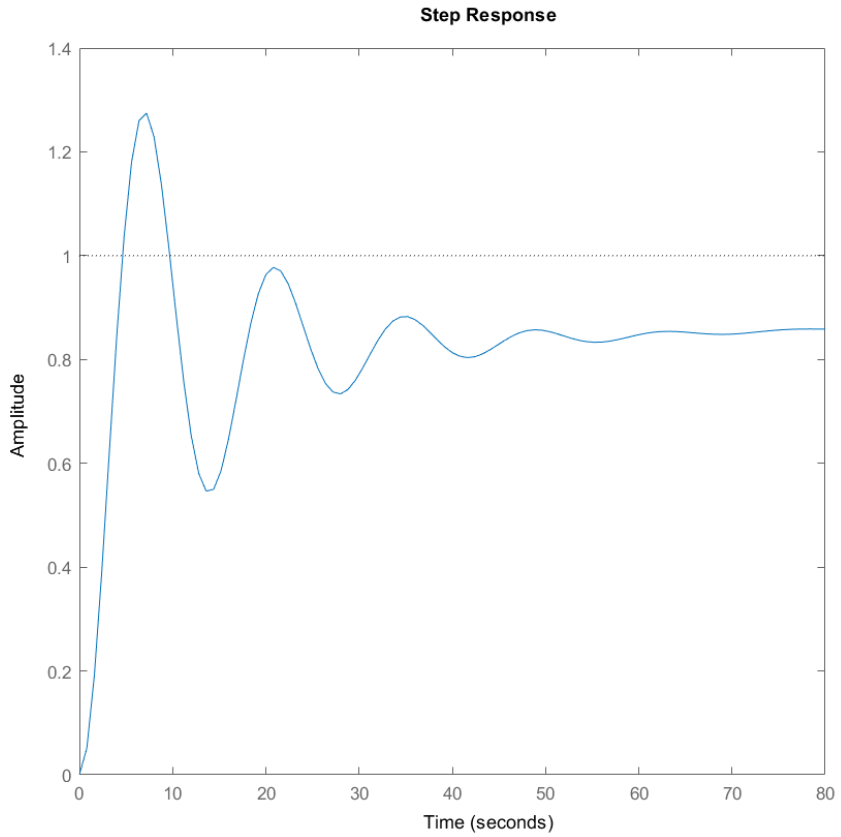

```
ans = struct with fields:
         RiseTime: 1.2891
    TransientTime: 6.6291
     SettlingTime: 6.6291
      SettlingMin: 0.8551
      SettlingMax: 1.0937
        Overshoot: 18.3451
       Undershoot: 0
             Peak: 1.0937
         PeakTime: 3.2127
```

Terlihat dari respon di atas, penambahan nilai parameter K_d pada kontroler PD dengan K_p konstan akan mereduksi *overshoot* dan *settling time*, namun tidak terlalu mempengaruhi *rise time* dan error *steady state*.

1.6 Proportional-Integral (PI) Control

Berdasarkan tabel sebelumnya, penambahan parameter kontroler integral K_i cenderung akan mengurangi *rise time*, meningkatkan *overshoot* dan *settling time*, serta mengeliminasi error *steady state*. Karena efek parameter kontroler integral hampir mirip seperti kontroler proporsional, sehingga perlu mereduksi nilai K_p yang awalnya 60 menjadi lebih kecil.

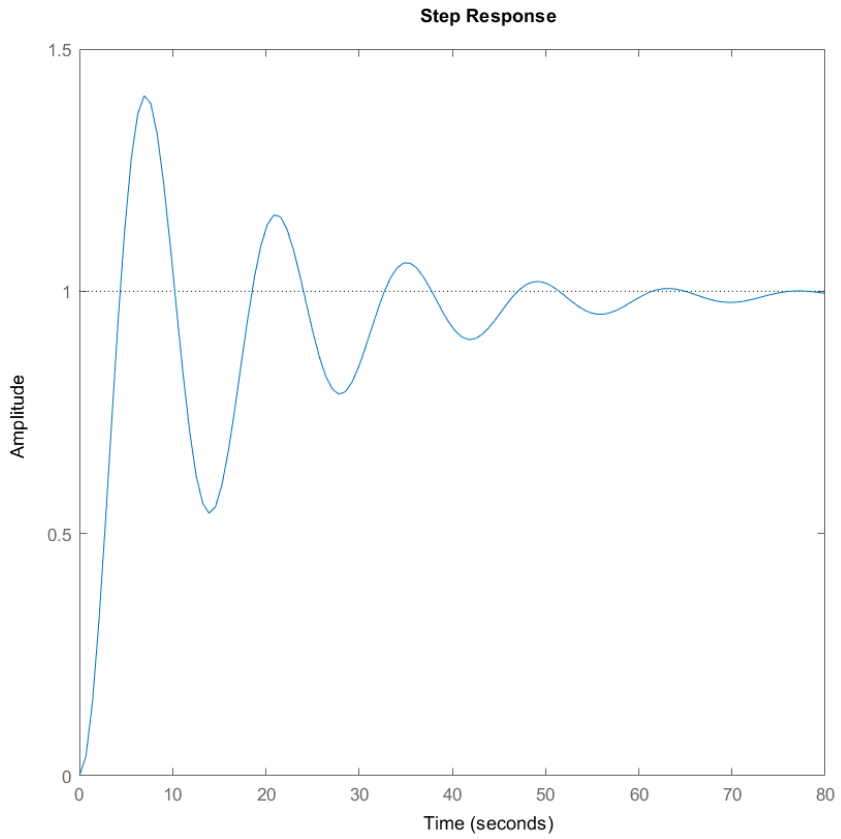
```
Kp = 20;
Ki = 0.1; %integral gain
C = pid(Kp,Ki,0);
T = feedback(C*P,1);

step(T,80)
stepinfo(T)
```




```
ans = struct with fields:
         RiseTime: 3.1614
    TransientTime: 566.7495
     SettlingTime: 566.7495
      SettlingMin: 0.5453
```

```
    SettlingMax: 1.2764
      Overshoot: 27.6386
     Undershoot: 0
           Peak: 1.2764
       PeakTime: 7.0860
```

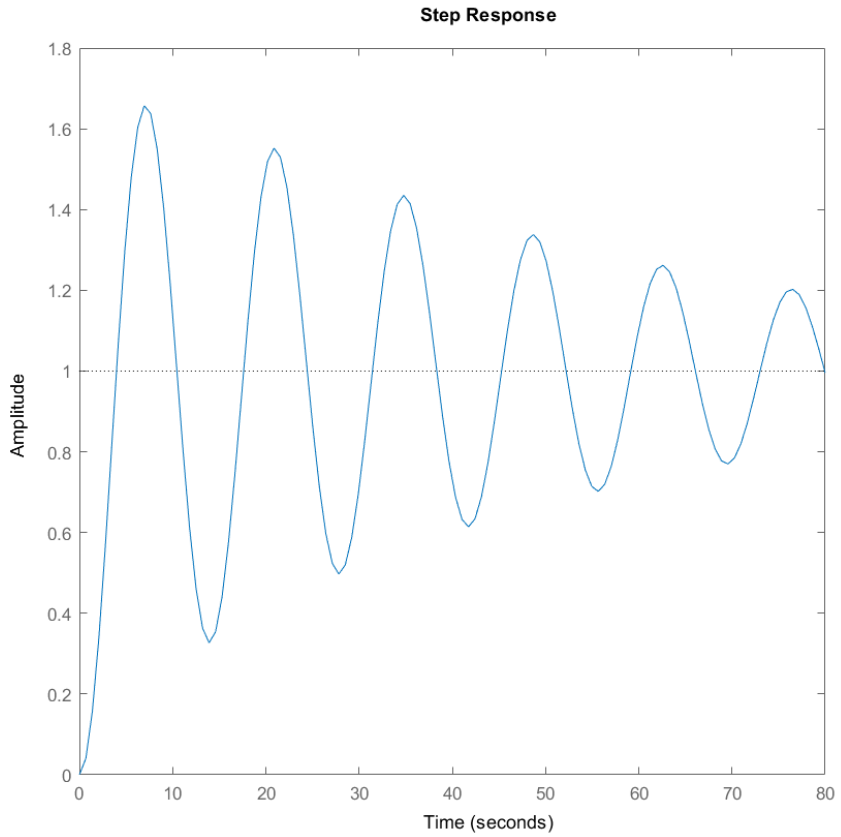
```
Kp = 20;
Ki = 1;
C = pid(Kp,Ki,0);
T = feedback(C*P,1);

step(T,80)
stepinfo(T)
```




```
ans = struct with fields:
         RiseTime: 2.9462
    TransientTime: 71.4084
     SettlingTime: 71.4084
      SettlingMin: 0.5420
      SettlingMax: 1.4034
        Overshoot: 40.3353
       Undershoot: 0
```

```
            Peak: 1.4034
       PeakTime: 6.9333
```

```
Kp = 20;
Ki = 3;
C = pid(Kp,Ki,0);
T = feedback(C*P,1);

step(T,80)
stepinfo(T)
```



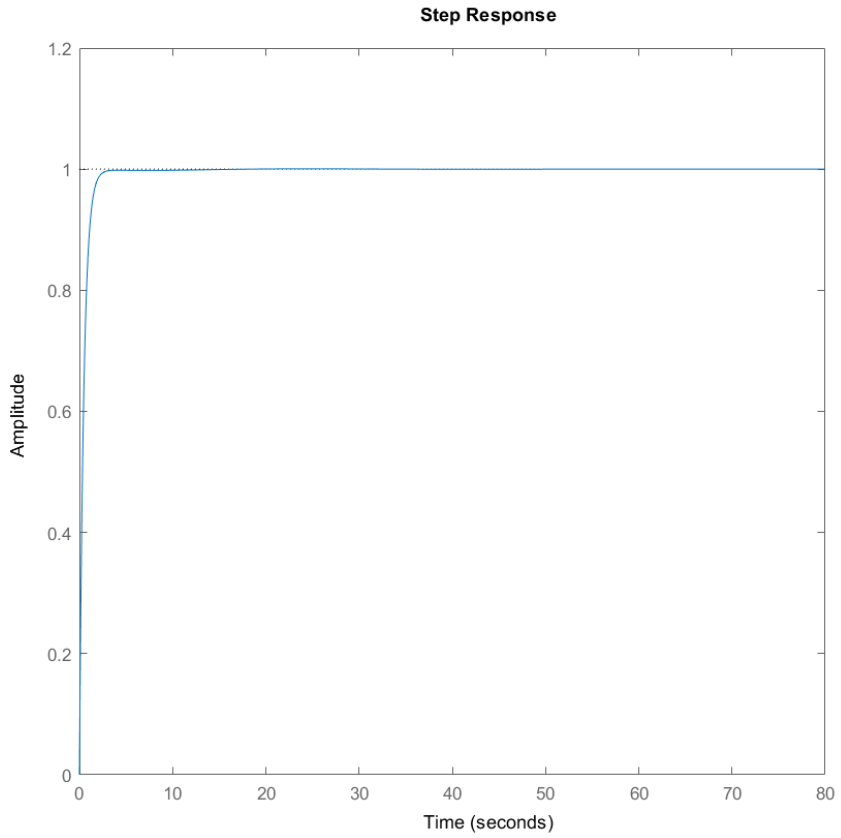

```
ans = struct with fields:
          RiseTime: 2.6554
    TransientTime: 200.9834
     SettlingTime: 200.9834
      SettlingMin: 0.3275
      SettlingMax: 1.6563
        Overshoot: 65.6292
       Undershoot: 0
             Peak: 1.6563
         PeakTime: 6.9276
```

Dapat dilihat dari respon di atas, penambahan kontroler integral akan membuat sistem berosilasi. Hal ini dikarenakan efek penambahan kontroler integral hampir mirip seperti efek penambahan kontroler proporsional sehingga akan mengakibatkan efek *double*. Selain itu dapat dilihat juga kedua kontroler ini dapat mengeliminasi error *steady state*. Pada plot pertama tidak terlihat bahwa error *steady state* telah tereliminasi karena waktu yang diperlukan hingga error tereliminasi semakin besar dengan gain K_i yang semakin kecil. Tetapi nilai *steady-state* dari respon sistem dapat dilihat pada garis putus-putus berwarna abu-abu dimana pada ketiga plot terletak pada nilai 1 (tanpa error).

1.7 Proportional-Integral-Derivative Control

Tujuan dalam penggunaan kontroler PID pada umumnya adalah untuk mendapatkan respon sistem yang bagus yaitu respon sistem yang tanpa *overshoot*, *rise time* cepat atau kecil, dan tanpa error *steady state*. Setelah beberapa kali dilakukan tuning parameter PID didapatkan nilai parameter menghasilkan respon yang diinginkan adalah $K_p = 40, K_i = 10, K_d = 250$ sebagai berikut

```
Kp = 40;
Ki = 10;
Kd = 250;

C = pid(Kp,Ki,Kd);
T = feedback(C*P,1);

step(T,80)
```




```
stepinfo(T)
```

```
ans = struct with fields:
         RiseTime: 1.0431
    TransientTime: 1.8707
     SettlingTime: 1.8707
      SettlingMin: 0.9043
      SettlingMax: 0.9979
        Overshoot: 0
       Undershoot: 0
             Peak: 0.9979
         PeakTime: 3.4740
```

Dapat dilihat berdasarkan respon sistem dan stepinfo, didapatkan output tanpa *overshoot*, *rise time* dan *settling time* lebih cepat, dan tanpa error *steady state*. Nilai parameter *gain* K_p, K_i, K_d didapat dari proses tuning yang akan dibahas pada sub bab berikutnya.

BAB 2 Desain Kontroler PID Metode Ziegler Nichols

Respon sistem yang sesuai dapat diperoleh dengan melakukan tuning parameter K_p, K_i, K_d dari kontroler PID seperti yang sudah disebutkan sebelumnya. Terdapat beberapa metode untuk melakukan tuning parameter yang akan dibahas. Pada bagian ini

akan dibahas mengenai metode tuning Ziegler Nichols. Metode ini terbagi menjadi 2 berdasarkan respon sistem yang digunakan yaitu

2.1 Metode Ziegler Nichols Pertama

Pada metode pertama ini akan digunakan respon dari sistem ***open loop***. Syarat dari penggunaan metode ini yaitu respon *open loop* sistem terhadap sinyal uji *step* harus berbentuk seperti kurva 's' atau stabil. Berikut langkah-langkah tuning Ziegler Nichols metode pertama,

1. Modelkan persamaan sistem dengan metode Ziegler-Nichols dengan menarik *tangent line* pada *inflection point* sehingga didapatkan waktu delay L dan konstanta waktu T.
2. Setelah didapatkan L dan konstanta waktu T, lakukan perhitungan parameter PID berdasarkan tabel di bawah ini. *Gain* K_i didapat dari $K_p \times \frac{1}{T_i}$ dan *gain* K_d didapat dari $K_p \times T_d$.

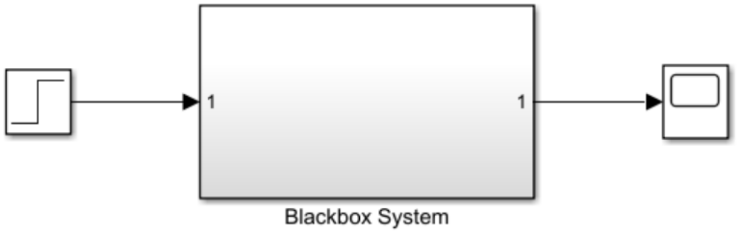
Type of Controller	K_p	T_i	T_d
P	$\frac{T}{L}$	∞	0
PI	$0.9\frac{T}{L}$	$\frac{L}{0.3}$	0
PID	$1.2\frac{T}{L}$	$2L$	$0.5L$

3. Fine tuning apabila hasil kurang memuaskan.

Inflection point merupakan titik dimana kurva memiliki gradien terbesar. Waktu delay L didapat dari selisih antara waktu awal terhadap waktu dimana terjadi perpotongan garis *tangent line* terhadap nilai respon awal (sumbu horizontal). Selanjutnya konstanta waktu T merupakan selisih antara waktu delay L terhadap waktu dimana terjadi perpotongan garis *tangent line* terhadap nilai respon *steady state*.

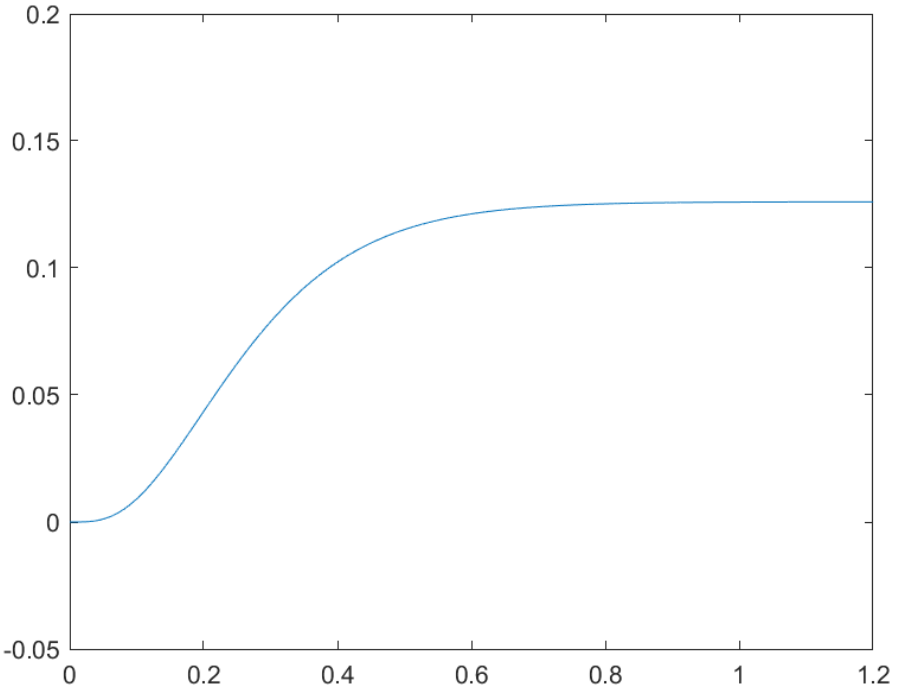
2.1.1 Contoh Soal (1)

Telah di-*generate* sebuah data respon step sistem dengan *time sampling* 0.00001 detik, berikut *blackbox system* yang digunakan untuk *generate* data.



Berikut plot data respon yang dihasilkan

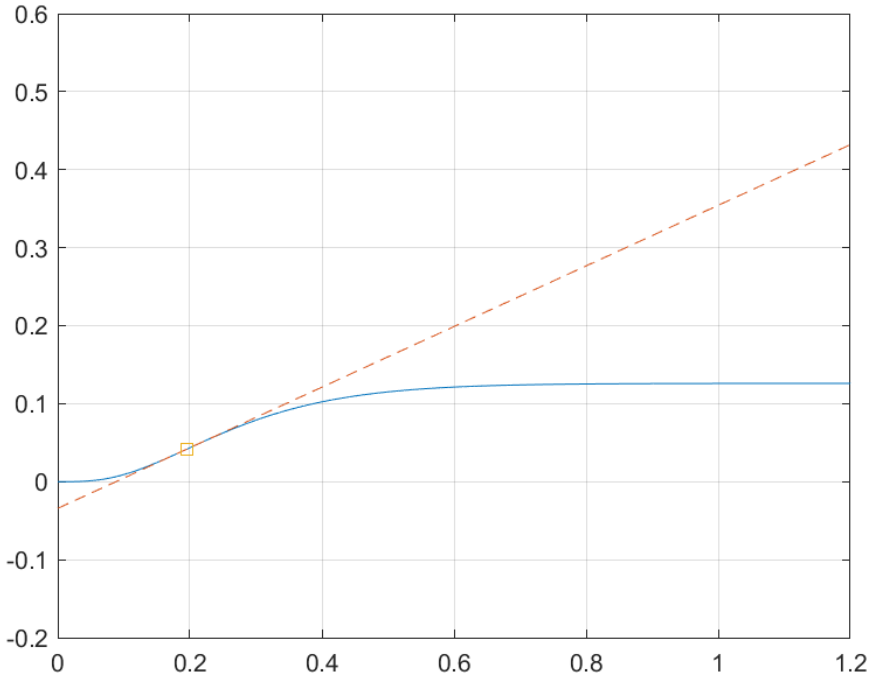
```
load('timeandoutput.mat') %load data
plot(t,Y)
axis([0 1.2 -0.05 0.2]);
```



Berikut adalah langkah-langkah yang ditempuh dalam desain kontroler PID

1. Modelkan persamaan sistem dengan metode Ziegler-Nichols dengan menarik *tangent line* pada *inflection point* sehingga didapatkan waktu delay L dan konstanta waktu T.

```
M = gradient(Y,t); %gradien tiap titik pada plot
[m i] = max(M); %gradien maks m pada data ke i
c = Y(i)-(m*t(i)); %menghitung pergeseran data

plot(t,Y,t,(m*t)+c,'--',t(i),Y(i),'s'); % plot respon
step, garis tangent, serta tanda titik inflection point
grid on;
axis([0 1.2 -0.2 0.6]);
```

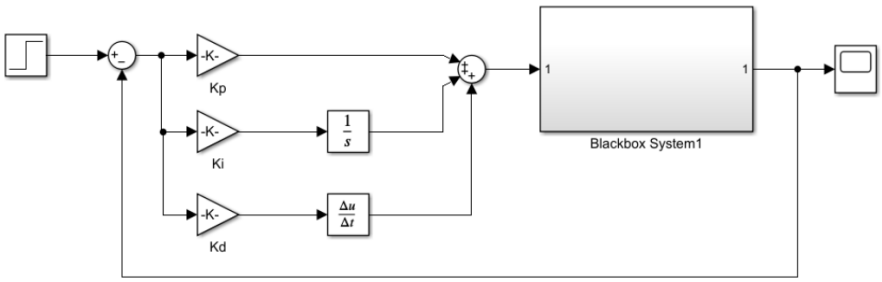



```
L = -c/m %waktu delay
T = ((Y(end)-c)/m)-L %konstanta waktu
```

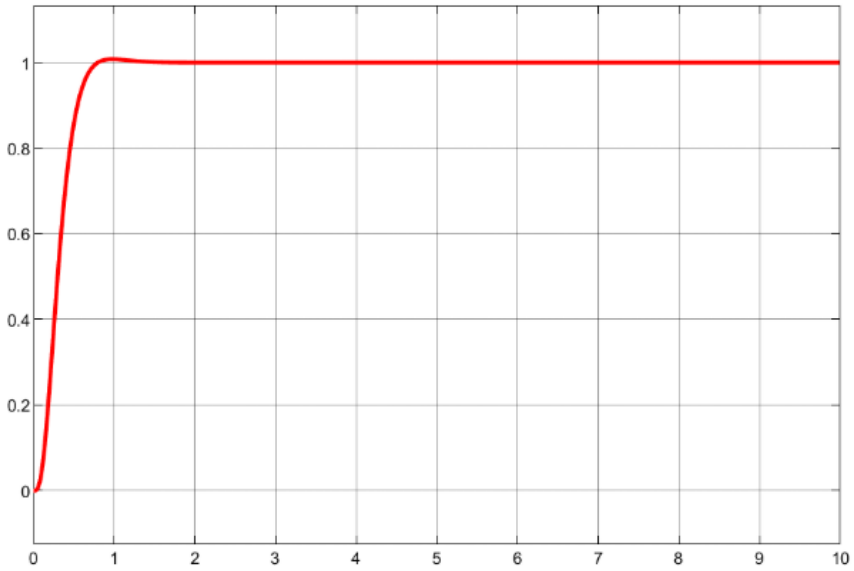
2. Setelah didapatkan L dan konstanta waktu T, lakukan perhitungan parameter PID berdasarkan tabel. Pada percobaan ini akan digunakan jenis kontroler PID.

```
Kp = 1.2*(T/L)
Kp = 4.4202


Ti = 2*L;
Ki = Kp/Ti
Ki = 25.0978


Td = 0.5*L;
Kd = Kp*Td
Kd = 0.1946
```

Berikut diagram blok *closed-loop system* dengan *plant blackbox* yang diberi kontroler PID hasil tuning ZN (Ziegler-Nichols)



Respon step yang dihasilkan sistem dengan kontroler PID yaitu



3. Fine tuning apabila hasil kurang memuaskan. Pada *plant* ini, step respon sudah menghasilkan grafik yang sesuai, sehingga *fine tuning* tidak perlu dilakukan.

Dapat dilihat bahwa data respon yang di-*generate* sebenarnya dalam bentuk **diskrit** bukan **kontinu**. Hal tersebut dikarenakan terdapat *time sampling* yang digunakan. Sehingga sebenarnya kita dapat melakukan *tuning* sistem diskrit melalui sistem kontinunya saja seperti pada contoh.

2.2 Metode Ziegler Nichols Kedua

Pada metode kedua ini akan digunakan respon dari sistem ***closed loop***. Sistem *closed loop* tersebut telah ditambahkan kontroler PID dengan *gain* K_p saja tanpa parameter K_i dan K_d. Metode ini dapat digunakan apabila metode pertama tidak dapat ditempuh seperti karena respon *open loop* yang tidak stabil sehingga tidak membentuk kurva 's'. Berikut langkah-langkah tuning Ziegler Nichols metode kedua,

1. Atur *gain* K_i dan K_d ke 0 (hanya kontroler P saja).

2. Ubah nilai K_p dan cari nilai kritis (disebut K_{cr}) yang menghasilkan output berupa osilasi terus menerus atau mencapai kestabilan marginal yang berarti pole-pole berada tepat di sumbu imajiner.
3. Tentukan periode osilasi output tersebut P_{cr}.
4. Lakukan perhitungan parameter PID berdasarkan tabel di bawah. *Gain* K_i didapat dari $K_p \times \frac{1}{T_i}$ dan *gain* K_d didapat dari $K_p \times T_d$.

Type of Controller	K_p	T_i	T_d
P	$0.5K_{cr}$	∞	0
PI	$0.45K_{cr}$	$\frac{1}{1.2}P_{cr}$	0
PID	$0.6K_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

5. Fine tuning apabila hasil kurang memuaskan.

Metode ini **tidak dapat digunakan** jika respon *closed loop* dari sistem tidak dapat menghasilkan osilasi atau sistem tidak dapat mencapai kestabilan marginal.

2.2.1 Contoh Soal (2)

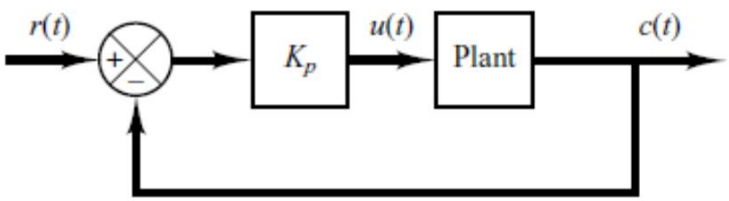
Fungsi alih atau *transfer function* dari *plant* yaitu

$$G_1(s) = \frac{1}{s(s+1)(s+5)}$$

```
s = tf('s');
G1 = 1/(s*(s+1)*(s+5)) %transfer function plant
H1 = 1; %unity feedback
```

Berikut adalah langkah-langkah yang ditempuh dalam desain kontroler PID

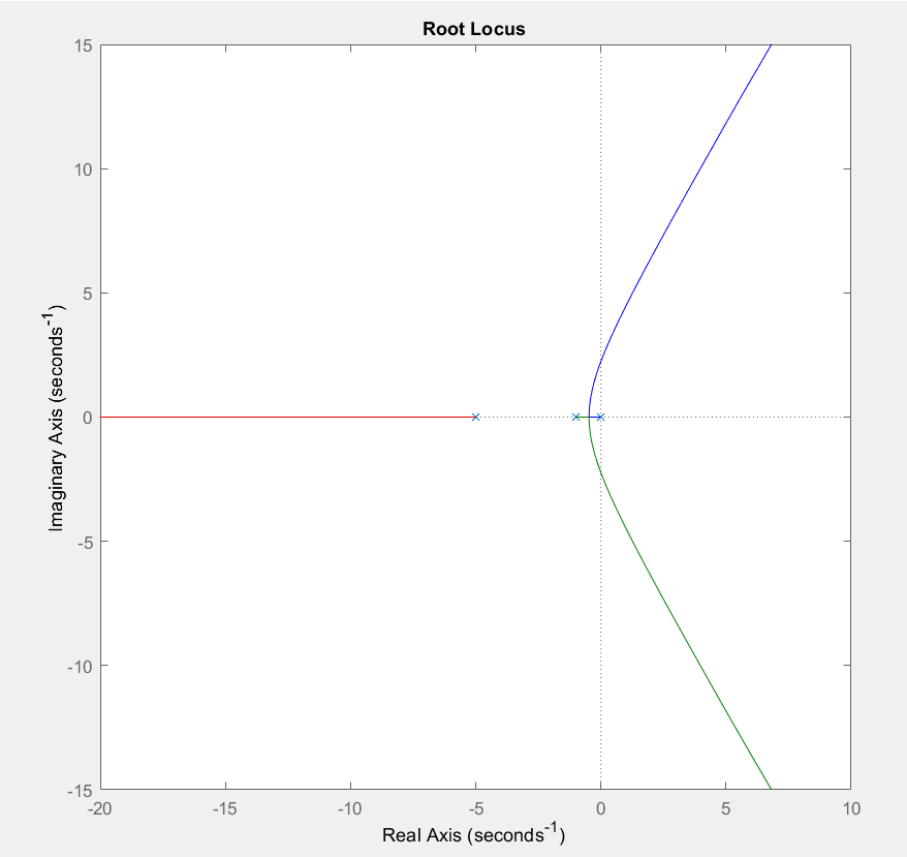
1. Atur *gain* K_i dan K_d ke 0 (hanya kontroler P saja). Diagram sistem *closed loop* menjadi



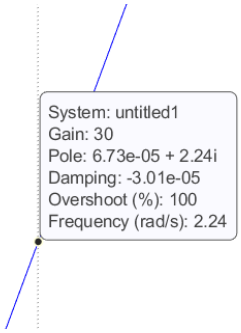
2. Cari nilai kritis K_{cr}. Nilai ini dapat dicari dengan 2 cara yaitu menggunakan plot *root locus* maupun tabel *routh* (analitik)

(Cara 1) *Root Locus*

```
rlocus(G1*H1)
```



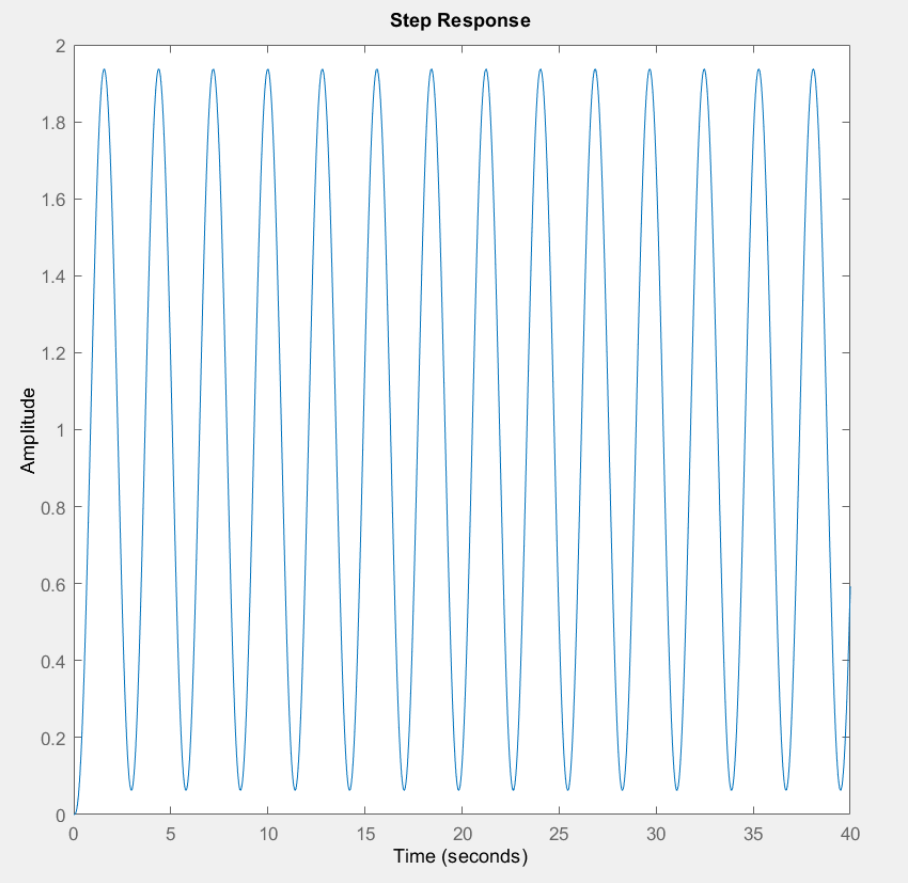
Dengan menggeser cursor dari plot pada sumbu imajiner maka didapat *gain* K_{cr} yaitu



Selain itu juga terdapat nilai frekuensi saat osilasi dalam satuan rad/s. Sehingga nantinya periode osilasi atau P_{cr} dapat ditentukan dengan $P_{cr} = \frac{2\pi}{\omega_{cr}}$ dimana ω_{cr} merupakan frekuensi tersebut. Dilakukan pengecekan respon step apakah sistem *marginally stable* dengan gain 30 sebagai berikut

```
Kcr = 30;
feedback(Kcr*G1,1)
```

```
step(feedback(Kcr*G1,1),40)
```



Terlihat respon step dari sistem berosilasi sehingga bisa dikatakan sistem stabil marginal.

(Cara 2) Tabel *Routh*

Cara ini dapat digunakan apabila kita sudah memiliki *transfer function open loop* dari sistem yaitu $G(s)$. Sedangkan cara sebelumnya dapat digunakan secara empiris. Pertama kita mencari mencari persamaan karakteristik dari sistem *closed loop unity feedback* dimana $H(s) = 1$ yaitu

$$\frac{C(s)}{R(s)} = \frac{KG(s)}{1 + KG(s)H(s)}$$
$$\frac{C(s)}{R(s)} = \frac{KG(s)}{1 + KG(s)}$$

Persamaan karakteristik dari sistem dan bentuk umum persamaannya yaitu

$$1 + KG(s) = 0$$
$$a_0 s^n + a_1 s^{n-1} + \cdots + a_{n-1} s + a_n = 0$$

Tabel *Routh* dapat dengan pola berikut

s^n	a_0	a_2	a_4	a_6	…
s^{n-1}	a_1	a_3	a_5	a_7	…
s^{n-2}	b_1	b_2	b_3	b_4	…
s^{n-3}	c_1	c_2	c_3	c_4	…
…	…	…			
s^2	e_1	e_2			
s^1	f_1				
s^0	g_1				

Nilai dari b_1, b_2, b_3, dst dapat ditentukan dengan

$$b_1 = \frac{a_1 a_2 - a_0 a_3}{a_1}$$
$$b_2 = \frac{a_1 a_4 - a_0 a_5}{a_1}$$
$$b_3 = \frac{a_1 a_6 - a_0 a_7}{a_1}$$
...

Nilai dari c_1, c_2, c_3, dst dapat ditentukan dengan

$$c_1 = \frac{b_1 a_3 - a_1 b_2}{b_1}$$
$$c_2 = \frac{b_1 a_5 - a_1 b_3}{b_1}$$
$$c_3 = \frac{b_1 a_6 - a_1 b_4}{b_1}$$
...

Nilai yang lain dapat ditentukan dengan pola yang sama

Dari penjabaran tersebut maka dapat ditentukan persamaan karakteristik dari sistem pada contoh soal yaitu

$$1 + KG(s) = 0$$
$$1 + K\frac{1}{s(s+1)(s+5)} = 0$$
$$\frac{s(s+1)(s+5) + K}{s(s+1)(s+5)} = 0$$
$$s(s+1)(s+5) + K = 0$$
$$s^3 + 6s^2 + 5s + K = 0$$

sehingga $a_0 = 1, a_1 = 6, a_2 = 5, a_3 = K$. Didapatkan tabel *Routh* sebagai berikut

s^3	1	5
s^2	6	K

s^1	$\frac{30-K}{6}$
s^0	K

Nilai *gain* kritis K_{cr} dapat dicari dengan membuat nilai pada kolom pertama menjadi nol atau disamadengankan 0. Kita akan menggunakan **baris s^1** karena terdapat *gain* K dan **bukan baris s^0** karena kita tidak ingin mendapatkan nilai *gain* kritis 0.

$$\frac{30-K}{6} = 0$$
$$K = 30$$

Nilai K tersebut merupakan nilai K_{cr} yang ingin dicari dimana nilainya sama dengan menggunakan metode *root locus*.

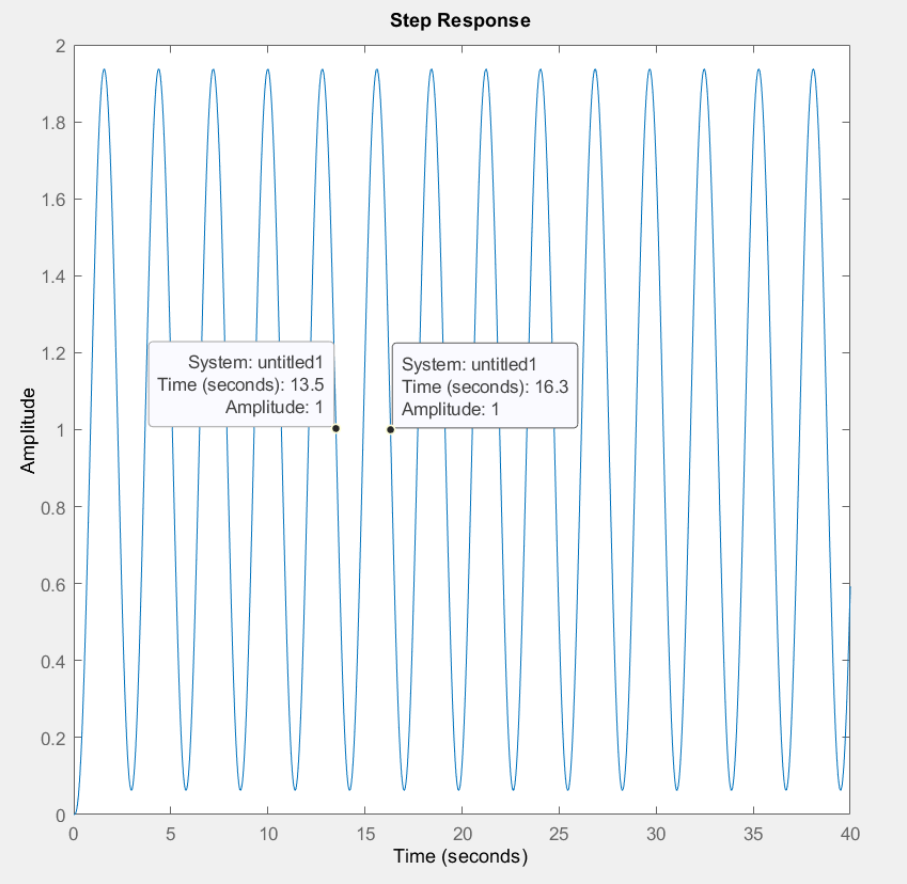
3. Tentukan periode osilasi output P_{cr}.
Dari langkah nomor 2 didapatkan frequensi 2.24 rad/s, sehingga untuk mencari periode harus dikonversi dulu

```
w = 2.24;
f = w/(2*pi)
Pcr = 1/f
```

```
Pcr = 2.8050
```

Kita juga dapat menentukan periode osilasi berdasarkan respon step sistem *closed loop* dengan menggunakan *gain* K_{cr}

```
step(feedback(Kcr*G1,1),40)
```



Periode osilasi merupakan waktu yang diperlukan untuk menempuh satu gelombang penuh osilasi. Pada plot tersebut ditentukan 2 titik dimana titik pertama merupakan titik referensi dan titik kedua merupakan titik setelah gelombang pertama dari titik referensi. Besar periode dapat dicari dari selisih waktu antara kedua titik yaitu

$$P_{cr} = 16.3 - 13.5 = 2.8$$

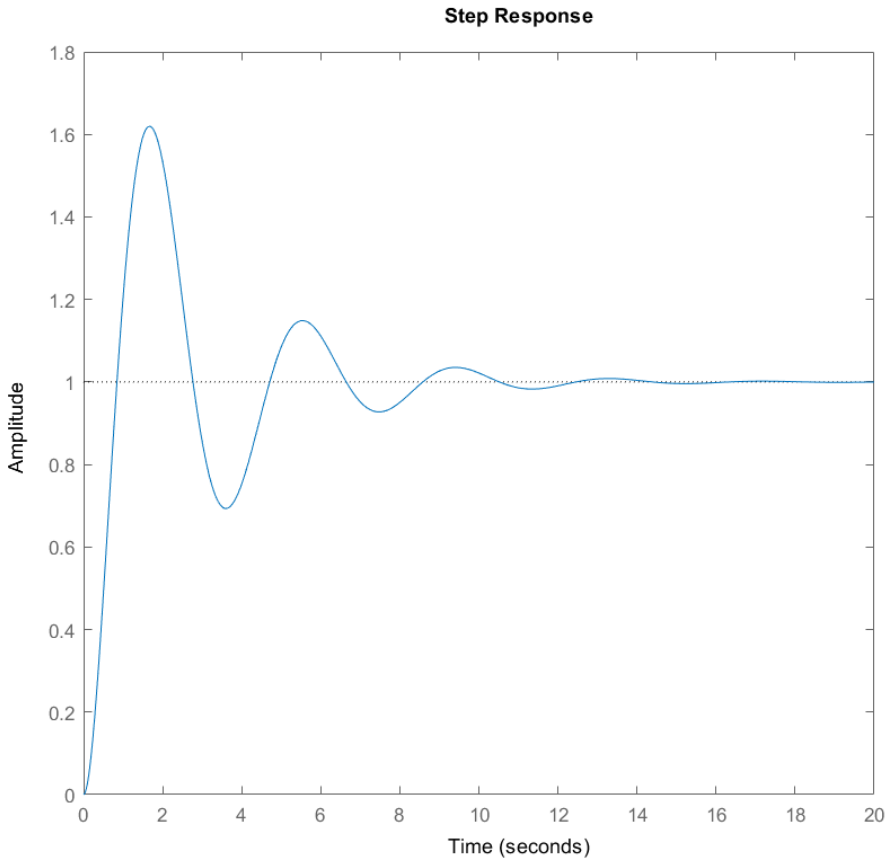
Nilai tersebut kurang lebih sama dengan nilai periode yang dihasilkan menggunakan *root locus*.

4. Lakukan perhitungan parameter PID berdasarkan tabel. Pada *plant* ini akan digunakan jenis kontroler PID langsung sebagai berikut.

```
Kp = 0.6*Kcr
Ti = 0.5*Pcr;
Ki = Kp/Ti
Td = 0.125*Pcr;
Kd = Kp*Td
```

```
Kp = 18
Ki = 12.8343
Kd = 6.3112
```

```
C = pid(Kp,Ki,Kd);


step(feedback(C*G1,1),20) %respon dengan PID
```




```
stepinfo(feedback(C*G1,1)) %informasi respon
```

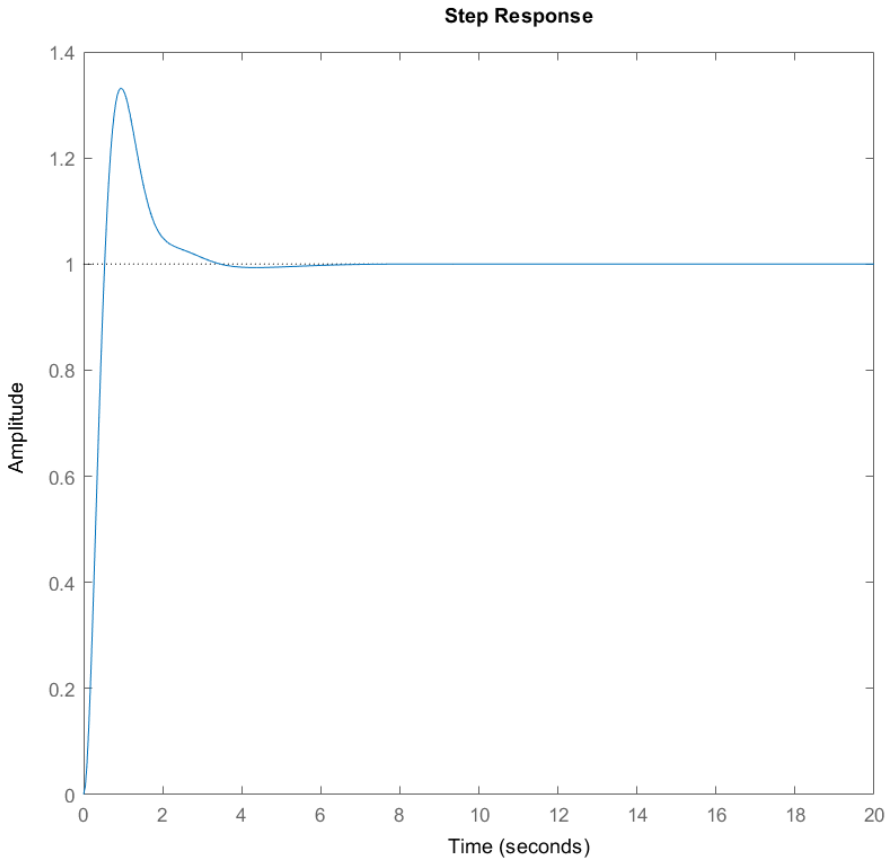
```
ans = struct with fields:
          RiseTime: 0.5780
     TransientTime: 10.0415
      SettlingTime: 10.0415
       SettlingMin: 0.6934
       SettlingMax: 1.6196
         Overshoot: 61.9608
        Undershoot: 0
              Peak: 1.6196
          PeakTime: 1.6701
```

5. Fine tuning apabila hasil kurang memuaskan.
   Respon step dari sistem *closed loop* yang ditambah kontroler PID sebelumnya masih kurang baik dalam hal *overshoot* yaitu sekitar 62% dan *settling time* yaitu sekitar 10 detik. Ziegler-Nichols belum menjamin respon step optimal, Ziegler-Nichols hanya memberi

parameter sebagai acuan awal. Sehingga perlu dilakukan fine tuning untuk mendapatkan respon paling bagus sesuai dengan karakteristik dari masing-masing *gain* PID. Secara singkat, bisa dilakukan cara berikut berdasarkan tabel pada awal bab ini sebelumnya

- Tingkatkan K_p untuk memperbaiki *rise time* dan error *steady-state*
- Tingkatkan K_d untuk mengurangi *overshoot* dan *settling time*
- Tingkatkan K_i untuk memperbaiki *rise time* dan mengeliminasi error *steady-state*

```
Kp = 30;
Ki = 18;
Kd = 18;
C = pid(Kp,Ki,Kd); %kontroler PID fine tuning

step(feedback(C*G1,1),20) %respon update
```




```
stepinfo(feedback(C*G1,1)) %informasi respon
```

```
ans = struct with fields:
```

```
      RiseTime: 0.3653
  TransientTime: 2.7468
   SettlingTime: 2.7468
    SettlingMin: 0.9696
    SettlingMax: 1.3317
      Overshoot: 33.1667
     Undershoot: 0
           Peak: 1.3317
       PeakTime: 0.9445
```

Dapat dilihat bahwa respon step dari *closed-loop plant* atau sistem sudah lebih baik dengan berkurangnya *rise time*, *settling time*, dan terutama *overshoot*.

2.2.2 Contoh Soal (3)

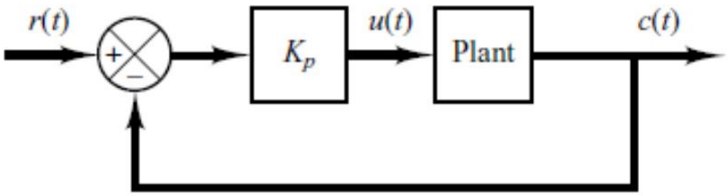
Fungsi alih atau *transfer function* dari *plant* yaitu

$$G_2(s) = \frac{(s+2)(s+3)}{s(s+1)(s+5)}$$

```
s = tf('s');
G2 = ((s+2)*(s+3))/(s*(s+1)*(s+5)) %transfer function plant
H2 = 1; %unity feedback
```

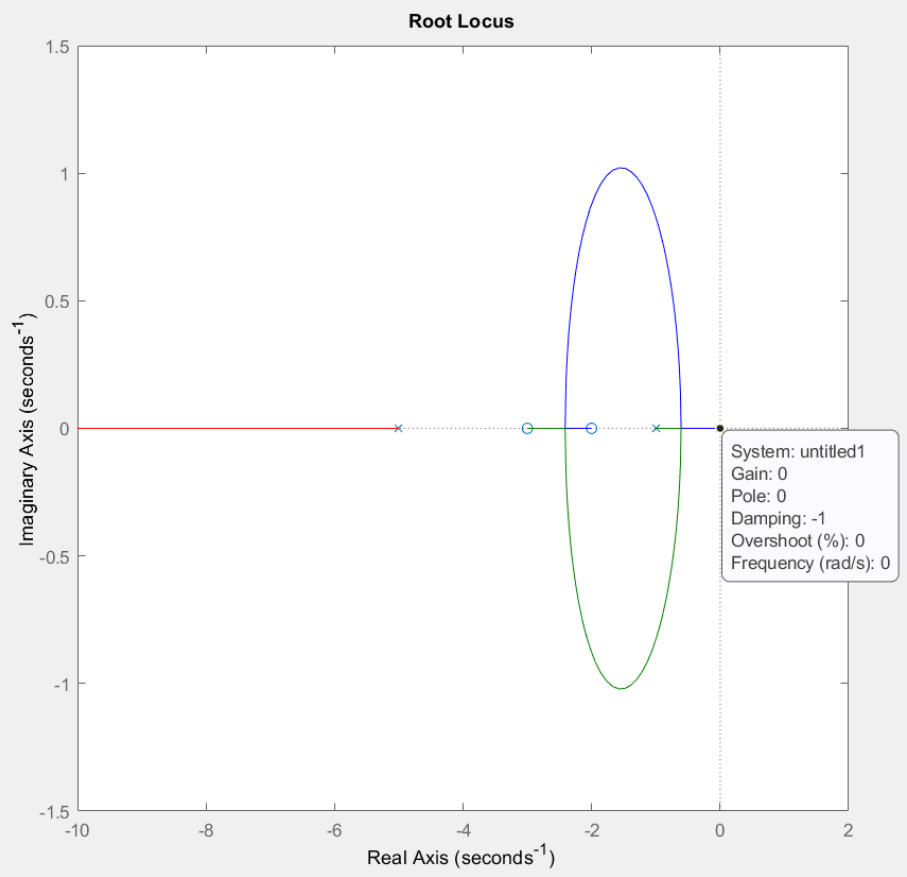
Berikut adalah langkah-langkah yang ditempuh dalam desain kontroler PID

1. Atur *gain* K_i dan K_d ke 0 (hanya kontroler P saja). Diagram sistem *closed loop* menjadi



2. Cari nilai kritis K_{cr}.
   (Cara 1) *Root Locus*

```
rlocus(G2*H2)
```



Tidak ditemukan kestabilan marginal untuk sistem ini karena pole-pole selalu berada di kiri sumbu imajiner

(Cara 2) Tabel *Routh*

Persamaan karakteristik dari sistem pada contoh soal yaitu

$$1 + KG(s) = 0$$
$$1 + K\frac{(s+2)(s+3)}{s(s+1)(s+5)} = 0$$
$$\frac{s(s+1)(s+5) + K(s+2)(s+3)}{s(s+1)(s+5)} = 0$$
$$s(s+1)(s+5) + K(s+2)(s+3) = 0$$
$$s^3 + 6s^2 + 5s + K(s^2 + 5s + 6) = 0$$
$$s^3 + (6+K)s^2 + (5+5K)s + 6K = 0$$

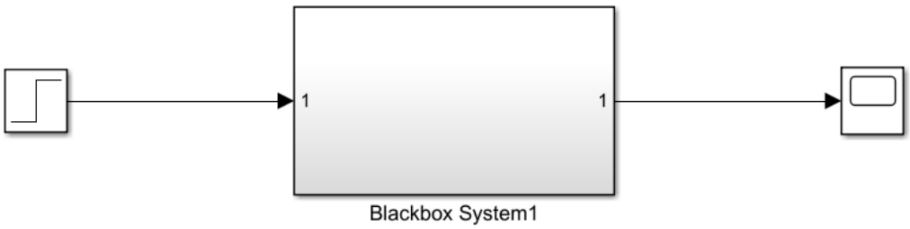
sehingga $a_0 = 1, a_1 = 6 + 5K, a_2 = 5 + 5K, a_3 = 6K$. Didapatkan tabel *Routh* sebagai berikut

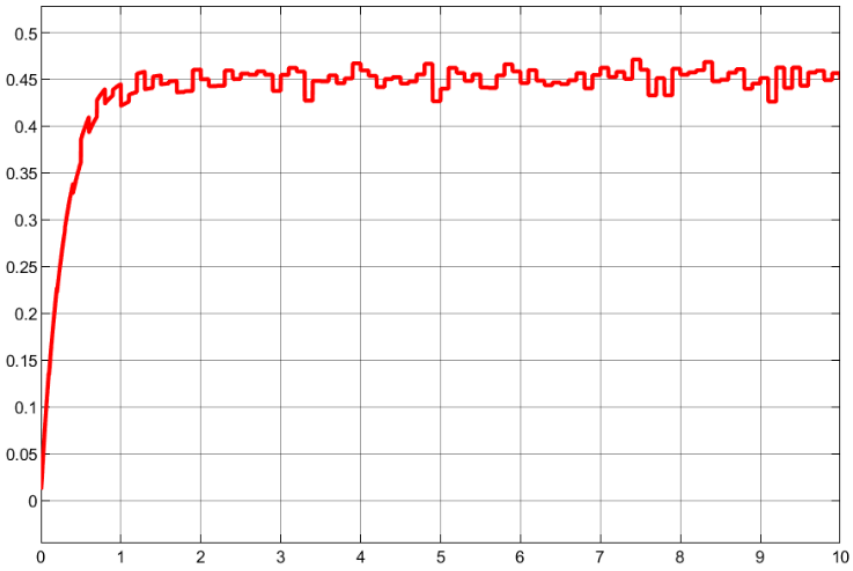
s^3	1	$5 + 5K$
s^2	$6 + K$	$6K$
s^1	$\frac{5K^2 + 29K + 30}{6 + K}$	
s^0	$6K$	

Nilai *gain* kritis K_{cr} dapat dicari dengan membuat nilai pada kolom pertama menjadi nol atau disamadengankan 0. Nilai *gain* yang didapatkan negatif semua. Hal ini berarti bahwa sistem stabil untuk semua nilai K positif dan tidak ditemukan kestabilan marginal sehingga metode Ziegler-Nichols tidak dapat digunakan.

BAB 3 Desain Kontroler PID Metode Analitik

Pada bagian ini akan dibahas mengenai metode tuning kontroler PID secara analitik. Metode ini akan menggunakan model dari sistem yaitu dalam bentuk *transfer function* sistem orde 2. Kontroler PID akan dirancang berdasarkan model sistem yang didapat pada subbab PRBS *modeling*. Berikut respon *open loop* dari sistem *blackbox* yang digunakan





Akan didesain sebuah kontroler PID untuk sistem orde kedua tanpa delay. Berikut adalah prosedur desain kontroler PID dengan cara analitik

1. Menentukan fungsi alih atau *transfer function* sistem orde dua

$$TF = \frac{K\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2}$$

2. Menentukan spesifikasi performansi respon orde pertama yang diinginkan (dikarenakan hasil desain yang diinginkan merupakan sistem orde pertama tanpa *overshoot* dan *zero offset* atau tanpa error *steady state*)
3. Menentukan parameter PID K_p, T_i, T_d dengan rumus-rumus sebagai berikut

$$K_p = \frac{2\xi}{\tau^* \omega_n K}$$
$$T_i = \frac{2\xi}{\omega_n}$$
$$T_d = \frac{1}{2\xi\omega_n}$$

dimana ξ adalah rasio redaman sistem, ω_n adalah frekuensi natural sistem, K adalah *gain* sistem, dan τ^* adalah konstanta waktu sistem hasil desain atau *desired*.

Selanjutnya akan dilakukan desain kontroler PID uuntuk sistem di atas dengan menggunakan metode ini.

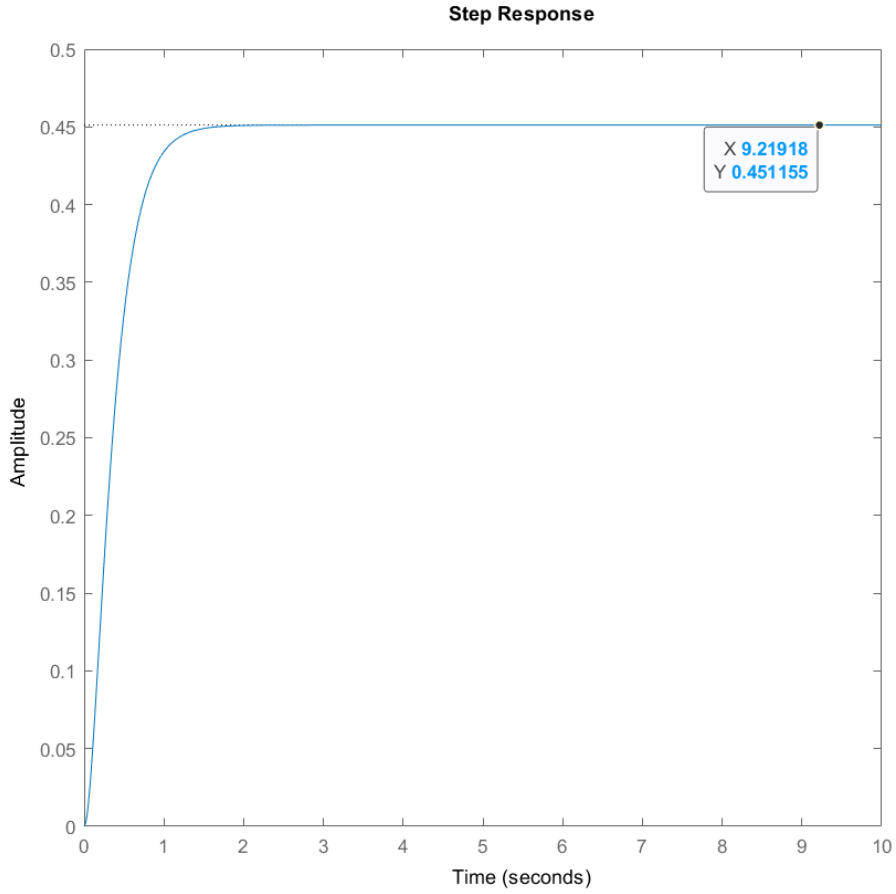
3.1 Contoh Soal (4)

Setelah dilakukan identifikasi atau *modelling* dari sistem di atas didapat *transfer function* sistem orde dua sebagai berikut

$$P_1(s) = \frac{12.7}{s^2 + 10.9s + 28.15}$$

Respon step dari sistem *open loop* dapat dicari dengan perintah sebagai berikut

```
s = tf('s');
P1 = 12.7/(s^2 + 10.9*s + 28.15)
step(P1,10) %respon step open loop
```




```
stepinfo(P1) %informasi respon step
```

```
ans = struct with fields:
         RiseTime: 0.6585
    TransientTime: 1.1572
     SettlingTime: 1.1572
      SettlingMin: 0.4074
      SettlingMax: 0.4511
        Overshoot: 0
       Undershoot: 0
             Peak: 0.4511
         PeakTime: 2.6181
```

Dapat dilihat bahwa error *steady state* atau ε_{ss} yang dihasilkan masih cukup besar yaitu sekitar 55%. Kemudian akan didesain kontroler PID sedemikian hingga keluaran sistem hasil mempunyai *settling time* atau $ts(\pm 2\%)$ sekitar 0.5 detik, $\varepsilon_{ss} = 0$ (*zero offset*), dan tidak memiliki *overshoot*. Berikut adalah langkah-langkah yang ditempuh

1. Menentukan parameter sistem K, ω_n, ξ berdasarkan *transfer function*

$$TF = \frac{K\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2} = \frac{12.7}{s^2 + 10.9s + 28.15}$$

$$\omega_n = \sqrt{28.15}$$

$$\xi = \frac{10.9}{2\omega_n}$$
$$K = \frac{12.7}{\omega_n^2}$$

Untuk menghitung parameter sistem tersebut, dapat digunakan *script* MATLAB sebagai berikut

```
wn = sqrt(28.15)
zeta = 10.9/(2*wn)
K = 12.7/wn^2
Val = (K*wn^2)/(s^2+2*zeta*wn*s+wn^2) %fungsi alih plant
```

2. Menentukan *gain* kontroler PID

$$K_p = \frac{2\xi}{\tau^* \omega_n K}$$
$$T_i = \frac{2\xi}{\omega_n}$$
$$T_d = \frac{1}{2\xi\omega_n}$$

Nilai dari konstanta waktu sistem orde satu yang diinginkan atau τ^* dapat dicari dengan persamaan karakteristik transien sistem orde satu pada bab sebelumnya yaitu

$$t_s^*(\pm 2\%) \approx 4\tau^*$$
$$\tau^* \approx \frac{t_s^*(\pm 2\%)}{4}$$
$$\tau^* \approx \frac{0.5}{4} = 0.125\ s$$

Nilai dari *gain* kontroler PID tersebut dapat dicari dengan menggunakan *script* MATLAB sebagai berikut

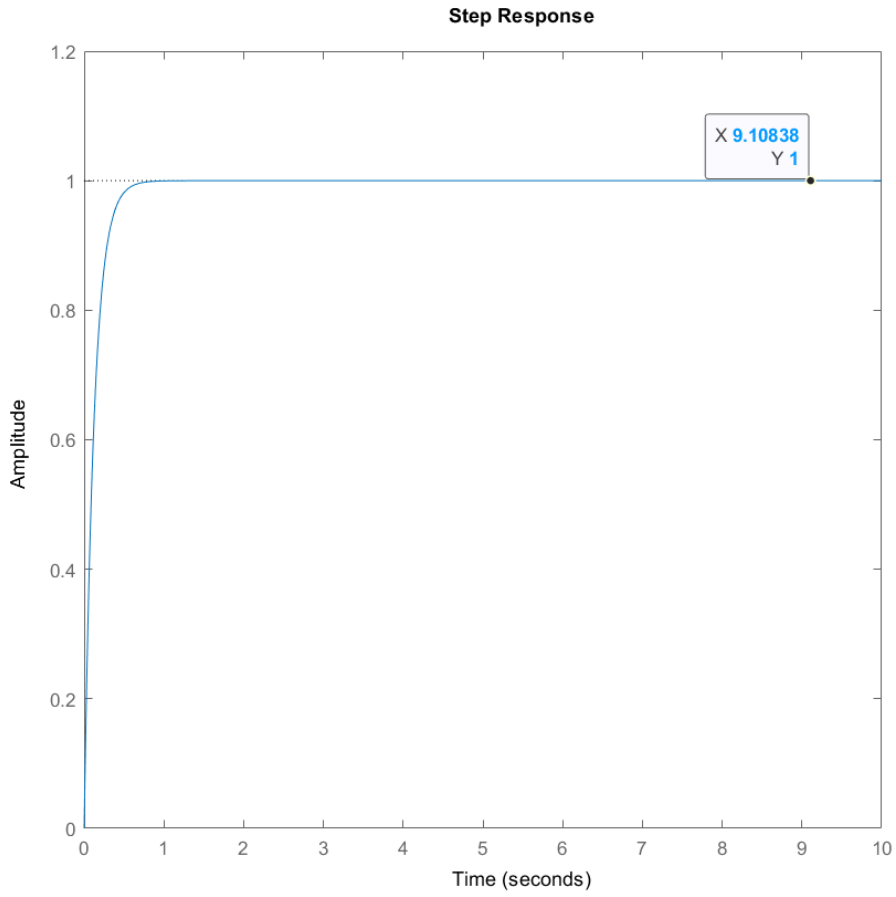
```
ts = 0.5; %settling time yang diinginkan
tau_star = ts/4; %konstanta waktu yang diinginkan

Kp = 2*zeta/(tau_star*wn*K); %gain proporsional
Ti = 2*zeta/wn;
Td = 1/(2*zeta*wn);
Ki = Kp/Ti; %gain integral
Kd = Kp*Td; %gain derivatif
```

```
C = pid(Kp,Ki,Kd) %kontroler PID
```

```
C =

             1
  Kp + Ki * --- + Kd * s
             s

  with Kp = 6.87, Ki = 17.7, Kd = 0.63

Continuous-time PID controller in parallel form.
Model Properties
```

3. Menguji respon *closed loop* dengan kontroler PID hasil desain
Respon *closed loop* dari sistem dengan menggunakan kontroler PID yang telah didesain dapat diuji dengan menggunakan perintah berikut

```
CLTF1 = feedback(P1*C,1); %transfer function closed loop
step(CLTF1,10) %respon step closed loop
```




```
stepinfo(CLTF1) %informasi respon step
```

```
ans = struct with fields:
```

```
      RiseTime: 0.2746
  TransientTime: 0.4890
   SettlingTime: 0.4890
    SettlingMin: 0.9045
    SettlingMax: 0.9993
      Overshoot: 0
     Undershoot: 0
           Peak: 0.9993
       PeakTime: 0.9153
```

Berdasarkan plot respon step dan step info, sistem sudah memenuhi kriteria desain yaitu tanpa *overshoot* dan error *steady state*, serta memiliki *settling time* sebesar 0.5 detik. Kemudian desain kontroler PID ini akan diimplementasikan di SIMULINK dengan model sistem tersebut.