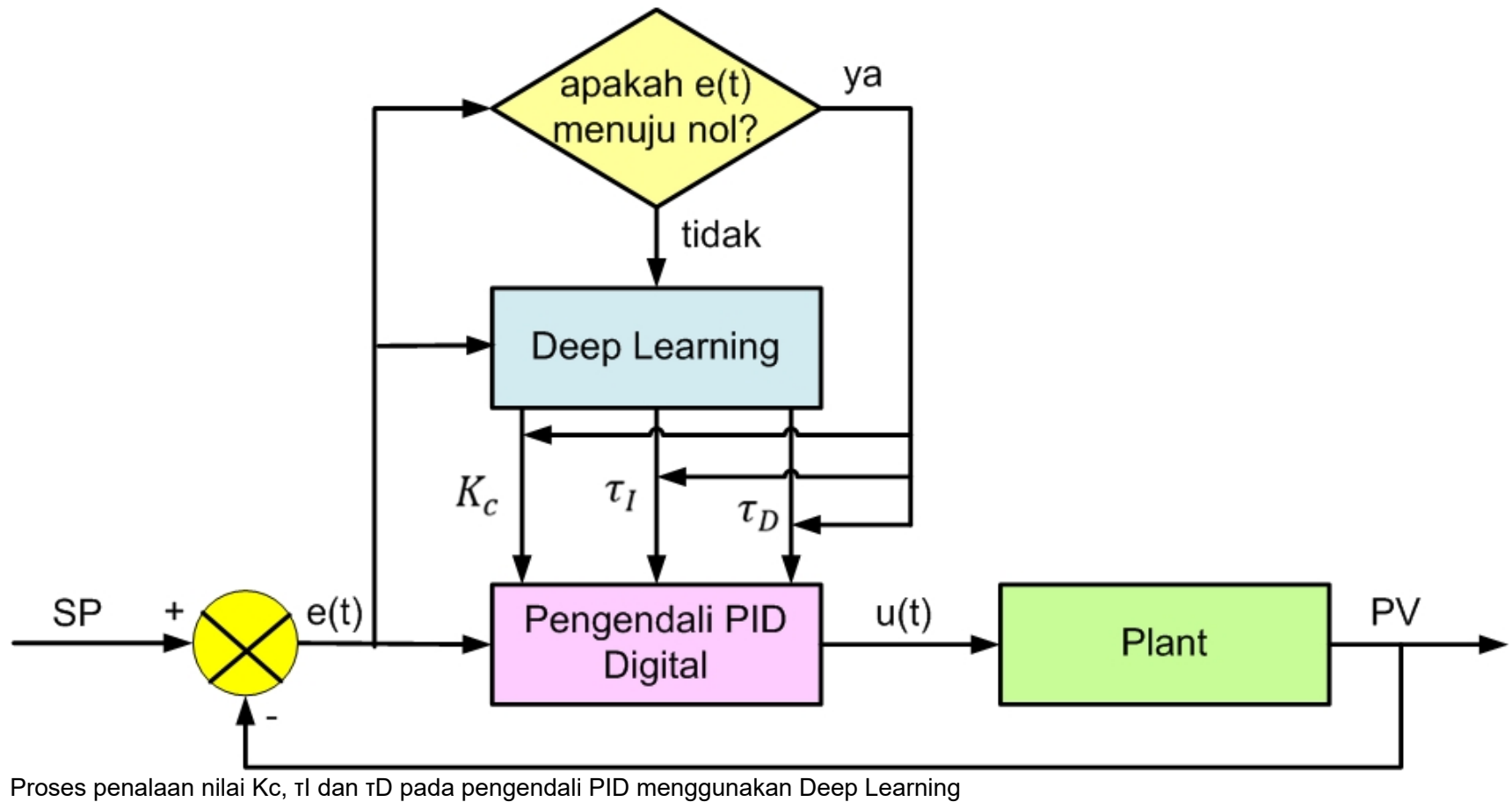


# Simulasi Penalaan Parameter PID Menggunakan Deep Learning

Oleh: Tim IO-T.NET (<https://io-t.net/itclab>)



## Library yang dibutuhkan

```
In [129]: import numpy as np # For matrix math
import matplotlib.pyplot as plt # For plotting

import sys # For printing
```

## Data Latih

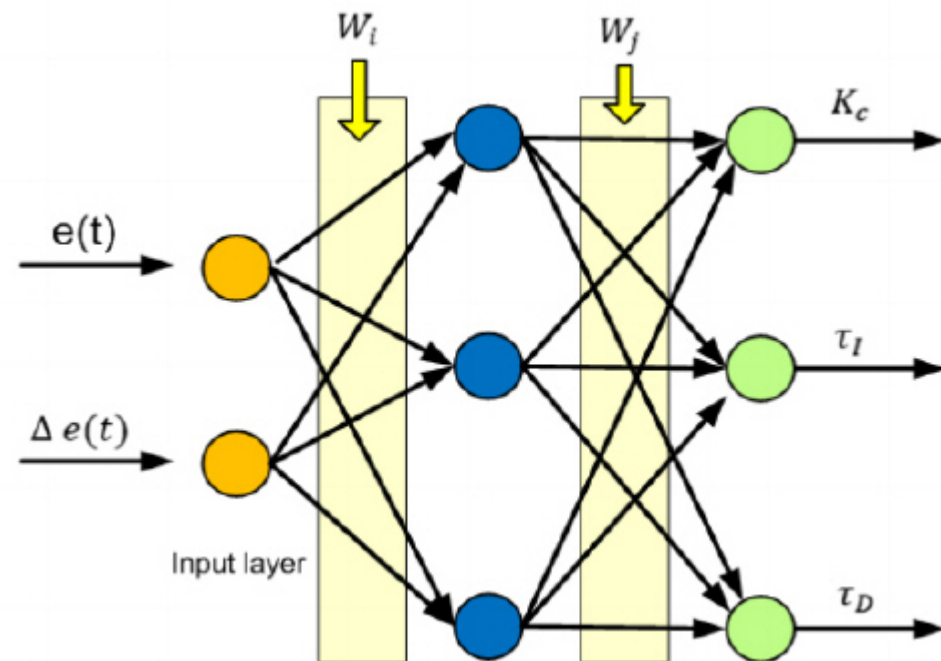
Misalkan error dan delta\_error ideal untuk membangkitkan gain PID Kc, taul dan tauD, sebagai berikut:

```
In [130]: # Data Latih.
X = np.array([
    [1, 1],
    [0.4, 1.2],
    [1.2, 0.1],
    [1, 0.1]
])

# Label untuk Data Latih.
y = np.array([
    [0.25, 4.31, 0.20],
    [0.2, 4.1, 0.1],
    [0.1, 4.0, 0],
    [0.1, 4.0, 0]
])
```

## Arsitektur Deep Learning

Arsitektur Deep Learning dengan Dua Masukan dan Tiga Keluaran



```
In [131]: # Impor `Sequential` dari `keras.models`
from keras.models import Sequential

# Impor `Dense` dari `keras.layers`
from keras.layers import Dense

# Inisialisasi konstruktor
model = Sequential()

# Tambahkan lapisan masukan
model.add(Dense(2, activation='sigmoid', input_shape=(2,)))

# Tambahkan satu lapisan tersembunyi
model.add(Dense(3, activation='sigmoid'))

# Tambahkan lapisan keluaran
model.add(Dense(3, activation='sigmoid'))
```

```
In [132]: # Bentuk keluaran model
model.output_shape

# Ringkasan model
model.summary()

# Konfigurasi model
model.get_config()

# Buat daftar semua tensor bobot
model.get_weights()
```

Model: "sequential\_3"

Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 2)	6
dense_10 (Dense)	(None, 3)	9
dense_11 (Dense)	(None, 3)	12

=====  
 Total params: 27  
 Trainable params: 27  
 Non-trainable params: 0  
 =====

```
Out[132]: [array([[ -1.0747703 ,  0.13438284],
          [-0.3540696 ,  0.89655507]]), dtype=float32),
          array([0., 0.], dtype=float32),
          array([[ -0.93595296, -0.42637336, -0.8650211 ],
          [-0.655743 , -0.68840504, -0.01161814]]), dtype=float32),
          array([0., 0., 0.], dtype=float32),
          array([[ -0.7498157 , -0.29627323, -0.36188388],
          [ 0.01366854, -0.21947813,  0.9318299 ],
          [-0.59939027,  0.19210243, -0.5551851 ]]), dtype=float32),
          array([0., 0., 0.], dtype=float32)]
```

Untuk pelatihan Deep Learning silahkan ketikkan skrip berikut.

```
In [133]: model.compile(loss='binary_crossentropy',
                        optimizer='adam',
                        metrics=['accuracy'])

model.fit(X, y, epochs=100, batch_size=1, verbose=1)
```

```
Epoch 1/100
4/4 [=====] - 0s 2ms/step - loss: 0.7517 - accuracy: 0.0000e+00
Epoch 2/100
4/4 [=====] - 0s 2ms/step - loss: 0.7375 - accuracy: 0.0000e+00
Epoch 3/100
4/4 [=====] - 0s 4ms/step - loss: 0.7235 - accuracy: 0.2500
Epoch 4/100
4/4 [=====] - 0s 3ms/step - loss: 0.7095 - accuracy: 0.5000
Epoch 5/100
4/4 [=====] - 0s 2ms/step - loss: 0.6956 - accuracy: 1.0000
Epoch 6/100
4/4 [=====] - 0s 3ms/step - loss: 0.6817 - accuracy: 1.0000
Epoch 7/100
4/4 [=====] - 0s 2ms/step - loss: 0.6680 - accuracy: 1.0000
Epoch 8/100
4/4 [=====] - 0s 2ms/step - loss: 0.6541 - accuracy: 1.0000
Epoch 9/100
4/4 [=====] - 0s 3ms/step - loss: 0.6406 - accuracy: 1.0000
Epoch 10/100
4/4 [=====] - 0s 2ms/step - loss: 0.6271 - accuracy: 1.0000
```

```
In [134]: Hasil_Prediksi_Keras = model.predict(X)
print(Hasil_Prediksi_Keras)
```

```
[[0.2116262 0.7111814 0.26767278]
 [0.21551791 0.70976496 0.26997936]
 [0.21004573 0.71002096 0.27128503]
 [0.21138996 0.70934224 0.27257684]]
```

## Dicoba diberi masukan e(t) sembarang

Pengujian ke-1

```
In [135]: ujicoba1 = np.array([
          [1, 1]
          ])
```

```
In [136]: ujicoba1
```

```
Out[136]: array([[1, 1]])
```

```
In [137]: outDL = model.predict(ujicoba1)
```

```
In [138]: outDL
```

```
Out[138]: array([[0.2116262 , 0.7111814 , 0.26767278]], dtype=float32)
```

```
In [139]: result_Kc   = outDL[0,0]
          result_tauI = outDL[0,1]
          result_tauD = outDL[0,2]
```

```
In [140]: result_Kc
```

```
Out[140]: 0.2116262
```

```
In [141]: result_tauI
```

```
Out[141]: 0.7111814
```

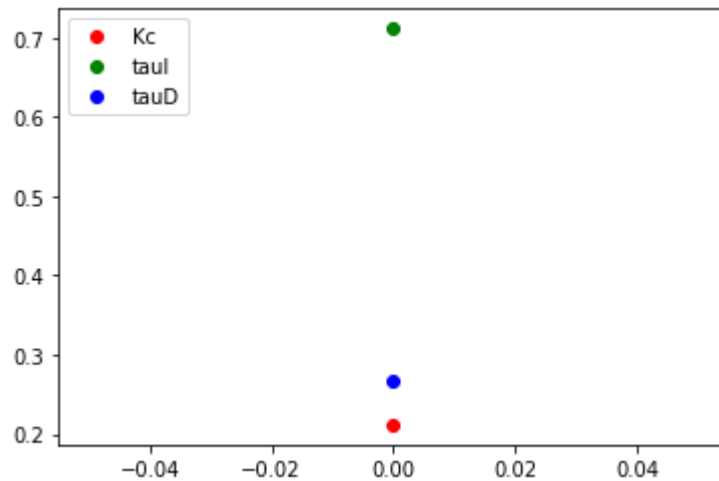
```
In [142]: result_tauD
```

```
Out[142]: 0.26767278
```

```
In [143]: # Visualize
plt.plot(result_Kc, 'ro', label='Kc')
plt.plot(result_tauI, 'go', label='tauI')
plt.plot(result_tauD, 'bo', label='tauD')

#plt.xlabel('Kc, tauI, tauD');
#plt.legend((result_Kc, result_tauI, result_tauD), ('Kc', 'tauI', 'tauD'))

plt.legend(loc='upper left')
#pylab.ylim(-1.5, 2.0)
plt.show()
```



```
In [144]: import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import ipywidgets as wg
from IPython.display import display
```

```

In [145]: n = 100 # time points to plot
          tf = 50.0 # final time
          SP_start = 2.0 # time of set point change

def process(y,t,u):
    Kp = 4.0
    taup = 3.0
    thetap = 1.0
    if t<(thetap+SP_start):
        dydt = 0.0 # time delay
    else:
        dydt = (1.0/taup) * (-y + Kp * u)
    return dydt

def pidPlot(Kc,tauI,tauD):
    t = np.linspace(0,tf,n) # create time vector
    P= np.zeros(n) # initialize proportional term
    I = np.zeros(n) # initialize integral term
    D = np.zeros(n) # initialize derivative term
    e = np.zeros(n) # initialize error
    OP = np.zeros(n) # initialize controller output
    PV = np.zeros(n) # initialize process variable
    SP = np.zeros(n) # initialize setpoint
    SP_step = int(SP_start/(tf/(n-1))+1) # setpoint start
    SP[0:SP_step] = 0.0 # define setpoint
    SP[SP_step:n] = 4.0 # step up
    y0 = 0.0 # initial condition
    # Loop through all time steps
    for i in range(1,n):
        # simulate process for one time step
        ts = [t[i-1],t[i]] # time interval
        y = odeint(process,y0,ts,args=(OP[i-1],)) # compute next step
        y0 = y[1] # record new initial condition
        # calculate new OP with PID
        PV[i] = y[1] # record PV
        e[i] = SP[i] - PV[i] # calculate error = SP - PV
        dt = t[i] - t[i-1] # calculate time step
        P[i] = Kc * e[i] # calculate proportional term
        I[i] = I[i-1] + (Kc/tauI) * e[i] * dt # calculate integral term
        D[i] = -Kc * tauD * (PV[i]-PV[i-1])/dt # calculate derivative term
        OP[i] = P[i] + I[i] + D[i] # calculate new controller output

```



```

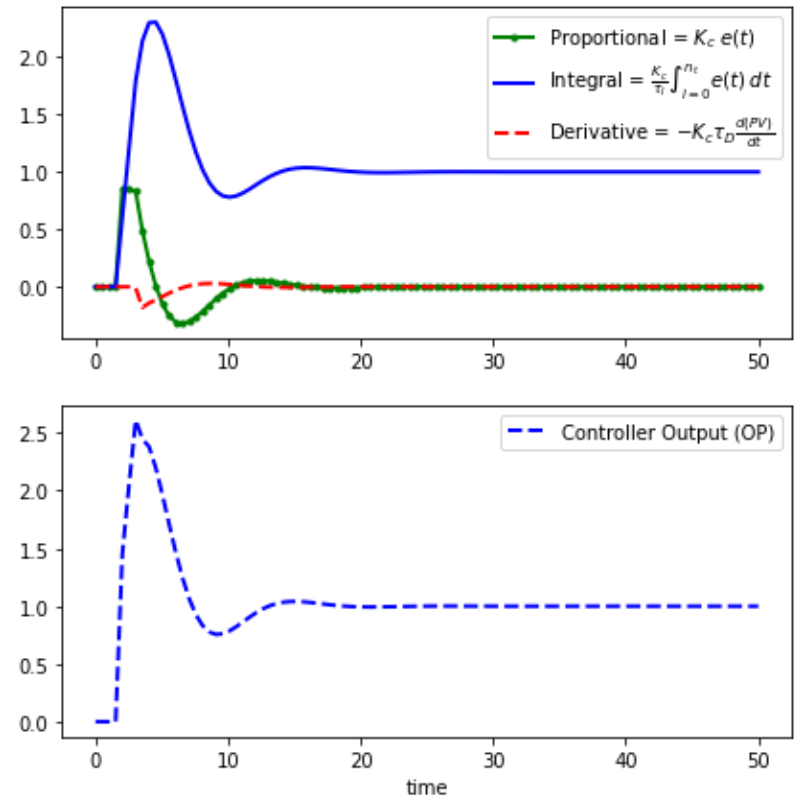
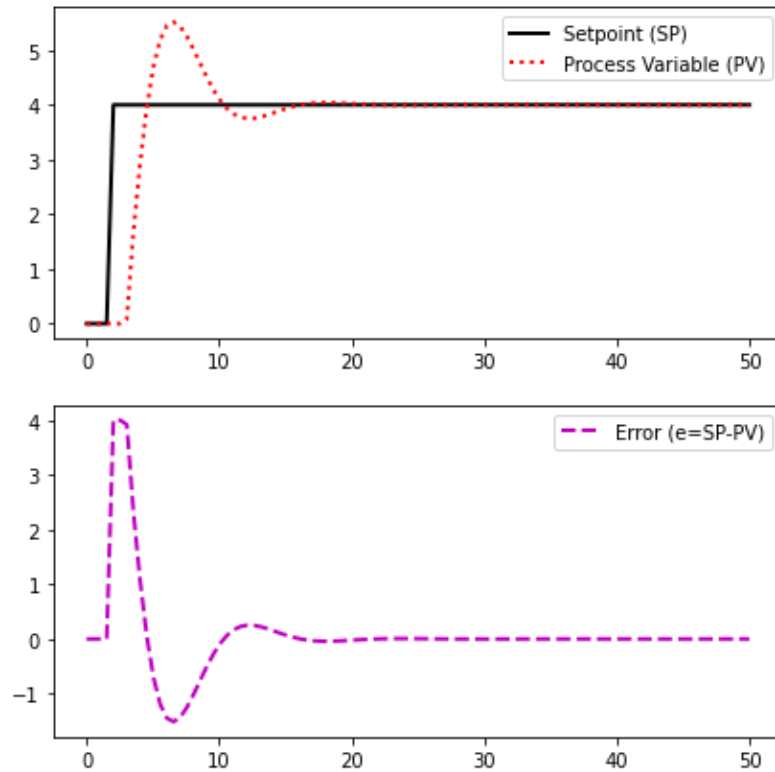
# plot PID response
plt.figure(1,figsize=(15,7))
plt.subplot(2,2,1)
plt.plot(t,SP,'k-',linewidth=2,label='Setpoint (SP)')
plt.plot(t,PV,'r:',linewidth=2,label='Process Variable (PV)')
plt.legend(loc='best')
plt.subplot(2,2,2)
plt.plot(t,P,'g.-',linewidth=2,label=r'Proportional = $K_c \; e(t)$')
plt.plot(t,I,'b-',linewidth=2,label=r'Integral = $\frac{K_c}{\tau_I} \int_{i=0}^{n_t} e(t) \; dt$')
plt.plot(t,D,'r--',linewidth=2,label=r'Derivative = $-K_c \tau_D \frac{d(PV)}{dt}$')
plt.legend(loc='best')
plt.subplot(2,2,3)
plt.plot(t,e,'m--',linewidth=2,label='Error (e=SP-PV)')
plt.legend(loc='best')
plt.subplot(2,2,4)
plt.plot(t,OP,'b--',linewidth=2,label='Controller Output (OP)')
plt.legend(loc='best')
plt.xlabel('time')

Kc_slide = wg.FloatSlider(value=result_Kc,min=-0.2,max=1.0,step=0.05)
tauI_slide = wg.FloatSlider(value=result_tauI,min=0.01,max=5.0,step=0.1)
tauD_slide = wg.FloatSlider(value=result_tauD,min=0.0,max=1.0,step=0.1)
wg.interact(pidPlot, Kc=Kc_slide, tauI=tauI_slide, tauD=tauD_slide)

```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

Out[145]: <function \_\_main\_\_.pidPlot(Kc, tauI, tauD)>



## Dicoba diberi masukan $e(t)$ sembarang

Pengujian ke-2

```
In [146]: ujicoba2 = np.array([
           [0.4, 1.2]
           ])
```

```
In [147]: ujicoba2
```

```
Out[147]: array([[0.4, 1.2]])
```

```
In [148]: outDL = model.predict(ujicoba2)
```

```
In [149]: outDL
```

```
Out[149]: array([[0.21551791, 0.70976496, 0.26997936]], dtype=float32)
```

```
In [150]: result_Kc = outDL[0,0]
result_tauI = outDL[0,1]
result_tauD = outDL[0,2]
```

```
In [151]: result_Kc
```

```
Out[151]: 0.21551791
```

```
In [152]: result_tauI
```

```
Out[152]: 0.70976496
```

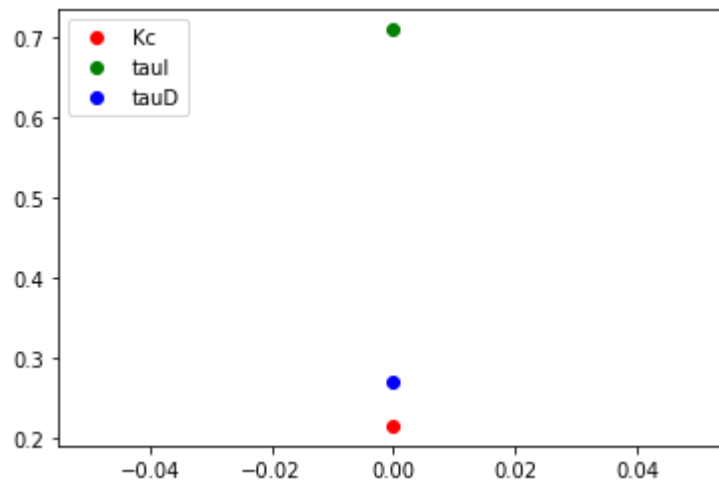
```
In [153]: result_tauD
```

```
Out[153]: 0.26997936
```

```
In [154]: # Visualize
plt.plot(result_Kc, 'ro', label='Kc')
plt.plot(result_tauI, 'go', label='tauI')
plt.plot(result_tauD, 'bo', label='tauD')

#plt.xlabel('Kc, tauI, tauD');
#plt.legend((result_Kc, result_tauI, result_tauD), ('Kc', 'tauI', 'tauD'))

plt.legend(loc='upper left')
#pylab.ylim(-1.5, 2.0)
plt.show()
```



```
In [155]: import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import ipywidgets as wg
from IPython.display import display
```

```

In [156]: n = 100 # time points to plot
          tf = 50.0 # final time
          SP_start = 2.0 # time of set point change

def process(y,t,u):
    Kp = 4.0
    taup = 3.0
    thetap = 1.0
    if t<(thetap+SP_start):
        dydt = 0.0 # time delay
    else:
        dydt = (1.0/taup) * (-y + Kp * u)
    return dydt

def pidPlot(Kc,tauI,tauD):
    t = np.linspace(0,tf,n) # create time vector
    P= np.zeros(n) # initialize proportional term
    I = np.zeros(n) # initialize integral term
    D = np.zeros(n) # initialize derivative term
    e = np.zeros(n) # initialize error
    OP = np.zeros(n) # initialize controller output
    PV = np.zeros(n) # initialize process variable
    SP = np.zeros(n) # initialize setpoint
    SP_step = int(SP_start/(tf/(n-1))+1) # setpoint start
    SP[0:SP_step] = 0.0 # define setpoint
    SP[SP_step:n] = 4.0 # step up
    y0 = 0.0 # initial condition
    # Loop through all time steps
    for i in range(1,n):
        # simulate process for one time step
        ts = [t[i-1],t[i]] # time interval
        y = odeint(process,y0,ts,args=(OP[i-1],)) # compute next step
        y0 = y[1] # record new initial condition
        # calculate new OP with PID
        PV[i] = y[1] # record PV
        e[i] = SP[i] - PV[i] # calculate error = SP - PV
        dt = t[i] - t[i-1] # calculate time step
        P[i] = Kc * e[i] # calculate proportional term
        I[i] = I[i-1] + (Kc/tauI) * e[i] * dt # calculate integral term
        D[i] = -Kc * tauD * (PV[i]-PV[i-1])/dt # calculate derivative term
        OP[i] = P[i] + I[i] + D[i] # calculate new controller output

```

```

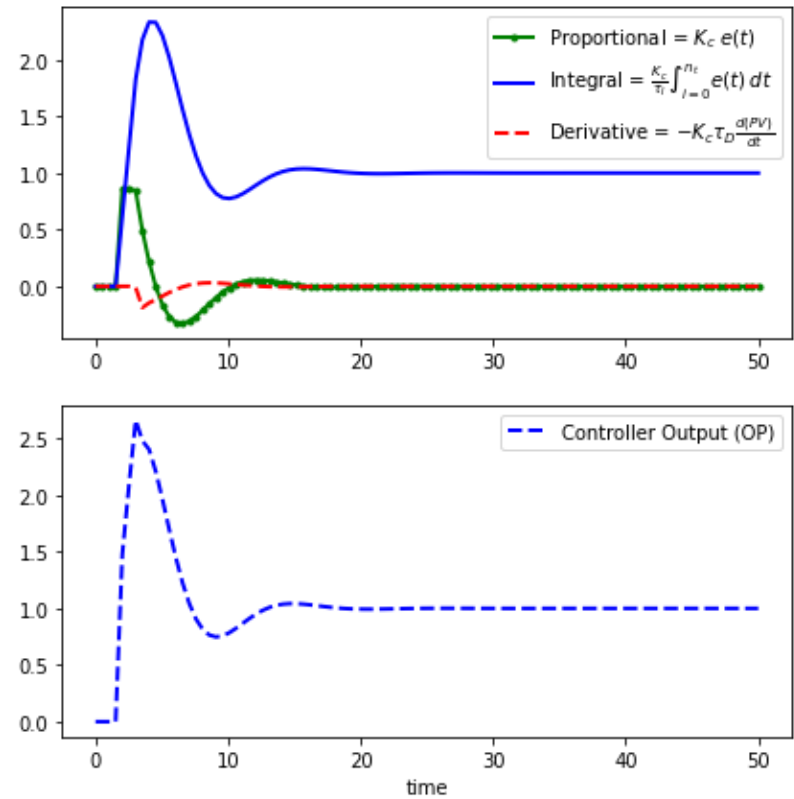
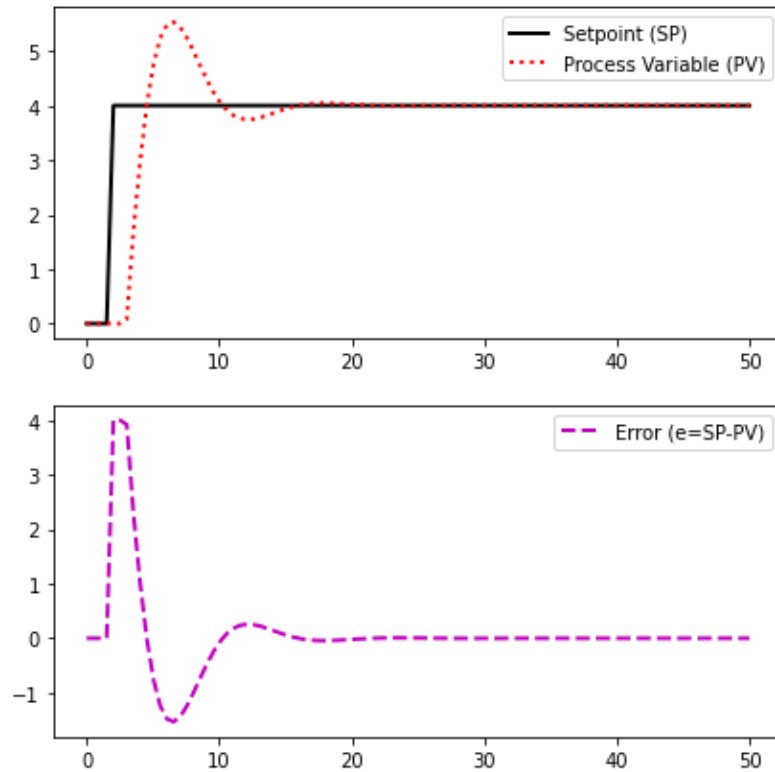
# plot PID response
plt.figure(1,figsize=(15,7))
plt.subplot(2,2,1)
plt.plot(t,SP,'k-',linewidth=2,label='Setpoint (SP)')
plt.plot(t,PV,'r:',linewidth=2,label='Process Variable (PV)')
plt.legend(loc='best')
plt.subplot(2,2,2)
plt.plot(t,P,'g.-',linewidth=2,label=r'Proportional = $K_c \; e(t)$')
plt.plot(t,I,'b-',linewidth=2,label=r'Integral = $\frac{K_c}{\tau_I} \int_{i=0}^{n_t} e(t) \; dt$')
plt.plot(t,D,'r--',linewidth=2,label=r'Derivative = $-K_c \tau_D \frac{d(PV)}{dt}$')
plt.legend(loc='best')
plt.subplot(2,2,3)
plt.plot(t,e,'m--',linewidth=2,label='Error (e=SP-PV)')
plt.legend(loc='best')
plt.subplot(2,2,4)
plt.plot(t,OP,'b--',linewidth=2,label='Controller Output (OP)')
plt.legend(loc='best')
plt.xlabel('time')

Kc_slide = wg.FloatSlider(value=result_Kc,min=-0.2,max=1.0,step=0.05)
tauI_slide = wg.FloatSlider(value=result_tauI,min=0.01,max=5.0,step=0.1)
tauD_slide = wg.FloatSlider(value=result_tauD,min=0.0,max=1.0,step=0.1)
wg.interact(pidPlot, Kc=Kc_slide, tauI=tauI_slide, tauD=tauD_slide)

```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
Out[156]: <function __main__.pidPlot(Kc, tauI, tauD)>
```



## Dicoba diberi masukan $e(t)$ sembarang

Pengujian ke-3

```
In [157]: ujicoba3 = np.array([
           [1.2, 0.1]
           ])
```

```
In [158]: ujicoba3
```

```
Out[158]: array([[1.2, 0.1]])
```

```
In [159]: outDL = model.predict(ujicoba3)
```

```
In [160]: outDL
```

```
Out[160]: array([[0.21004573, 0.71002096, 0.27128503]], dtype=float32)
```

```
In [161]: result_Kc = outDL[0,0]  
result_tauI = outDL[0,1]  
result_tauD = outDL[0,2]
```

```
In [162]: result_Kc
```

```
Out[162]: 0.21004573
```

```
In [163]: result_tauI
```

```
Out[163]: 0.71002096
```

```
In [164]: result_tauD
```

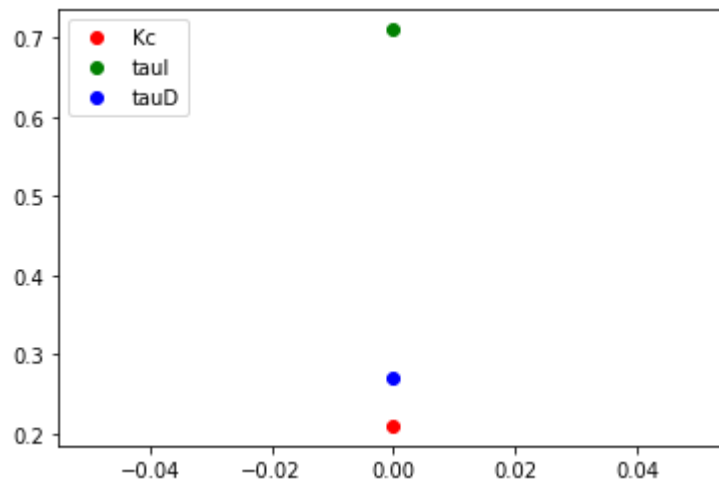
```
Out[164]: 0.27128503
```



```
In [165]: # Visualize
plt.plot(result_Kc, 'ro', label='Kc')
plt.plot(result_tauI, 'go', label='tauI')
plt.plot(result_tauD, 'bo', label='tauD')

#plt.xlabel('Kc, tauI, tauD');
#plt.legend((result_Kc, result_tauI, result_tauD), ('Kc', 'tauI', 'tauD'))

plt.legend(loc='upper left')
#pylab.ylim(-1.5, 2.0)
plt.show()
```



```
In [166]: import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import ipywidgets as wg
from IPython.display import display
```

```

In [167]: n = 100 # time points to plot
          tf = 50.0 # final time
          SP_start = 2.0 # time of set point change

def process(y,t,u):
    Kp = 4.0
    taup = 3.0
    thetap = 1.0
    if t<(thetap+SP_start):
        dydt = 0.0 # time delay
    else:
        dydt = (1.0/taup) * (-y + Kp * u)
    return dydt

def pidPlot(Kc,tauI,tauD):
    t = np.linspace(0,tf,n) # create time vector
    P= np.zeros(n) # initialize proportional term
    I = np.zeros(n) # initialize integral term
    D = np.zeros(n) # initialize derivative term
    e = np.zeros(n) # initialize error
    OP = np.zeros(n) # initialize controller output
    PV = np.zeros(n) # initialize process variable
    SP = np.zeros(n) # initialize setpoint
    SP_step = int(SP_start/(tf/(n-1))+1) # setpoint start
    SP[0:SP_step] = 0.0 # define setpoint
    SP[SP_step:n] = 4.0 # step up
    y0 = 0.0 # initial condition
    # Loop through all time steps
    for i in range(1,n):
        # simulate process for one time step
        ts = [t[i-1],t[i]] # time interval
        y = odeint(process,y0,ts,args=(OP[i-1],)) # compute next step
        y0 = y[1] # record new initial condition
        # calculate new OP with PID
        PV[i] = y[1] # record PV
        e[i] = SP[i] - PV[i] # calculate error = SP - PV
        dt = t[i] - t[i-1] # calculate time step
        P[i] = Kc * e[i] # calculate proportional term
        I[i] = I[i-1] + (Kc/tauI) * e[i] * dt # calculate integral term
        D[i] = -Kc * tauD * (PV[i]-PV[i-1])/dt # calculate derivative term
        OP[i] = P[i] + I[i] + D[i] # calculate new controller output

```

```

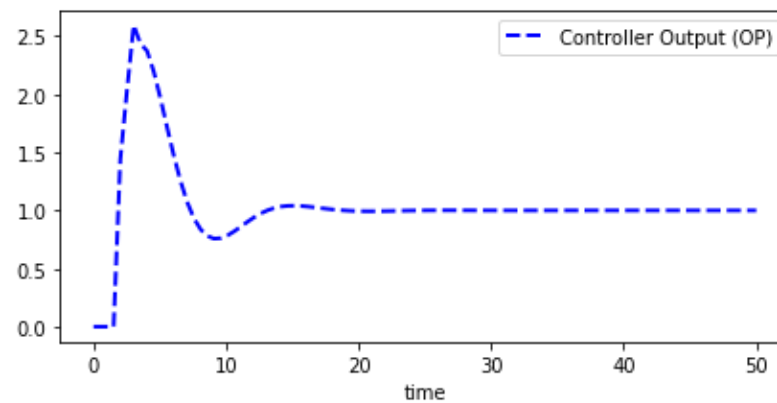
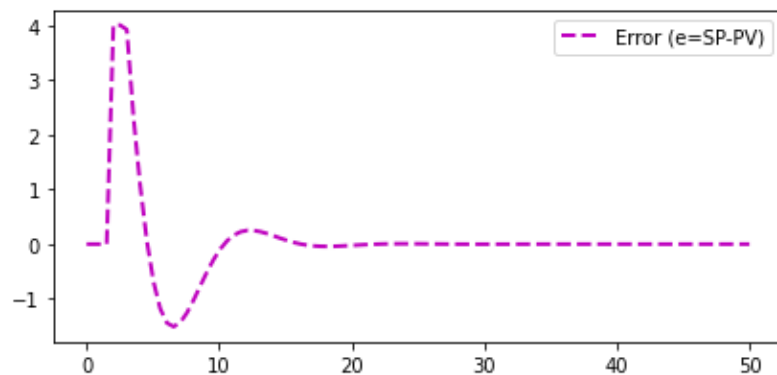
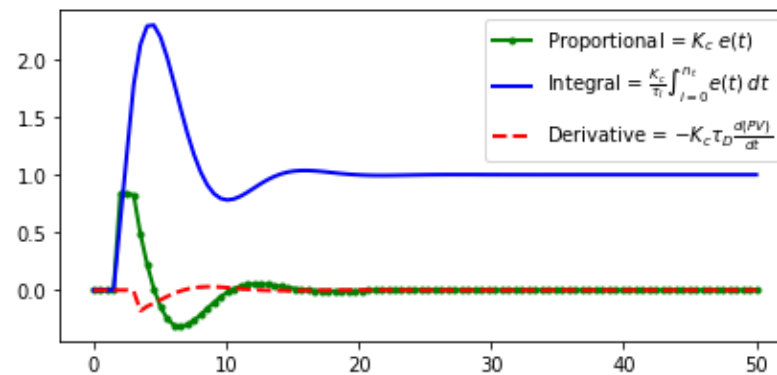
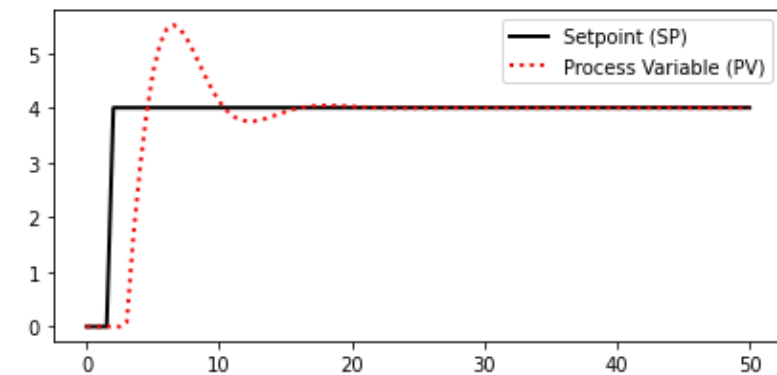
# plot PID response
plt.figure(1,figsize=(15,7))
plt.subplot(2,2,1)
plt.plot(t,SP,'k-',linewidth=2,label='Setpoint (SP)')
plt.plot(t,PV,'r:',linewidth=2,label='Process Variable (PV)')
plt.legend(loc='best')
plt.subplot(2,2,2)
plt.plot(t,P,'g.-',linewidth=2,label=r'Proportional = $K_c \; e(t)$')
plt.plot(t,I,'b-',linewidth=2,label=r'Integral = $\frac{K_c}{\tau_I} \int_{i=0}^{n_t} e(t) \; dt$')
plt.plot(t,D,'r--',linewidth=2,label=r'Derivative = $-K_c \tau_D \frac{d(PV)}{dt}$')
plt.legend(loc='best')
plt.subplot(2,2,3)
plt.plot(t,e,'m--',linewidth=2,label='Error (e=SP-PV)')
plt.legend(loc='best')
plt.subplot(2,2,4)
plt.plot(t,OP,'b--',linewidth=2,label='Controller Output (OP)')
plt.legend(loc='best')
plt.xlabel('time')

Kc_slide = wg.FloatSlider(value=result_Kc,min=-0.2,max=1.0,step=0.05)
tauI_slide = wg.FloatSlider(value=result_tauI,min=0.01,max=5.0,step=0.1)
tauD_slide = wg.FloatSlider(value=result_tauD,min=0.0,max=1.0,step=0.1)
wg.interact(pidPlot, Kc=Kc_slide, tauI=tauI_slide, tauD=tauD_slide)

```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
Out[167]: <function __main__.pidPlot(Kc, tauI, tauD)>
```



In [ ]: