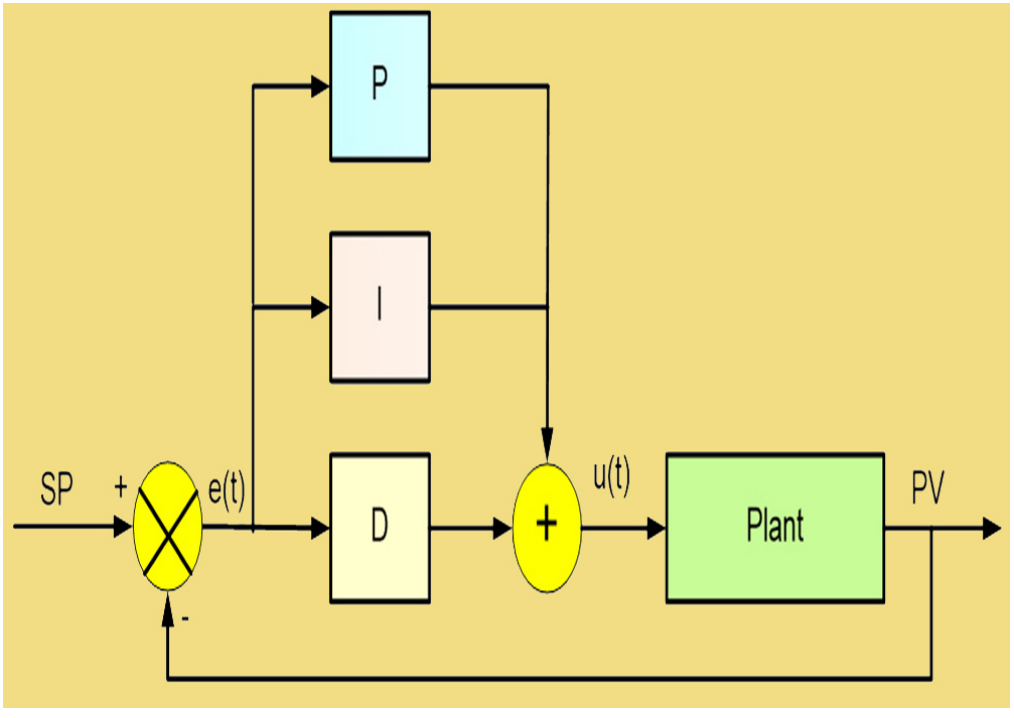
Mengenal Kendali PID

Administrator | 22 April 2022

Mengenal Sistem Kendali Proporsional Integral dan Derivatif (PID)



time yang sangat cepat. Aksi kendali integral mempunyai keunggulan untuk memperkecil error. Sedangkan aksi kendali derivatif mempunyai keunggulan untuk memperkecil error atau mereduam overshoot. Tujuan penggabungan ketiga aksi kendali ini agar dihasilkan keluaran dengan risetime yang cepat dan error yang kecil.

Pengendali PID dalam kerjanya secara otomatis menyesuaikan keluaran kendali berdasarkan perbedaan antara Setpoint (SP) dan variabel proses yang terukur (PV), sebagai error pengendalian e(t). Nilai keluaran Pengendali u(t) ditransfer sebagai masukan sistem. Masing-masing hubungan yang digunakan seperti terlihat pada Persamaan (1) dan (2) **(BYU, 2018a)**.

$$e(t) = SP - PV \quad (1)$$

$$u(t) = u_{bias} + K_c\, e(t) + \frac{K_c}{\tau_I} \int_0^t e(t)dt - K_c\, \tau_D\, \frac{d(PV)}{dt} \quad (2)$$

Istilah ubias adalah konstanta yang biasanya diatur ke nilai u(t) ketika pengendali pertama beralih dari mode manual ke mode otomatis. Ini memberikan transfer "bumpless" jika kesalahannya nol ketika pengendali dihidupkan. Ketiga nilai penalaan atau tuning untuk pengendali PID adalah gain Kc, konstanta waktu integral Tho_I, dan konstanta waktu derivatif Tho_D.

Nilai Kc adalah penguat error proporsional dan istilah integral membuat pengendali lebih agresif dalam menanggapi error dari Setpoint. Konstanta waktu integral Tho_I (juga dikenal sebagai waktu reset integral) harus positif dan memiliki satuan waktu. Karena Tho_I semakin kecil, istilah integralnya lebih besar karena Tho_I berada di penyebut. Konstanta waktu derivatif Tho_D juga memiliki satuan waktu dan harus positif.

Setpoint (SP) adalah nilai target, dan variabel proses (PV) adalah nilai terukur yang mungkin menyimpang dari nilai yang diinginkan. Kesalahan dari setpoint adalah perbedaan antara SP dan PV dan didefinisikan sebagai error, e(t) = SP - PV.

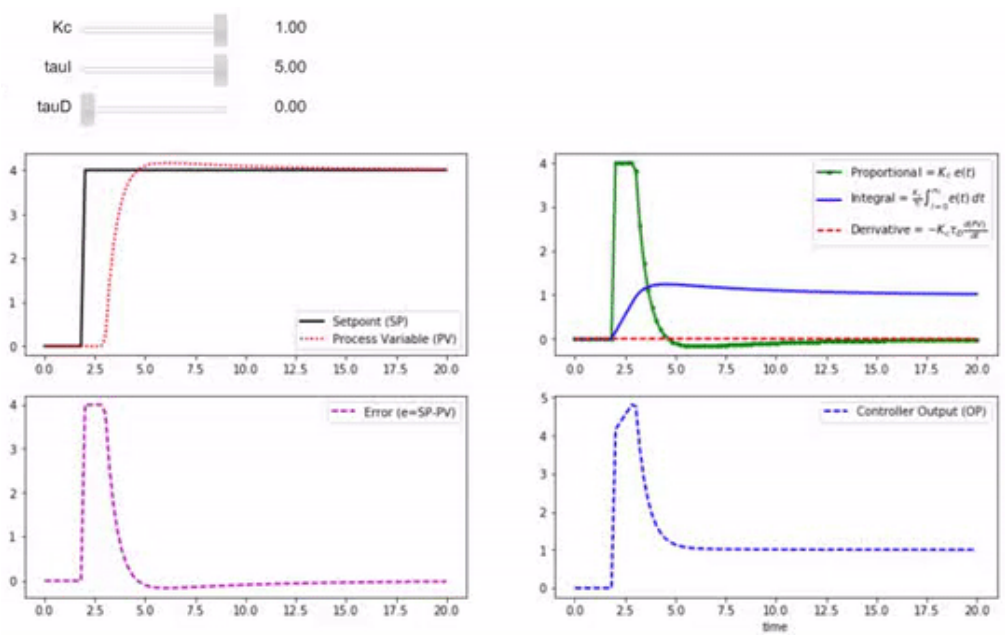
Selanjutnya, untuk keperluan imlementasi digunakan Pengendali PID Diskrit. Pengendali digital diimplementasikan dengan periode sampling diskrit. Bentuk terpisah dari persamaan PID diperlukan untuk memperkirakan integral dan derivatif dari error. Modifikasi ini menggantikan bentuk kontinyu integral dengan penjumlahan error dan menggunakan delta_t sebagai waktu antara contoh pengambilan sampel dan nt sebagai jumlah contoh pengambilan sampel. Ini juga menggantikan derivatif dengan versi turunannya atau metode lain yang disaring untuk memperkirakan kemiringan instan (PV). Persamaan (2) jika dinyatakan kedalam bentuk digital seperti diperlihatkan pada Persamaan (3) **(BYU, 2018a)**.

$$u(t) = u_{bias} + K_c\, e(t) + \frac{K_c}{\tau_I} \sum_{i=1}^{n_t} e_i(t)\, \Delta t - K_c\, \tau_D\, \frac{PV_{n_t} - PV_{n_t-1}}{\Delta t} \quad (3)$$

Dari Persamaan (3) dapat dilihat tiga parameter penentu keberhasilan proses kendali, yaitu gain Kc, konstanta waktu integral Tho_I dan konstanta waktu derivatif Tho_D. Proses pencarian atau pengaturan atau penalaan agar diperoleh nilai Kc, Tho_I dan Tho_D terbaik ini umumnya disebut dengan proses penalaan atau tuning.

Simulasi Sistem Kendali PID

Untuk bisa membayangkan pengaruh perubahan nilai parameter gain Kc, konstanta waktu integral Tho_I dan konstanta waktu derivatif Tho_D, terhadap kinerja Sistem Kendali PID, dapat dilihat dari simulasi berikut **(BYU, 2018a)**.



Program Simulasi Sistem Kendali PID

Untuk mensimulasikan Sistem Kendali PID di atas, silahkan copy-paste script program menggunakan Python Jupyter Notebook berikut ini **(BYU, 2018a)**.

```
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import ipywidgets as wg
from IPython.display import display
```

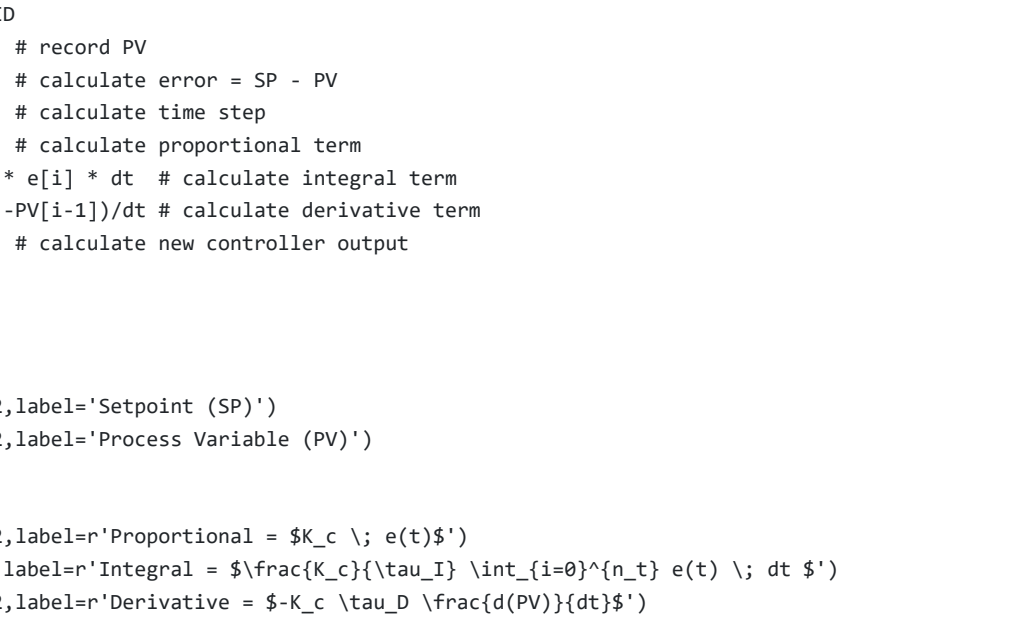
```
def process(y,t,u):
    Kp = 4.0
    taup = 3.0
    thetap = 1.0
    if t<(thetap+SP_start):
        dydt = 0.0  # time delay
    else:
        dydt = (1.0/taup) * (-y + Kp * u)
    return dydt

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=0.1,min=-0.2,max=1.0,step=0.05)
tauI_slide = wg.FloatSlider(value=4.0,min=0.01,max=5.0,step=0.1)
tauD_slide = wg.FloatSlider(value=0.0,min=0.0,max=1.0,step=0.1)
wg.interact(pidPlot, Kc=Kc_slide, tauI=tauI_slide, tauD=tauD_slide)
```

Jika dijalankan, maka hasilnya seperti terlihat pada gambar di bawah ini.



Download Program Simulasi Kendali PID dalam Python Jupyter Notebook (silahkan klik-kanan Save link as), **di sini : pid_widget.ipynb.**

Administrator | 10 April 2022

Kit iTCLab Test 2

Administrator | 10 April 2022

Riset Kendali PID Dasar

Administrator | 15 April 2022

Riset PWM iTCLab

Administrator | 16 April 2022

Arduino-Python iTCLab Test

Administrator | 16 April 2022

Arduino-Python PID Test

Administrator | 16 April 2022

Riset PID-iTCLab GUI

Administrator | 16 April 2022

Riset IoT Basic

Administrator | 17 April 2022

Riset IoT On/Off PWM

Administrator | 18 April 2022

Riset PID dengan Arduino

Administrator | 20 April 2022

Riset IoT PID Monitor

Administrator | 20 April 2022

Riset IoT PID Control

Administrator | 03 May 2022

Riset Deep Learning - XOR

Administrator | 22 August 2022

Riset Deep Learning - PID

Administrator | 22 August 2022

Riset Deep - PID - iTCLab

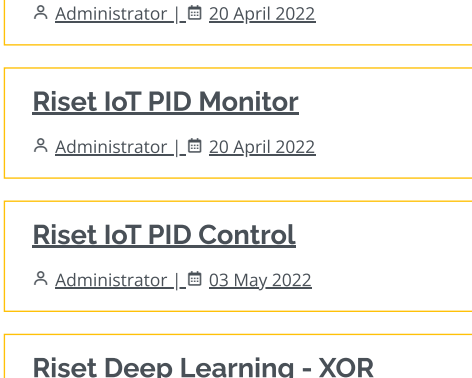
Administrator | 22 August 2022

Riset Deep - PID - iTCLab - IoT

Administrator | 22 August 2022

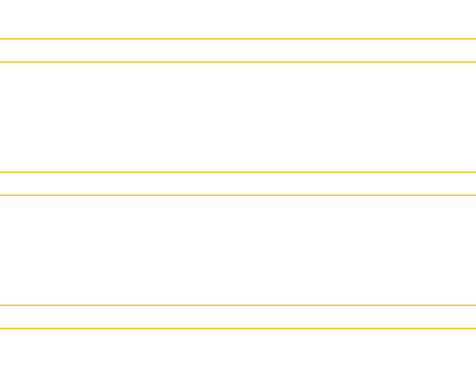
Lihat semua Riset iTCLab

Artikel lainnya



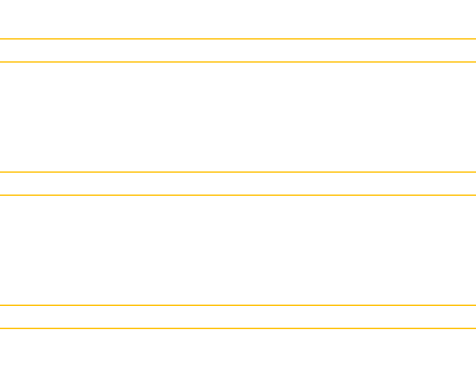
Pengu

PENGUJI



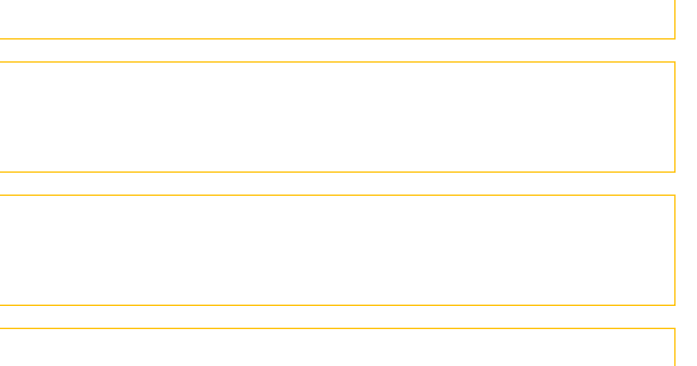
Pengu

PENGUJI



Peng

PROGRAM



Kit iTCLab Arduino-Python

PEMBACAAN KIT ITCLAB DENG...