Supplementary Material

The connection between KINGVIEW and Python is achieved by the program codes and the algorithm program of conventional PID and BPNN self-tuning PID are all completed in Python. This part aims to characterize how we finished these tasks.

1 Process realizing and logging OPC Date Access

The OPC Data Access can also be implemented using program codes through related functions. The variables in OPC Date Access cannot be written to directly like other data exchange agreement, and you need to log the collected data firstly to write desired data to them. In OPC Data Access, note that, logging takes place at group level; due to impossibility in logging inactive items in a group, data change records may not include all items, and OPC logging tasks are not guaranteed to complete, and logged data is not guaranteed to be regularly sampled. In terms of the special situation above, we added groups for each item and logged the first specified data when collecting real-time data every second we sample, by which we can write data to variables anytime we want.

2 Program codes realizing two kinds of PID algorithm in Python

The traditional and BPNN (Backward-Propagation Neural Network) self-tuning PID output is calculated by the Python program codes based on the real-time collecting data every second we sample, which is listed below.

2.1 The exclusive PID Function

```
def pid(pv, sv, e_1, e_0, t, kp, ti, td):
du = kp*(sv-pv)*(1+t/ti+td/t)-kp*e_1*(1+2*td/t)+kp*e_0*td/t
return du
        %This function calculates output value of the digital PID based process value.
```

2.2 The PLC-Data Reading Function

```
import snap7
from snap7 import util, client
def get_plc_data():
  %This function is used to read the PLC data [1) PV, the process value of steam drum; 2) u, the
                                valve opening] and write to the PLC modules [u, the valve opening].
my_plc = client.Client()
my_plc.set_connection_type(3)
                                %This function is used to set connection attribute.
my_plc.connect('192.168.2.1', 0, 1)
```

```
                %This function is used to set the IP address, the rack number, and the slot number.
print(my_plc.get_connected())
                                %True is returned when the connection is done.
plc_data = my_plc.read_area(snap7.types.Areas.DB, 1, 0, 20)
%This function is used to define Digital output register, including the byte array, address
                number, the starting address of byte and the array size of byte.
pv2 = util.get_real(plc_data, 0)
u = util.get_real(plc_data, 4)
value2 = util.get_bool(IO_status, 0, 3)
  %This function is used to read the data of real and bool type from the relative register above.
util.get_real(pressure, 0)
util.get_bool(IO_status, 0, 3)
my_plc.disconnect()
my_plc.destroy()
print('read success')
return pv2, u
def write_plc_data(u):
my_plc = client.Client()
my_plc.set_connection_type(3)
my_plc.connect('192.168.2.1', 0, 1)
print(my_plc.get_connected())
plc_data = my_plc.read_area(snap7.types.Areas.DB, 1, 0, 20)
util.set_bool(IO_status, 0, 3, True)
util.set_real(plc_data, 12, 100.0)
util.set_real(plc_data, 12, u)
my_plc.write_area(snap7.types.Areas.PA, 0, 0, IO_status)
my_plc.write_area(snap7.types.Areas.DB, 1, 0, plc_data)
my_plc.disconnect()
my_plc.destroy()
print('write success')
return None
```

2.3 The traditional PID output algorithm in Python

```
import numpy as np
import time
from pid import pid
from PLC_Data import *                                  % Initialization of parameters.
du, u0, u1 = 0, 0, 0
SV, PV1, PV2 = 1.0, 0, 0
E_2, E_1 = 0, 0
T = 60
                                        % Define the initial value of process variable


while 1:
```

```
                                                    % Read the PLC data.
PV2, u1 = get_plc_data()
                                % Intercept the newest record respectively from IO address

# time.sleep(4.5)
E_0 = E_1
E_1 = E_2
E_2 = SV-PV2
xi = np.array([SV, PV2, E_2]).reshape(1, 3)
x = np.array([[E_2-E_1], [E_2*0.2], [(E_2-2*E_1+E_0)/2]])
ih = xi@wi_0.T
du = pid(PV2, SV, E_1, E_0, T, Ko[0,0], Ko[1,0], Ko[2,0])
u1 = u0 + du
if u1 > 100:
    u1 = 100
elif u1 < 0:
    u1 = 0
uo = u1
dyu = np.sign(((PV2-PV1)/(u1-u0+0.00000001)))
E_0 = E_1
E_1 = E_2
PV1 = PV2
write_plc_data(u1)
time.sleep(60)
```

2.4 The BPNN self-tuning PID output algorithm in Python

```
import numpy as np
import time
from pid import pid
from PLC_Data import *                                  % Initialization of parameters.
du, u0, u1 = 0, 0, 0
SV, PV1, PV2 = 1.0, 0, 0
E_2, E_1 = 0, 0
T = 60
                                                % Define the initial value of process variable
input, hid, out = 3, 5, 3
eita = 0.25
alfa = 0.9
  % Define the neuron number of input/hidden/output layer, the learning rate and the inertial
                                                                        coefficient in BPNN
wi_0 = np.random.rand(hid, input)
wi_1 = np.random.rand(hid, input)
                    % Generate the weighting coefficient between input layer and hidden layer
wo_0 = np.random.rand(out, hid)
```

```
wo_1 = np.random.rand(out, hid)
                % Generate the weighting coefficient between hidden layer and output layer
d_wi0 = np.zeros([hid, input])
d_wi1 = np.zeros([hid, input])

while 1:
                                                                % Read the PLC data.
    PV2, u1 = get_plc_data()
                                % Intercept the newest record respectively from IO address
    dKo = np.zeros([out, 1])
    dKh = np.zeros([hid, 1])
    delta_1 = np.zeros([out, 1])
    delta_j = np.zeros([hid, 1])

    # time.sleep(3.5)
    Kh = np.zeros([hid, 1])
    Ko = np.zeros([out, 1])



    E_0 = E_1
    E_1 = E_2
    E_2 = SV-PV2
    xi = np.array([SV, PV2, E_2]).reshape(1, 3)
    x = np.array([[E_2-E_1], [E_2*0.2], [(E_2-2*E_1+E_0)/2]])
    ih = xi@wi_0.T
    for j in range(hid):
        Kh[j, 0] = (np.exp(ih[0,j])-np.exp(-ih[0,j]))/(np.exp(ih[0,j])+np.exp(-ih[0,j]))
                                                        %Calculate the output of hidden layer
      ho = wo_0@Kh                          %The correction output value of hidden layer

    du = pid(PV2, SV, E_1, E_0, T, Ko[0,0], Ko[1,0], Ko[2,0])
    u1 = u0 + du
    if u1 > 100:
        u1 = 100
    elif u1 < 0:
        u1 = 0
    uo = u1
    dyu = np.sign(((PV2-PV1)/(u1-u0+0.00000001)))



    for i in range(out):
        for j in range(hid):
            d_wo1[i,j] = alfa*d_wo0[i,j]+eita*delta_l[i,0]*Kh[j,0]
    wo_1 = wo_0 + d_wo1
```

```
d_wo0 = d_wo1
segma = delta_l.T@wo_1

for j in range(hid):
        dKh[j,0] = 4/(np.exp(ih[0,j])+np.exp(-ih[0,j]))**2
        delta_j[j,0] = segma[0,j]*dKh[j,0]



wi_1 = wi_0 + d_wi1
d_wi0 = d_wi1
wo_0 = wo_1
wi_0 = wi_1
E_0 = E_1
E_1 = E_2
PV1 = PV2
write_plc_data(u1)
time.sleep(60)
```