**LAPORAN FINAL PROJECT**

**PENGUJIAN KIT ITCLAB MENGGUNAKAN PROGRAM ITCLAB**

****

**Dosen Pengampu :**

**Assoc. Prof. Dr. Basuki Rahmat, S.Si, MT, ITS-AI**

**Oleh :**

**Mohammad Faris Al Fatih (22081010277)**

**PROGRAM STUDI INFORMATIKA**

**FAKULTAS ILMU KOMPUTER**

**UNIVERSITAS PEMBANGUNAN NASIONAL JAWA TIMUR**

**2024**

# DAFTAR ISI

# PENDAHULUAN

Pengendalian suhu adalah salah satu aspek penting dalam berbagai sistem industri, ilmiah, dan teknologi, di mana pemeliharaan suhu yang stabil sangat diperlukan untuk mendukung kinerja perangkat atau sistem secara optimal. Dalam beberapa dekade terakhir, penerapan teknologi Internet of Things (IoT) telah memberikan kemajuan besar dalam bidang ini, memungkinkan pengendalian suhu secara lebih efisien dan terautomasi. Salah satu aplikasi yang dapat digunakan untuk memahami dan mengaplikasikan konsep-konsep pengendalian suhu adalah melalui eksperimen berbasis perangkat yang dapat dimodifikasi dan diprogram sesuai dengan kebutuhan.

Sebagai mahasiswa yang mempelajari teknologi dan sistem kontrol, saya diberikan kesempatan untuk melakukan percobaan menggunakan kode yang disusun oleh Assoc. Prof. Dr. Basuki Rahmat, S.Si, MT, ITS-AI. Kode ini bertujuan untuk mengimplementasikan pengendalian suhu berbasis IoT dengan menggunakan mikrokontroler ESP32. Melalui percobaan ini, saya dapat memahami lebih dalam tentang dinamika sistem pengendalian suhu, serta mengembangkan keterampilan dalam pemrograman dan analisis data yang berkaitan dengan sistem kontrol suhu.

Tujuan dari percobaan ini adalah untuk menguji kinerja sistem pengendalian suhu dengan perangkat iTCLab yang telah dikembangkan, yang melibatkan penggunaan sensor suhu, pemanas, dan mikrokontroler untuk memanipulasi dan memonitor suhu dalam batas yang telah ditentukan. Percobaan ini juga mencakup penerapan metode kontrol Proportional Integral Derivative (PID) untuk mengoptimalkan pengaturan suhu, yang merupakan salah satu metode kontrol yang banyak digunakan dalam berbagai aplikasi teknik.

Dengan melakukan percobaan ini, saya berharap dapat memperoleh pemahaman yang lebih baik mengenai pengendalian suhu, teknik kontrol, serta penerapan teknologi IoT dalam sistem yang lebih besar. Selain itu, eksperimen ini juga memberikan kesempatan untuk mengeksplorasi penggunaan machine learning dalam tuning kontrol PID, yang dapat menjadi dasar untuk riset dan pengembangan lebih lanjut di bidang ini.

# METODE

Untuk melakukan percobaan pengendalian suhu berbasis IoT menggunakan perangkat iTCLab, berikut adalah langkah-langkah yang dilakukan:

1. **Upload Program Arduino ke iTCLab Kit** Langkah pertama adalah meng-upload program arduino ke mikrokontroler ESP32 yang terdapat pada iTCLab Kit. Program ini bertanggung jawab untuk menghubungkan perangkat dengan lingkungan pemrograman Python dan memastikan komunikasi antara ESP32 dan komputer berjalan dengan baik. Program ini diunggah menggunakan Arduino IDE.
2. **Menyiapkan File Python** Setelah program Arduino berhasil di-upload, langkah berikutnya adalah menyiapkan file Python untuk melakukan pengendalian suhu. Tempatkan file .py di dalam folder yang sama dengan program python\_testing.ipynb. File itclab.py berfungsi sebagai penghubung antara Python dan perangkat iTCLab, sementara file python\_testing.ipynb adalah program yang akan dijalankan di Jupyter Notebook untuk menguji sistem pengendalian suhu.
3. **Menjalankan Program di Jupyter Notebook** Langkah terakhir adalah menjalankan program python .ipynb di Jupyter Notebook. Program ini akan berinteraksi dengan iTCLab Kit untuk menguji pengendalian suhu dan memonitor suhu yang dihasilkan oleh perangkat. Selama eksperimen, pengguna dapat memodifikasi parameter kontrol suhu dan memonitor hasilnya secara langsung melalui notebook.

Dengan mengikuti langkah-langkah di atas, sistem pengendalian suhu berbasis IoT dapat diuji dan dianalisis, memungkinkan pengguna untuk memahami dinamika pengendalian suhu serta mempelajari penerapan teknik kontrol seperti PID pada perangkat IoT.

# HASIL DAN PEMBAHASAN

## iTCLab 1

**Code iTCLab\_Testing.ino :**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Program : iTCLab\_Testing

\* By : Assoc. Prof. Dr. Basuki Rahmat, S.Si, MT, ITS-AI,

\* Assoc. Prof. Dr. Muljono, S.Si, M.Kom, et al

\* Pro. Team : i-ot.net, io-t.net

\* R. Group : Intelligent Control, Robotics **and** Automation Systems Research Group

\* Univ. : Universitas Pembangunan Nasional "Veteran" Jawa Timur

\* Country : Indonesia

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <Arduino.h>

// constants

const int baud = **115200**; // serial baud rate

// pin numbers corresponding to signals on the iTCLab Shield

const int pinT1 = **34**; // T1

const int pinT2 = **35**; // T2

const int pinQ1 = **32**; // Q1

const int pinQ2 = **33**; // Q2

const int pinLED = **26**; // LED

// setting PWM properties

const int freq = **5000**; //**5000**

const int ledChannel = **0**;

const int Q1Channel = **1**;

const int Q2Channel = **2**;

const int resolutionLedChannel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ1Channel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ2Channel = **8**; //Resolution **8**, **10**, **12**, **15**

float cel, cel1, degC, degC1;

const float upper\_temperature\_limit = **55**;

// **global** variables

float Q1 = **0**; // value written to Q1 pin

float Q2 = **0**; // value written to Q2 pin

int iwrite\_max = **255**; // integer value **for** writing

int iwrite\_min = **0**; // integer value **for** writing

void setup() {

// put your setup code here, to run once:

Serial.begin(baud);

**while** (!Serial) {

; // wait **for** serial port to connect.

}

// configure pinQ1 PWM functionalitites

ledcSetup(Q1Channel, freq, resolutionQ1Channel);

// attach the channel to the pinQ1 to be controlled

ledcAttachPin(pinQ1, Q1Channel);

// configure pinQ2 PWM functionalitites

ledcSetup(Q2Channel, freq, resolutionQ2Channel);

// attach the channel to the pinQ2 to be controlled

ledcAttachPin(pinQ2, Q2Channel);

// configure pinLED PWM functionalitites

ledcSetup(ledChannel, freq, resolutionLedChannel);

// attach the channel to the pinLED to be controlled

ledcAttachPin(pinLED, ledChannel);

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

ledcWrite(ledChannel,**0**);

}

void Q1on(){

ledcWrite(Q1Channel,iwrite\_max/**255**\***100**);

//Q1 = iwrite\_max/**255**\***100**;

//Serial.println(Q1);

}

void Q1off(){

ledcWrite(Q1Channel,iwrite\_min/**255**\***100**);

//Q1 = iwrite\_min/**255**\***100**;

//Serial.println(Q1);

}

void Q2on(){

ledcWrite(Q2Channel,iwrite\_max/**255**\***100**);

//Q2 = iwrite\_max/**255**\***100**;

//Serial.println(Q2);

}

void Q2off(){

ledcWrite(Q2Channel,iwrite\_min/**255**\***100**);

//Q2 = iwrite\_min/**255**\***100**;

//Serial.println(Q2);

}

void ledon(){

ledcWrite(ledChannel,iwrite\_max);

}

void ledoff(){

ledcWrite(ledChannel,iwrite\_min);

}

void cektemp(){

degC = analogRead(pinT1) \* **0.322265625** ; // use **for** **3.3**v AREF

cel = degC/**10**;

degC1 = analogRead(pinT2) \* **0.322265625** ; // use **for** **3.3**v AREF

cel1 = degC1/**10**;

Serial.**print**("Temperature: ");

Serial.**print**(cel); // **print** the temperature T1 **in** Celsius

Serial.**print**("°C");

Serial.**print**(" ~ "); // separator between Celsius **and** Fahrenheit

Serial.**print**(cel1); // **print** the temperature T2 **in** Celsius

Serial.println("°C");

}

void loop() {

// put your main code here, to run repeatedly:

cektemp();

**if** (cel > upper\_temperature\_limit){

Q1off();

ledon();

}

**else** {

Q1on();

ledoff();

}

**if** (cel1 > upper\_temperature\_limit){

Q2off();

ledon();

}

**else** {

Q2on();

ledoff();

}

delay (**100**);

}

**Output :**

Sebuah gambar berisi teks, cuplikan layar, Font, software

Deskripsi dibuat secara otomatis

## iTCLab 2

**Code PWM\_testing.ino**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Program : PWM\_Testing

\* By : Assoc. Prof. Dr. Basuki Rahmat, S.Si, MT, ITS-AI,

\* Assoc. Prof. Dr. Muljono, S.Si, M.Kom, et al

\* Pro. Team : i-ot.net, io-t.net

\* R. Group : Intelligent Control, Robotics **and** Automation Systems Research Group

\* Univ. : Universitas Pembangunan Nasional "Veteran" Jawa Timur

\* Country : Indonesia

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// the number of the LED pin

const int ledPin = **26**;

// setting PWM properties

const int freq = **5000**;

const int ledChannel = **0**;

const int resolution = **8**;

void setup(){

// configure LED PWM functionalitites

ledcSetup(ledChannel, freq, resolution);

// attach the channel to the GPIO to be controlled

ledcAttachPin(ledPin, ledChannel);

}

void loop(){

// increase the LED brightness

**for**(int dutyCycle = **0**; dutyCycle <= **255**; dutyCycle++){

// changing the LED brightness **with** PWM

ledcWrite(ledChannel, dutyCycle);

delay(**20**);

}

// decrease the LED brightness

**for**(int dutyCycle = **255**; dutyCycle >= **0**; dutyCycle--){

// changing the LED brightness **with** PWM

ledcWrite(ledChannel, dutyCycle);

delay(**20**);

}

}

**Output :**

**Sebuah gambar berisi elektronik, teks, cuplikan layar, komputer

Deskripsi dibuat secara otomatisSebuah gambar berisi komputer, cuplikan layar, teks, computer

Deskripsi dibuat secara otomatis**

## iTCLab 3

**Code arduino\_python.ino**

/\*

iTCLab Internet-Based Temperature Control Lab Firmware

Jeffrey Kantor, Initial Version

John Hedengren, Modified

Oct **2017**

Basuki Rahmat, Modified

April **2022**

This firmware **is** loaded into the Internet-Based Temperature Control Laboratory ESP32 to

provide a high level interface to the Internet-Based Temperature Control Lab. The firmware

scans the serial port looking **for** case-insensitive commands:

Q1 set Heater **1**, range **0** to **100**% subject to limit (**0**-**255** int)

Q2 set Heater **2**, range **0** to **100**% subject to limit (**0**-**255** int)

T1 get Temperature T1, returns deg C **as** string

T2 get Temperature T2, returns dec C **as** string

VER get firmware version string

X stop, enter sleep mode

Limits on the heater can be configured **with** the constants below.

\*/

#include <Arduino.h>

// constants

const String vers = "1.04"; // version of this firmware

const int baud = **115200**; // serial baud rate

const char sp = ' '; // command separator

const char nl = '**\n**'; // command terminator

// pin numbers corresponding to signals on the iTCLab Shield

const int pinT1 = **34**; // T1

const int pinT2 = **35**; // T2

const int pinQ1 = **32**; // Q1

const int pinQ2 = **33**; // Q2

const int pinLED = **26**; // LED

//Q1 **32** - T1 **34**

//Q2 **33** - T2 **35**

// setting PWM properties

const int freq = **5000**; //**5000**

const int ledChannel = **0**;

const int Q1Channel = **1**;

const int Q2Channel = **2**;

const int resolutionLedChannel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ1Channel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ2Channel = **8**; //Resolution **8**, **10**, **12**, **15**

const double upper\_temperature\_limit = **59**;

// **global** variables

char Buffer[**64**]; // buffer **for** parsing serial input

String cmd; // command

double pv = **0**; // pin value

float level; // LED level (**0**-**100**%)

double Q1 = **0**; // value written to Q1 pin

double Q2 = **0**; // value written to Q2 pin

int iwrite = **0**; // integer value **for** writing

float dwrite = **0**; // float value **for** writing

int n = **10**; // number of samples **for** each temperature measurement

void parseSerial(void) {

int ByteCount = Serial.readBytesUntil(nl,Buffer,sizeof(Buffer));

String read\_ = String(Buffer);

memset(Buffer,**0**,sizeof(Buffer));

// separate command **from** **associated** **data**

int idx = read\_.indexOf(sp);

cmd = read\_.substring(**0**,idx);

cmd.trim();

cmd.toUpperCase();

// extract data. toInt() returns **0** on error

String data = read\_.substring(idx+**1**);

data.trim();

pv = data.toFloat();

}

// Q1\_max = **100**%

// Q2\_max = **100**%

void dispatchCommand(void) {

**if** (cmd == "Q1") {

Q1 = max(**0.0**, min(**25.0**, pv));

iwrite = int(Q1 \* **2.0**); // **10.**? max

iwrite = max(**0**, min(**255**, iwrite));

ledcWrite(Q1Channel,iwrite);

Serial.println(Q1);

}

**else** **if** (cmd == "Q2") {

Q2 = max(**0.0**, min(**25.0**, pv));

iwrite = int(Q2 \* **2.0**); // **10.**? max

iwrite = max(**0**, min(**255**, iwrite));

ledcWrite(Q2Channel,iwrite);

Serial.println(Q2);

}

**else** **if** (cmd == "T1") {

float mV = **0.0**;

float degC = **0.0**;

**for** (int i = **0**; i < n; i++) {

mV = (float) analogRead(pinT1) \* **0.322265625**;

degC = degC + mV/**10.0**;

}

degC = degC / float(n);

Serial.println(degC);

}

**else** **if** (cmd == "T2") {

float mV = **0.0**;

float degC = **0.0**;

**for** (int i = **0**; i < n; i++) {

mV = (float) analogRead(pinT2) \* **0.322265625**;

degC = degC + mV/**10.0**;

}

degC = degC / float(n);

Serial.println(degC);

}

**else** **if** ((cmd == "V") **or** (cmd == "VER")) {

Serial.println("TCLab Firmware Version " + vers);

}

**else** **if** (cmd == "LED") {

level = max(**0.0**, min(**100.0**, pv));

iwrite = int(level \* **0.5**);

iwrite = max(**0**, min(**50**, iwrite));

ledcWrite(ledChannel, iwrite);

Serial.println(level);

}

**else** **if** (cmd == "X") {

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

Serial.println("Stop");

}

}

// check temperature **and** shut-off heaters **if** above high limit

void checkTemp(void) {

float mV = (float) analogRead(pinT1) \* **0.322265625**;

//float degC = (mV - **500.0**)/**10.0**;

float degC = mV/**10.0**;

**if** (degC >= upper\_temperature\_limit) {

Q1 = **0.0**;

Q2 = **0.0**;

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

//Serial.println("High Temp 1 (> upper\_temperature\_limit): ");

Serial.println(degC);

}

mV = (float) analogRead(pinT2) \* **0.322265625**;

//degC = (mV - **500.0**)/**10.0**;

degC = mV/**10.0**;

**if** (degC >= upper\_temperature\_limit) {

Q1 = **0.0**;

Q2 = **0.0**;

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

//Serial.println("High Temp 2 (> upper\_temperature\_limit): ");

Serial.println(degC);

}

}

// arduino startup

void setup() {

//analogReference(EXTERNAL);

Serial.begin(baud);

**while** (!Serial) {

; // wait **for** serial port to connect.

}

// configure pinQ1 PWM functionalitites

ledcSetup(Q1Channel, freq, resolutionQ1Channel);

// attach the channel to the pinQ1 to be controlled

ledcAttachPin(pinQ1, Q1Channel);

// configure pinQ2 PWM functionalitites

ledcSetup(Q2Channel, freq, resolutionQ2Channel);

// attach the channel to the pinQ2 to be controlled

ledcAttachPin(pinQ2, Q2Channel);

// configure pinLED PWM functionalitites

ledcSetup(ledChannel, freq, resolutionLedChannel);

// attach the channel to the pinLED to be controlled

ledcAttachPin(pinLED, ledChannel);

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

}

// arduino main event loop

void loop() {

parseSerial();

dispatchCommand();

checkTemp();

}

**Code itclab.py**

**import** **sys**

**import** **time**

**import** **numpy** **as** **np**

**try**:

**import** **serial**

**except**:

**import** **pip**

pip.main(['install','pyserial'])

**import** **serial**

**from** **serial.tools** **import** list\_ports

**class** **iTCLab**(object):

**def** **\_\_init\_\_**(self, port=None, baud=**115200**):

port = self.findPort()

**print**('Opening connection')

self.sp = serial.Serial(port=port, baudrate=baud, timeout=**2**)

self.sp.flushInput()

self.sp.flushOutput()

time.sleep(**3**)

**print**('iTCLab connected via Arduino on port ' + port)

**def** **findPort**(self):

found = False

**for** port **in** list(list\_ports.comports()):

# Arduino Uno

**if** port[**2**].startswith('USB VID:PID=16D0:0613'):

port = port[**0**]

found = True

# Arduino HDuino

**if** port[**2**].startswith('USB VID:PID=1A86:7523'):

port = port[**0**]

found = True

# Arduino Leonardo

**if** port[**2**].startswith('USB VID:PID=2341:8036'):

port = port[**0**]

found = True

# Arduino ESP32

**if** port[**2**].startswith('USB VID:PID=10C4:EA60'):

port = port[**0**]

found = True

# Arduino ESP32 - Tipe yg berbeda

**if** port[**2**].startswith('USB VID:PID=1A86:55D4'):

port = port[**0**]

found = True

**if** (**not** found):

**print**('Arduino COM port not found')

**print**('Please ensure that the USB cable is connected')

**print**('--- Printing Serial Ports ---')

**for** port **in** list(serial.tools.list\_ports.comports()):

**print**(port[**0**] + ' ' + port[**1**] + ' ' + port[**2**])

**print**('For Windows:')

**print**(' Open device manager, select "Ports (COM & LPT)"')

**print**(' Look for COM port of Arduino such as COM4')

**print**('For MacOS:')

**print**(' Open terminal and type: ls /dev/\*.')

**print**(' Search for /dev/tty.usbmodem\* or /dev/tty.usbserial\*. The port number is \*.')

**print**('For Linux')

**print**(' Open terminal and type: ls /dev/tty\*')

**print**(' Search for /dev/ttyUSB\* or /dev/ttyACM\*. The port number is \*.')

**print**('')

port = input('Input port: ')

# or hard-code it here

#port = 'COM3' # for Windows

#port = '/dev/tty.wchusbserial1410' # for MacOS

**return** port

**def** **stop**(self):

**return** self.read('X')

**def** **version**(self):

**return** self.read('VER')

**@property**

**def** **T1**(self):

self.\_T1 = float(self.read('T1'))

**return** self.\_T1

**@property**

**def** **T2**(self):

self.\_T2 = float(self.read('T2'))

**return** self.\_T2

**def** **LED**(self,pwm):

pwm = max(**0.0**,min(**100.0**,pwm))/**2.0**

self.write('LED',pwm)

**return** pwm

**def** **Q1**(self,pwm):

pwm = max(**0.0**,min(**100.0**,pwm))

self.write('Q1',pwm)

**return** pwm

**def** **Q2**(self,pwm):

pwm = max(**0.0**,min(**100.0**,pwm))

self.write('Q2',pwm)

**return** pwm

# save txt file with data and set point

# t = time

# u1,u2 = heaters

# y1,y2 = tempeatures

# sp1,sp2 = setpoints

**def** **save\_txt**(self,t,u1,u2,y1,y2,sp1,sp2):

data = np.vstack((t,u1,u2,y1,y2,sp1,sp2)) # vertical stack

data = data.T # transpose data

top = 'Time (sec), Heater 1 (%), Heater 2 (%), ' \

+ 'Temperature 1 (degC), Temperature 2 (degC), ' \

+ 'Set Point 1 (degC), Set Point 2 (degC)'

np.savetxt('data.txt',data,delimiter=',',header=top,comments='')

**def** **read**(self,cmd):

cmd\_str = self.build\_cmd\_str(cmd,'')

**try**:

self.sp.write(cmd\_str.encode())

self.sp.flush()

**except** **Exception**:

**return** None

**return** self.sp.readline().decode('UTF-8').replace("**\r\n**", "")

**def** **write**(self,cmd,pwm):

cmd\_str = self.build\_cmd\_str(cmd,(pwm,))

**try**:

self.sp.write(cmd\_str.encode())

self.sp.flush()

**except**:

**return** None

**return** self.sp.readline().decode('UTF-8').replace("**\r\n**", "")

**def** **build\_cmd\_str**(self,cmd, args=None):

"""

Build a command string that can be sent to the arduino.

Input:

cmd (str): the command to send to the arduino, must not

contain a % character

args (iterable): the arguments to send to the command

"""

**if** args:

args = ' '.join(map(str, args))

**else**:

args = ''

**return** "{cmd} {args}**\n**".format(cmd=cmd, args=args)

**def** **close**(self):

**try**:

self.sp.close()

**print**('Arduino disconnected successfully')

**except**:

**print**('Problems disconnecting from Arduino.')

**print**('Please unplug and reconnect Arduino.')

**return** True

**Code python\_testing.ipynb**

**import** **itclab**

**import** **time**

# Connect to Arduino

a = itclab.iTCLab()

**print**('LED On')

a.LED(**100**)

# Pause for 1 second

time.sleep(**1.0**)

**print**('LED Off')

a.LED(**0**)

a.close()

**import** **itclab**

**import** **time**

# Connect to Arduino

a = itclab.iTCLab()

# Get Version

**print**(a.version)

# Turn LED on

**print**('LED On')

a.LED(**100**)

# Taper LED off

**for** i **in** range(**100**,-**1**,-**10**):

**print**('LED Power ' + str(i))

time.sleep(**0.5**)

a.LED(i)

a.close()

a.close()

## iTCLab 4

Code **pid\_simulation.ipynb**

**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

n = **100** # time points to plot

tf = **20.0** # 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=**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)

## iTCLab 5

**Code PID\_Arduino.ino**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Program : PID-iTCLab Programming Using Arduino

\* By : Assoc. Prof. Dr. Basuki Rahmat, S.Si, MT, ITS-AI,

\* Assoc. Prof. Dr. Muljono, S.Si, M.Kom, et al

\* Pro. Team : i-ot.net, io-t.net

\* R. Group : Intelligent Control, Robotics **and** Automation Systems Research Group

\* Univ. : Universitas Pembangunan Nasional "Veteran" Jawa Timur

\* Country : Indonesia

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <Arduino.h>

// constants

const int baud = **115200**; // serial baud rate

// pin numbers corresponding to signals on the iTCLab Shield

const int pinT1 = **34**; // T1

const int pinT2 = **35**; // T2

const int pinQ1 = **32**; // Q1

const int pinQ2 = **33**; // Q2

const int pinLED = **26**; // LED

// setting PWM properties

const int freq = **5000**; //**5000**

const int ledChannel = **0**;

const int Q1Channel = **1**;

const int Q2Channel = **2**;

const int resolutionLedChannel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ1Channel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ2Channel = **8**; //Resolution **8**, **10**, **12**, **15**

float cel, cel1, degC, degC1;

float P, I, D, Kc, tauI, tauD;

float KP, KI, KD, op0, ophi, oplo, error, dpv;

float sp = **35**, //set point

pv = **0**, //current temperature

pv\_last = **0**, //prior temperature

ierr = **0**, //integral error

dt = **0**, //time between measurements

op = **0**; //PID controller output

unsigned long ts = **0**, new\_ts = **0**; //timestamp

const float upper\_temperature\_limit = **58**;

// **global** variables

float Q1 = **0**; // value written to Q1 pin

float Q2 = **0**; // value written to Q2 pin

int iwrite\_value = **25**; // integer value **for** writing

int iwrite\_led = **255**; // integer value **for** writing

int iwrite\_min = **0**; // integer value **for** writing

void setup() {

// put your setup code here, to run once:

ts = millis();

Serial.begin(baud);

**while** (!Serial) {

; // wait **for** serial port to connect.

}

// configure pinQ1 PWM functionalitites

ledcSetup(Q1Channel, freq, resolutionQ1Channel);

// attach the channel to the pinQ1 to be controlled

ledcAttachPin(pinQ1, Q1Channel);

// configure pinQ2 PWM functionalitites

ledcSetup(Q2Channel, freq, resolutionQ2Channel);

// attach the channel to the pinQ2 to be controlled

ledcAttachPin(pinQ2, Q2Channel);

// configure pinLED PWM functionalitites

ledcSetup(ledChannel, freq, resolutionLedChannel);

// attach the channel to the pinLED to be controlled

ledcAttachPin(pinLED, ledChannel);

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

ledcWrite(ledChannel,**0**);

}

void Q1on(){

ledcWrite(Q1Channel,iwrite\_value);

//Serial.println(Q1);

}

void Q1off(){

ledcWrite(Q1Channel,iwrite\_min);

//Serial.println(Q1);

}

void Q2on(){

ledcWrite(Q2Channel,iwrite\_value);

//Serial.println(Q2);

}

void Q2off(){

ledcWrite(Q2Channel,iwrite\_min);

//Serial.println(Q2);

}

void ledon(){

ledcWrite(ledChannel,iwrite\_led);

}

void ledoff(){

ledcWrite(ledChannel,iwrite\_min);

}

void cektemp(){

degC = analogRead(pinT1) \* **0.322265625** ; // use **for** **3.3**v AREF

cel = degC/**10**;

degC1 = analogRead(pinT2) \* **0.322265625** ; // use **for** **3.3**v AREF

cel1 = degC1/**10**;

Serial.**print**("Temperature T1: ");

Serial.**print**(cel); // **print** the temperature T1 **in** Celsius

Serial.**print**("°C");

Serial.**print**(" ~ "); // separator between Celsius **and** Fahrenheit

Serial.**print**("Temperature T2: ");

Serial.**print**(cel1); // **print** the temperature T2 **in** Celsius

Serial.println("°C");

}

float pid(float sp, float pv, float pv\_last, float& ierr, float dt) {

float Kc = **10.0**; // K / %Heater

float tauI = **50.0**; // sec

float tauD = **1.0**; // sec

// PID coefficients

float KP = Kc;

float KI = Kc / tauI;

float KD = Kc\*tauD;

// upper **and** lower bounds on heater level

float ophi = **100**;

float oplo = **0**;

// calculate the error

float error = sp - pv;

// calculate the integral error

ierr = ierr + KI \* error \* dt;

// calculate the measurement derivative

float dpv = (pv - pv\_last) / dt;

// calculate the PID output

float P = KP \* error; //proportional contribution

float I = ierr; //integral contribution

float D = -KD \* dpv; //derivative contribution

float op = P + I + D;

// implement anti-reset windup

**if** ((op < oplo) || (op > ophi)) {

I = I - KI \* error \* dt;

// clip output

op = max(oplo, min(ophi, op));

}

ierr = I;

Serial.println("sp="+String(sp) + " pv=" + String(pv) + " dt=" + String(dt) + " op=" + String(op) + " P=" + String(P) + " I=" + String(I) + " D=" + String(D));

**return** op;

}

void loop() {

new\_ts = millis();

**if** (new\_ts - ts > **1000**) {

// put your main code here, to run repeatedly:

cektemp();

**if** (cel > upper\_temperature\_limit){

Q1off();

ledon();

}

**else** {

Q1on();

ledoff();

}

**if** (cel1 > upper\_temperature\_limit){

Q2off();

ledon();

}

**else** {

Q2on();

ledoff();

}

pv = cel; // Temperature T1

dt = (new\_ts - ts) / **1000.0**;

ts = new\_ts;

op = pid(sp,pv,pv\_last,ierr,dt);

ledcWrite(Q1Channel,op);

pv\_last = pv;

delay (**200**);

}

}

## iTCLab 6

**Code PID\_Python.ino**

/\*

iTCLab Internet-Based Temperature Control Lab Firmware

Jeffrey Kantor, Initial Version

John Hedengren, Modified

Oct **2017**

Basuki Rahmat, Modified

April **2022**

This firmware **is** loaded into the Internet-Based Temperature Control Laboratory ESP32 to

provide a high level interface to the Internet-Based Temperature Control Lab. The firmware

scans the serial port looking **for** case-insensitive commands:

Q1 set Heater **1**, range **0** to **100**% subject to limit (**0**-**255** int)

Q2 set Heater **2**, range **0** to **100**% subject to limit (**0**-**255** int)

T1 get Temperature T1, returns deg C **as** string

T2 get Temperature T2, returns dec C **as** string

VER get firmware version string

X stop, enter sleep mode

Limits on the heater can be configured **with** the constants below.

\*/

#include <Arduino.h>

// constants

const String vers = "1.04"; // version of this firmware

const int baud = **115200**; // serial baud rate

const char sp = ' '; // command separator

const char nl = '**\n**'; // command terminator

// pin numbers corresponding to signals on the iTCLab Shield

const int pinT1 = **34**; // T1

const int pinT2 = **35**; // T2

const int pinQ1 = **32**; // Q1

const int pinQ2 = **33**; // Q2

const int pinLED = **26**; // LED

// setting PWM properties

const int freq = **5000**; //**5000**

const int ledChannel = **0**;

const int Q1Channel = **1**;

const int Q2Channel = **2**;

const int resolutionLedChannel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ1Channel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ2Channel = **8**; //Resolution **8**, **10**, **12**, **15**

const double upper\_temperature\_limit = **59**;

// **global** variables

char Buffer[**64**]; // buffer **for** parsing serial input

String cmd; // command

double pv = **0**; // pin value

float level; // LED level (**0**-**100**%)

double Q1 = **0**; // value written to Q1 pin

double Q2 = **0**; // value written to Q2 pin

int iwrite = **0**; // integer value **for** writing

float dwrite = **0**; // float value **for** writing

int n = **10**; // number of samples **for** each temperature measurement

void parseSerial(void) {

int ByteCount = Serial.readBytesUntil(nl,Buffer,sizeof(Buffer));

String read\_ = String(Buffer);

memset(Buffer,**0**,sizeof(Buffer));

// separate command **from** **associated** **data**

int idx = read\_.indexOf(sp);

cmd = read\_.substring(**0**,idx);

cmd.trim();

cmd.toUpperCase();

// extract data. toInt() returns **0** on error

String data = read\_.substring(idx+**1**);

data.trim();

pv = data.toFloat();

}

// Q1\_max = **100**%

// Q2\_max = **100**%

void dispatchCommand(void) {

**if** (cmd == "Q1") {

Q1 = max(**0.0**, min(**25.0**, pv));

iwrite = int(Q1 \* **2.0**); // **10.**? max

iwrite = max(**0**, min(**255**, iwrite));

ledcWrite(Q1Channel,iwrite);

Serial.println(Q1);

}

**else** **if** (cmd == "Q2") {

Q2 = max(**0.0**, min(**25.0**, pv));

iwrite = int(Q2 \* **2.0**); // **10.**? max

iwrite = max(**0**, min(**255**, iwrite));

ledcWrite(Q2Channel,iwrite);

Serial.println(Q2);

}

**else** **if** (cmd == "T1") {

float mV = **0.0**;

float degC = **0.0**;

**for** (int i = **0**; i < n; i++) {

mV = (float) analogRead(pinT1) \* **0.322265625**;

degC = degC + mV/**10.0**;

}

degC = degC / float(n);

Serial.println(degC);

}

**else** **if** (cmd == "T2") {

float mV = **0.0**;

float degC = **0.0**;

**for** (int i = **0**; i < n; i++) {

mV = (float) analogRead(pinT2) \* **0.322265625**;

degC = degC + mV/**10.0**;

}

degC = degC / float(n);

Serial.println(degC);

}

**else** **if** ((cmd == "V") **or** (cmd == "VER")) {

Serial.println("TCLab Firmware Version " + vers);

}

**else** **if** (cmd == "LED") {

level = max(**0.0**, min(**100.0**, pv));

iwrite = int(level \* **0.5**);

iwrite = max(**0**, min(**50**, iwrite));

ledcWrite(ledChannel, iwrite);

Serial.println(level);

}

**else** **if** (cmd == "X") {

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

Serial.println("Stop");

}

}

// check temperature **and** shut-off heaters **if** above high limit

void checkTemp(void) {

float mV = (float) analogRead(pinT1) \* **0.322265625**;

//float degC = (mV - **500.0**)/**10.0**;

float degC = mV/**10.0**;

**if** (degC >= upper\_temperature\_limit) {

Q1 = **0.0**;

Q2 = **0.0**;

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

//Serial.println("High Temp 1 (> upper\_temperature\_limit): ");

Serial.println(degC);

}

mV = (float) analogRead(pinT2) \* **0.322265625**;

//degC = (mV - **500.0**)/**10.0**;

degC = mV/**10.0**;

**if** (degC >= upper\_temperature\_limit) {

Q1 = **0.0**;

Q2 = **0.0**;

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

//Serial.println("High Temp 2 (> upper\_temperature\_limit): ");

Serial.println(degC);

}

}

// arduino startup

void setup() {

//analogReference(EXTERNAL);

Serial.begin(baud);

**while** (!Serial) {

; // wait **for** serial port to connect.

}

// configure pinQ1 PWM functionalitites

ledcSetup(Q1Channel, freq, resolutionQ1Channel);

// attach the channel to the pinQ1 to be controlled

ledcAttachPin(pinQ1, Q1Channel);

// configure pinQ2 PWM functionalitites

ledcSetup(Q2Channel, freq, resolutionQ2Channel);

// attach the channel to the pinQ2 to be controlled

ledcAttachPin(pinQ2, Q2Channel);

// configure pinLED PWM functionalitites

ledcSetup(ledChannel, freq, resolutionLedChannel);

// attach the channel to the pinLED to be controlled

ledcAttachPin(pinLED, ledChannel);

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

}

// arduino main event loop

void loop() {

parseSerial();

dispatchCommand();

checkTemp();

}

**Code itclab.py**

**import** **sys**

**import** **time**

**import** **numpy** **as** **np**

**try**:

**import** **serial**

**except**:

**import** **pip**

pip.main(['install','pyserial'])

**import** **serial**

**from** **serial.tools** **import** list\_ports

**class** **iTCLab**(object):

**def** **\_\_init\_\_**(self, port=None, baud=**115200**):

port = self.findPort()

**print**('Opening connection')

self.sp = serial.Serial(port=port, baudrate=baud, timeout=**2**)

self.sp.flushInput()

self.sp.flushOutput()

time.sleep(**3**)

**print**('iTCLab connected via Arduino on port ' + port)

**def** **findPort**(self):

found = False

**for** port **in** list(list\_ports.comports()):

# Arduino Uno

**if** port[**2**].startswith('USB VID:PID=16D0:0613'):

port = port[**0**]

found = True

# Arduino HDuino

**if** port[**2**].startswith('USB VID:PID=1A86:7523'):

port = port[**0**]

found = True

# Arduino Leonardo

**if** port[**2**].startswith('USB VID:PID=2341:8036'):

port = port[**0**]

found = True

# Arduino ESP32

**if** port[**2**].startswith('USB VID:PID=10C4:EA60'):

port = port[**0**]

found = True

# Arduino ESP32 - Tipe yg berbeda

**if** port[**2**].startswith('USB VID:PID=1A86:55D4'):

port = port[**0**]

found = True

**if** (**not** found):

**print**('Arduino COM port not found')

**print**('Please ensure that the USB cable is connected')

**print**('--- Printing Serial Ports ---')

**for** port **in** list(serial.tools.list\_ports.comports()):

**print**(port[**0**] + ' ' + port[**1**] + ' ' + port[**2**])

**print**('For Windows:')

**print**(' Open device manager, select "Ports (COM & LPT)"')

**print**(' Look for COM port of Arduino such as COM4')

**print**('For MacOS:')

**print**(' Open terminal and type: ls /dev/\*.')

**print**(' Search for /dev/tty.usbmodem\* or /dev/tty.usbserial\*. The port number is \*.')

**print**('For Linux')

**print**(' Open terminal and type: ls /dev/tty\*')

**print**(' Search for /dev/ttyUSB\* or /dev/ttyACM\*. The port number is \*.')

**print**('')

port = input('Input port: ')

# or hard-code it here

#port = 'COM3' # for Windows

#port = '/dev/tty.wchusbserial1410' # for MacOS

**return** port

**def** **stop**(self):

**return** self.read('X')

**def** **version**(self):

**return** self.read('VER')

**@property**

**def** **T1**(self):

self.\_T1 = float(self.read('T1'))

**return** self.\_T1

**@property**

**def** **T2**(self):

self.\_T2 = float(self.read('T2'))

**return** self.\_T2

**def** **LED**(self,pwm):

pwm = max(**0.0**,min(**100.0**,pwm))/**2.0**

self.write('LED',pwm)

**return** pwm

**def** **Q1**(self,pwm):

pwm = max(**0.0**,min(**100.0**,pwm))

self.write('Q1',pwm)

**return** pwm

**def** **Q2**(self,pwm):

pwm = max(**0.0**,min(**100.0**,pwm))

self.write('Q2',pwm)

**return** pwm

# save txt file with data and set point

# t = time

# u1,u2 = heaters

# y1,y2 = tempeatures

# sp1,sp2 = setpoints

**def** **save\_txt**(self,t,u1,u2,y1,y2,sp1,sp2):

data = np.vstack((t,u1,u2,y1,y2,sp1,sp2)) # vertical stack

data = data.T # transpose data

top = 'Time (sec), Heater 1 (%), Heater 2 (%), ' \

+ 'Temperature 1 (degC), Temperature 2 (degC), ' \

+ 'Set Point 1 (degC), Set Point 2 (degC)'

np.savetxt('data.txt',data,delimiter=',',header=top,comments='')

**def** **read**(self,cmd):

cmd\_str = self.build\_cmd\_str(cmd,'')

**try**:

self.sp.write(cmd\_str.encode())

self.sp.flush()

**except** **Exception**:

**return** None

**return** self.sp.readline().decode('UTF-8').replace("**\r\n**", "")

**def** **write**(self,cmd,pwm):

cmd\_str = self.build\_cmd\_str(cmd,(pwm,))

**try**:

self.sp.write(cmd\_str.encode())

self.sp.flush()

**except**:

**return** None

**return** self.sp.readline().decode('UTF-8').replace("**\r\n**", "")

**def** **build\_cmd\_str**(self,cmd, args=None):

"""

Build a command string that can be sent to the arduino.

Input:

cmd (str): the command to send to the arduino, must not

contain a % character

args (iterable): the arguments to send to the command

"""

**if** args:

args = ' '.join(map(str, args))

**else**:

args = ''

**return** "{cmd} {args}**\n**".format(cmd=cmd, args=args)

**def** **close**(self):

**try**:

self.sp.close()

**print**('Arduino disconnected successfully')

**except**:

**print**('Problems disconnecting from Arduino.')

**print**('Please unplug and reconnect Arduino.')

**return** True

**Code PID\_Python.ipynb**

**import** **itclab**

**import** **numpy** **as** **np**

**import** **time**

**import** **matplotlib.pyplot** **as** **plt**

**from** **scipy.integrate** **import** odeint

######################################################

# Use this script for evaluating model predictions #

# and PID controller performance for the TCLab #

# Adjust only PID and model sections #

######################################################

######################################################

# PID Controller #

######################################################

# inputs -----------------------------------

# sp = setpoint

# pv = current temperature

# pv\_last = prior temperature

# ierr = integral error

# dt = time increment between measurements

# outputs ----------------------------------

# op = output of the PID controller

# P = proportional contribution

# I = integral contribution

# D = derivative contribution

**def** **pid**(sp,pv,pv\_last,ierr,dt):

Kc = **10.0** # K/%Heater

tauI = **50.0** # sec

tauD = **1.0** # sec

# Parameters in terms of PID coefficients

KP = Kc

KI = Kc/tauI

KD = Kc\*tauD

# ubias for controller (initial heater)

op0 = **0**

# upper and lower bounds on heater level

ophi = **100**

oplo = **0**

# calculate the error

error = sp-pv

# calculate the integral error

ierr = ierr + KI \* error \* dt

# calculate the measurement derivative

dpv = (pv - pv\_last) / dt

# calculate the PID output

P = KP \* error

I = ierr

D = -KD \* dpv

op = op0 + P + I + D

# implement anti-reset windup

**if** op < oplo **or** op > ophi:

I = I - KI \* error \* dt

# clip output

op = max(oplo,min(ophi,op))

# return the controller output and PID terms

**return** [op,P,I,D]

######################################################

# FOPDT model #

######################################################

Kp = **0.5** # degC/%

tauP = **120.0** # seconds

thetaP = **10** # seconds (integer)

Tss = **23** # degC (ambient temperature)

Qss = **0** # % heater

######################################################

# Energy balance model #

######################################################

**def** **heat**(x,t,Q):

# Parameters

Ta = **23** + **273.15** # K

U = **10.0** # W/m^2-K

m = **4.0**/**1000.0** # kg

Cp = **0.5** \* **1000.0** # J/kg-K

A = **12.0** / **100.0**\*\***2** # Area in m^2

alpha = **0.01** # W / % heater

eps = **0.9** # Emissivity

sigma = **5.67e-8** # Stefan-Boltzman

# Temperature State

T = x[**0**]

# Nonlinear Energy Balance

dTdt = (**1.0**/(m\*Cp))\*(U\*A\*(Ta-T) \

+ eps \* sigma \* A \* (Ta\*\***4** - T\*\***4**) \

+ alpha\*Q)

**return** dTdt

######################################################

# Do not adjust anything below this point #

######################################################

# Connect to Arduino

a = itclab.iTCLab()

# Turn LED on

**print**('LED On')

a.LED(**100**)

# Run time in minutes

run\_time = **15.0**

# Number of cycles

loops = int(**60.0**\*run\_time)

tm = np.zeros(loops)

# Temperature

# set point (degC)

Tsp1 = np.ones(loops) \* **25.0**

Tsp1[**60**:] = **45.0**

Tsp1[**360**:] = **30.0**

Tsp1[**660**:] = **35.0**

T1 = np.ones(loops) \* a.T1 # measured T (degC)

error\_sp = np.zeros(loops)

Tsp2 = np.ones(loops) \* **23.0** # set point (degC)

T2 = np.ones(loops) \* a.T2 # measured T (degC)

# Predictions

Tp = np.ones(loops) \* a.T1

error\_eb = np.zeros(loops)

Tpl = np.ones(loops) \* a.T1

error\_fopdt = np.zeros(loops)

# impulse tests (0 - 100%)

Q1 = np.ones(loops) \* **0.0**

Q2 = np.ones(loops) \* **0.0**

**print**('Running Main Loop. Ctrl-C to end.')

**print**(' Time SP PV Q1 = P + I + D')

**print**(('{:6.1f} {:6.2f} {:6.2f} ' + \

'{:6.2f} {:6.2f} {:6.2f} {:6.2f}').format( \

tm[**0**],Tsp1[**0**],T1[**0**], \

Q1[**0**],**0.0**,**0.0**,**0.0**))

# Create plot

plt.figure(figsize=(**10**,**7**))

plt.ion()

plt.show()

# Main Loop

start\_time = time.time()

prev\_time = start\_time

# Integral error

ierr = **0.0**

**try**:

**for** i **in** range(**1**,loops):

# Sleep time

sleep\_max = **1.0**

sleep = sleep\_max - (time.time() - prev\_time)

**if** sleep>=**0.01**:

time.sleep(sleep-**0.01**)

**else**:

time.sleep(**0.01**)

# Record time and change in time

t = time.time()

dt = t - prev\_time

prev\_time = t

tm[i] = t - start\_time

# Read temperatures in Kelvin

T1[i] = a.T1

T2[i] = a.T2

# Simulate one time step with Energy Balance

Tnext = odeint(heat,Tp[i-**1**]+**273.15**,[**0**,dt],args=(Q1[i-**1**],))

Tp[i] = Tnext[**1**]-**273.15**

# Simulate one time step with linear FOPDT model

z = np.exp(-dt/tauP)

Tpl[i] = (Tpl[i-**1**]-Tss) \* z \

+ (Q1[max(**0**,i-int(thetaP)-**1**)]-Qss)\*(**1**-z)\*Kp \

+ Tss

# Calculate PID output

[Q1[i],P,ierr,D] = pid(Tsp1[i],T1[i],T1[i-**1**],ierr,dt)

# Start setpoint error accumulation after 1 minute (60 seconds)

**if** i>=**60**:

error\_eb[i] = error\_eb[i-**1**] + abs(Tp[i]-T1[i])

error\_fopdt[i] = error\_fopdt[i-**1**] + abs(Tpl[i]-T1[i])

error\_sp[i] = error\_sp[i-**1**] + abs(Tsp1[i]-T1[i])

# Write output (0-100)

a.Q1(Q1[i])

a.Q2(**0.0**)

# Print line of data

**print**(('{:6.1f} {:6.2f} {:6.2f} ' + \

'{:6.2f} {:6.2f} {:6.2f} {:6.2f}').format( \

tm[i],Tsp1[i],T1[i], \

Q1[i],P,ierr,D))

# Plot

plt.clf()

ax=plt.subplot(**4**,**1**,**1**)

ax.grid()

plt.plot(tm[**0**:i],T1[**0**:i],'r.',label=r'$T\_1$ measured')

plt.plot(tm[**0**:i],Tsp1[**0**:i],'k--',label=r'$T\_1$ set point')

plt.ylabel('Temperature (degC)')

plt.legend(loc=**2**)

ax=plt.subplot(**4**,**1**,**2**)

ax.grid()

plt.plot(tm[**0**:i],Q1[**0**:i],'b-',label=r'$Q\_1$')

plt.ylabel('Heater')

plt.legend(loc='best')

ax=plt.subplot(**4**,**1**,**3**)

ax.grid()

plt.plot(tm[**0**:i],T1[**0**:i],'r.',label=r'$T\_1$ measured')

plt.plot(tm[**0**:i],Tp[**0**:i],'k-',label=r'$T\_1$ energy balance')

plt.plot(tm[**0**:i],Tpl[**0**:i],'g-',label=r'$T\_1$ linear model')

plt.ylabel('Temperature (degC)')

plt.legend(loc=**2**)

ax=plt.subplot(**4**,**1**,**4**)

ax.grid()

plt.plot(tm[**0**:i],error\_sp[**0**:i],'r-',label='Set Point Error')

plt.plot(tm[**0**:i],error\_eb[**0**:i],'k-',label='Energy Balance Error')

plt.plot(tm[**0**:i],error\_fopdt[**0**:i],'g-',label='Linear Model Error')

plt.ylabel('Cumulative Error')

plt.legend(loc='best')

plt.xlabel('Time (sec)')

plt.draw()

plt.pause(**0.05**)

# Turn off heaters

a.Q1(**0**)

a.Q2(**0**)

# Save figure

plt.savefig('test\_PID.png')

# Allow user to end loop with Ctrl-C

**except** **KeyboardInterrupt**:

# Disconnect from Arduino

a.Q1(**0**)

a.Q2(**0**)

**print**('Shutting down')

a.close()

plt.savefig('test\_PID.png')

# Make sure serial connection still closes when there's an error

**except**:

# Disconnect from Arduino

a.Q1(**0**)

a.Q2(**0**)

**print**('Error: Shutting down')

a.close()

plt.savefig('test\_PID.png')

**raise**

a.close()

## iTCLab 7

**Code PID\_Python.ino**

/\*

iTCLab Internet-Based Temperature Control Lab Firmware

Jeffrey Kantor, Initial Version

John Hedengren, Modified

Oct **2017**

Basuki Rahmat, Modified

April **2022**

This firmware **is** loaded into the Internet-Based Temperature Control Laboratory ESP32 to

provide a high level interface to the Internet-Based Temperature Control Lab. The firmware

scans the serial port looking **for** case-insensitive commands:

Q1 set Heater **1**, range **0** to **100**% subject to limit (**0**-**255** int)

Q2 set Heater **2**, range **0** to **100**% subject to limit (**0**-**255** int)

T1 get Temperature T1, returns deg C **as** string

T2 get Temperature T2, returns dec C **as** string

VER get firmware version string

X stop, enter sleep mode

Limits on the heater can be configured **with** the constants below.

\*/

#include <Arduino.h>

// constants

const String vers = "1.04"; // version of this firmware

const int baud = **115200**; // serial baud rate

const char sp = ' '; // command separator

const char nl = '**\n**'; // command terminator

// pin numbers corresponding to signals on the iTCLab Shield

const int pinT1 = **34**; // T1

const int pinT2 = **35**; // T2

const int pinQ1 = **32**; // Q1

const int pinQ2 = **33**; // Q2

const int pinLED = **26**; // LED

// setting PWM properties

const int freq = **5000**; //**5000**

const int ledChannel = **0**;

const int Q1Channel = **1**;

const int Q2Channel = **2**;

const int resolutionLedChannel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ1Channel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ2Channel = **8**; //Resolution **8**, **10**, **12**, **15**

const double upper\_temperature\_limit = **59**;

// **global** variables

char Buffer[**64**]; // buffer **for** parsing serial input

String cmd; // command

double pv = **0**; // pin value

float level; // LED level (**0**-**100**%)

double Q1 = **0**; // value written to Q1 pin

double Q2 = **0**; // value written to Q2 pin

int iwrite = **0**; // integer value **for** writing

float dwrite = **0**; // float value **for** writing

int n = **10**; // number of samples **for** each temperature measurement

void parseSerial(void) {

int ByteCount = Serial.readBytesUntil(nl,Buffer,sizeof(Buffer));

String read\_ = String(Buffer);

memset(Buffer,**0**,sizeof(Buffer));

// separate command **from** **associated** **data**

int idx = read\_.indexOf(sp);

cmd = read\_.substring(**0**,idx);

cmd.trim();

cmd.toUpperCase();

// extract data. toInt() returns **0** on error

String data = read\_.substring(idx+**1**);

data.trim();

pv = data.toFloat();

}

// Q1\_max = **100**%

// Q2\_max = **100**%

void dispatchCommand(void) {

**if** (cmd == "Q1") {

Q1 = max(**0.0**, min(**25.0**, pv));

iwrite = int(Q1 \* **2.0**); // **10.**? max

iwrite = max(**0**, min(**255**, iwrite));

ledcWrite(Q1Channel,iwrite);

Serial.println(Q1);

}

**else** **if** (cmd == "Q2") {

Q2 = max(**0.0**, min(**25.0**, pv));

iwrite = int(Q2 \* **2.0**); // **10.**? max

iwrite = max(**0**, min(**255**, iwrite));

ledcWrite(Q2Channel,iwrite);

Serial.println(Q2);

}

**else** **if** (cmd == "T1") {

float mV = **0.0**;

float degC = **0.0**;

**for** (int i = **0**; i < n; i++) {

mV = (float) analogRead(pinT1) \* **0.322265625**;

degC = degC + mV/**10.0**;

}

degC = degC / float(n);

Serial.println(degC);

}

**else** **if** (cmd == "T2") {

float mV = **0.0**;

float degC = **0.0**;

**for** (int i = **0**; i < n; i++) {

mV = (float) analogRead(pinT2) \* **0.322265625**;

degC = degC + mV/**10.0**;

}

degC = degC / float(n);

Serial.println(degC);

}

**else** **if** ((cmd == "V") **or** (cmd == "VER")) {

Serial.println("TCLab Firmware Version " + vers);

}

**else** **if** (cmd == "LED") {

level = max(**0.0**, min(**100.0**, pv));

iwrite = int(level \* **0.5**);

iwrite = max(**0**, min(**50**, iwrite));

ledcWrite(ledChannel, iwrite);

Serial.println(level);

}

**else** **if** (cmd == "X") {

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

Serial.println("Stop");

}

}

// check temperature **and** shut-off heaters **if** above high limit

void checkTemp(void) {

float mV = (float) analogRead(pinT1) \* **0.322265625**;

//float degC = (mV - **500.0**)/**10.0**;

float degC = mV/**10.0**;

**if** (degC >= upper\_temperature\_limit) {

Q1 = **0.0**;

Q2 = **0.0**;

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

//Serial.println("High Temp 1 (> upper\_temperature\_limit): ");

Serial.println(degC);

}

mV = (float) analogRead(pinT2) \* **0.322265625**;

//degC = (mV - **500.0**)/**10.0**;

degC = mV/**10.0**;

**if** (degC >= upper\_temperature\_limit) {

Q1 = **0.0**;

Q2 = **0.0**;

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

//Serial.println("High Temp 2 (> upper\_temperature\_limit): ");

Serial.println(degC);

}

}

// arduino startup

void setup() {

//analogReference(EXTERNAL);

Serial.begin(baud);

**while** (!Serial) {

; // wait **for** serial port to connect.

}

// configure pinQ1 PWM functionalitites

ledcSetup(Q1Channel, freq, resolutionQ1Channel);

// attach the channel to the pinQ1 to be controlled

ledcAttachPin(pinQ1, Q1Channel);

// configure pinQ2 PWM functionalitites

ledcSetup(Q2Channel, freq, resolutionQ2Channel);

// attach the channel to the pinQ2 to be controlled

ledcAttachPin(pinQ2, Q2Channel);

// configure pinLED PWM functionalitites

ledcSetup(ledChannel, freq, resolutionLedChannel);

// attach the channel to the pinLED to be controlled

ledcAttachPin(pinLED, ledChannel);

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

}

// arduino main event loop

void loop() {

parseSerial();

dispatchCommand();

checkTemp();

}

**Code itclab.py**

**import** **sys**

**import** **time**

**import** **numpy** **as** **np**

**try**:

**import** **serial**

**except**:

**import** **pip**

pip.main(['install','pyserial'])

**import** **serial**

**from** **serial.tools** **import** list\_ports

**class** **iTCLab**(object):

**def** **\_\_init\_\_**(self, port=None, baud=**115200**):

port = self.findPort()

**print**('Opening connection')

self.sp = serial.Serial(port=port, baudrate=baud, timeout=**2**)

self.sp.flushInput()

self.sp.flushOutput()

time.sleep(**3**)

**print**('iTCLab connected via Arduino on port ' + port)

**def** **findPort**(self):

found = False

**for** port **in** list(list\_ports.comports()):

# Arduino Uno

**if** port[**2**].startswith('USB VID:PID=16D0:0613'):

port = port[**0**]

found = True

# Arduino HDuino

**if** port[**2**].startswith('USB VID:PID=1A86:7523'):

port = port[**0**]

found = True

# Arduino Leonardo

**if** port[**2**].startswith('USB VID:PID=2341:8036'):

port = port[**0**]

found = True

# Arduino ESP32

**if** port[**2**].startswith('USB VID:PID=10C4:EA60'):

port = port[**0**]

found = True

# Arduino ESP32 - Tipe yg berbeda

**if** port[**2**].startswith('USB VID:PID=1A86:55D4'):

port = port[**0**]

found = True

**if** (**not** found):

**print**('Arduino COM port not found')

**print**('Please ensure that the USB cable is connected')

**print**('--- Printing Serial Ports ---')

**for** port **in** list(serial.tools.list\_ports.comports()):

**print**(port[**0**] + ' ' + port[**1**] + ' ' + port[**2**])

**print**('For Windows:')

**print**(' Open device manager, select "Ports (COM & LPT)"')

**print**(' Look for COM port of Arduino such as COM4')

**print**('For MacOS:')

**print**(' Open terminal and type: ls /dev/\*.')

**print**(' Search for /dev/tty.usbmodem\* or /dev/tty.usbserial\*. The port number is \*.')

**print**('For Linux')

**print**(' Open terminal and type: ls /dev/tty\*')

**print**(' Search for /dev/ttyUSB\* or /dev/ttyACM\*. The port number is \*.')

**print**('')

port = input('Input port: ')

# or hard-code it here

#port = 'COM3' # for Windows

#port = '/dev/tty.wchusbserial1410' # for MacOS

**return** port

**def** **stop**(self):

**return** self.read('X')

**def** **version**(self):

**return** self.read('VER')

**@property**

**def** **T1**(self):

self.\_T1 = float(self.read('T1'))

**return** self.\_T1

**@property**

**def** **T2**(self):

self.\_T2 = float(self.read('T2'))

**return** self.\_T2

**def** **LED**(self,pwm):

pwm = max(**0.0**,min(**100.0**,pwm))/**2.0**

self.write('LED',pwm)

**return** pwm

**def** **Q1**(self,pwm):

pwm = max(**0.0**,min(**100.0**,pwm))

self.write('Q1',pwm)

**return** pwm

**def** **Q2**(self,pwm):

pwm = max(**0.0**,min(**100.0**,pwm))

self.write('Q2',pwm)

**return** pwm

# save txt file with data and set point

# t = time

# u1,u2 = heaters

# y1,y2 = tempeatures

# sp1,sp2 = setpoints

**def** **save\_txt**(self,t,u1,u2,y1,y2,sp1,sp2):

data = np.vstack((t,u1,u2,y1,y2,sp1,sp2)) # vertical stack

data = data.T # transpose data

top = 'Time (sec), Heater 1 (%), Heater 2 (%), ' \

+ 'Temperature 1 (degC), Temperature 2 (degC), ' \

+ 'Set Point 1 (degC), Set Point 2 (degC)'

np.savetxt('data.txt',data,delimiter=',',header=top,comments='')

**def** **read**(self,cmd):

cmd\_str = self.build\_cmd\_str(cmd,'')

**try**:

self.sp.write(cmd\_str.encode())

self.sp.flush()

**except** **Exception**:

**return** None

**return** self.sp.readline().decode('UTF-8').replace("**\r\n**", "")

**def** **write**(self,cmd,pwm):

cmd\_str = self.build\_cmd\_str(cmd,(pwm,))

**try**:

self.sp.write(cmd\_str.encode())

self.sp.flush()

**except**:

**return** None

**return** self.sp.readline().decode('UTF-8').replace("**\r\n**", "")

**def** **build\_cmd\_str**(self,cmd, args=None):

"""

Build a command string that can be sent to the arduino.

Input:

cmd (str): the command to send to the arduino, must not

contain a % character

args (iterable): the arguments to send to the command

"""

**if** args:

args = ' '.join(map(str, args))

**else**:

args = ''

**return** "{cmd} {args}**\n**".format(cmd=cmd, args=args)

**def** **close**(self):

**try**:

self.sp.close()

**print**('Arduino disconnected successfully')

**except**:

**print**('Problems disconnecting from Arduino.')

**print**('Please unplug and reconnect Arduino.')

**return** True

**Code control\_arduino.py**

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Wed Jan 16 10:24:26 2019

@author: evertoncolling

@email: evertoncolling@gmail.com

@licence: MIT

"""

**from** **\_\_future\_\_** **import** print\_function, division

**from** **ipywidgets** **import** widgets **as** wi

**from** **IPython.display** **import** display

**import** **threading**

**import** **time**

**import** **numpy** **as** **np**

**from** **gekko** **import** GEKKO

**import** **bqplot** **as** **bq**

**from** **itclab** **import** iTCLab

**class** **GUI**(object):

"""

Class that defines the \_GUI applications

"""

**def** **\_\_init\_\_**(self):

"""

Initialize the \_GUI elements

"""

#######################################################################

# PLOTTING CONFIGURATION

#######################################################################

colors = [

'#1f77b4', # T1

'#5ebdff', # T1\_SP

'#d62728', # T2

'#ff7f0e', # T2\_SP

'#1f77b4', # Q1

'#d62728' # Q2

]

# Create scales

t\_sc = bq.LinearScale()

T\_sc = bq.LinearScale(min=**20.**, max=**60.**)

Q\_sc = bq.LinearScale(min=**0.**, max=**100.**)

T\_sc\_set = {'x': t\_sc, 'y': T\_sc}

Q\_sc\_set = {'x': t\_sc, 'y': Q\_sc}

# Create Axis

ax\_x1 = bq.Axis(label="", scale=t\_sc, visible=False) # upper

ax\_x2 = bq.Axis(label="time [min]", scale=t\_sc) # down

ax\_T1 = bq.Axis(label="Temperature 1 [C]", scale=T\_sc,

orientation="vertical", label\_color=colors[**0**])

ax\_T2 = bq.Axis(label="Temperature 2 [C]", scale=T\_sc,

orientation="vertical", label\_color=colors[**2**])

ax\_Q = bq.Axis(label="Heater Output [%]", scale=Q\_sc,

orientation="vertical", label\_color="black")

# Create Lines/Markers

self.\_T1\_meas = bq.Scatter(x=[], y=[], scales=T\_sc\_set,

marker='circle', colors=[colors[**0**]],

display\_legend=False, default\_size=**20**)

self.\_T2\_meas = bq.Scatter(x=[], y=[], scales=T\_sc\_set,

marker='circle', colors=[colors[**2**]],

display\_legend=False, default\_size=**20**)

self.\_T1\_set\_point = bq.Lines(x=[], y=[], scales=T\_sc\_set,

stroke\_width=**4**, colors=[colors[**1**]],

interpolation='step-before',

display\_legend=False)

self.\_T2\_set\_point = bq.Lines(x=[], y=[], scales=T\_sc\_set,

stroke\_width=**4**, colors=[colors[**3**]],

interpolation='step-before',

display\_legend=False)

self.\_u1 = bq.Lines(x=[], y=[], scales=Q\_sc\_set,

stroke\_width=**2**, colors=[colors[**4**]],

interpolation='step-before',

display\_legend=True, labels=['Heater 1'])

self.\_u2 = bq.Lines(x=[], y=[], scales=Q\_sc\_set,

stroke\_width=**2**, colors=[colors[**5**]],

interpolation='step-before',

display\_legend=True, labels=['Heater 2'])

# Mix everything and create figures

fig\_lay\_up = wi.Layout(width="400px", height="240px")

fig\_lay\_down = wi.Layout(width="400px", height="290px")

box1 = dict(top=**10**, bottom=**0**, left=**60**, right=**5**)

box2 = dict(top=**10**, bottom=**0**, left=**60**, right=**20**)

box3 = dict(top=**10**, bottom=**50**, left=**60**, right=**10**)

fig1 = bq.Figure(layout=fig\_lay\_up, axes=[ax\_x1, ax\_T1],

marks=[self.\_T1\_meas, self.\_T1\_set\_point],

fig\_margin=box1)

fig2 = bq.Figure(layout=fig\_lay\_up, axes=[ax\_x1, ax\_T2],

marks=[self.\_T2\_meas, self.\_T2\_set\_point],

fig\_margin=box2)

fig3 = bq.Figure(layout=fig\_lay\_down, axes=[ax\_x2, ax\_Q],

marks=[self.\_u1, self.\_u2], fig\_margin=box3,

legend\_location="left",

legend\_style={"fill": "white",

"fill-opacity": "0.7",

"width": "130px"})

#######################################################################

# SPACING WIDGETS CRIATION

#######################################################################

v\_space = wi.Label(value="", layout=wi.Layout(height='2px'))

h\_space = wi.Label(value="", layout=wi.Layout(width='2px'))

#######################################################################

# VARYING ARRAYS TO STORE DATA

#######################################################################

self.\_t = np.array([])

self.\_Q1 = np.array([])

self.\_Q2 = np.array([])

self.\_T = np.array([[]]).reshape((**0**, **2**))

self.\_SP\_T1 = np.array([])

self.\_SP\_T2 = np.array([])

#######################################################################

# PARAMETERS

#######################################################################

self.\_delta\_t = **4.0**

self.\_maxtime = int(**500**/self.\_delta\_t)

self.\_T1\_SP = **30**

self.\_T2\_SP = **30**

self.\_Q10 = **0**

self.\_Q20 = **0**

self.\_flag = False

self.\_sleep = **0.5**

self.\_Tc0 = np.array([**293.15**, **293.15**])

self.\_q1\_dt\_on\_off = **0.1**

self.\_q2\_dt\_on\_off = **0.1**

self.\_pid1\_gain = **10.**

self.\_pid1\_reset = **50.**

self.\_pid1\_rate = **1.**

self.\_pid2\_gain = **10.**

self.\_pid2\_reset = **50.**

self.\_pid2\_rate = **1.**

self.\_SOLVER = '1 - APOPT'

self.\_CVTYPE = '1 - Deadband'

self.\_T1\_dt = **0.1**

self.\_T1\_tau = **10.**

self.\_T2\_dt = **0.1**

self.\_T2\_tau = **10.**

self.\_Q1\_DMAX = **30.**

self.\_Q1\_DCOST = **1.**

self.\_Q2\_DMAX = **30.**

self.\_Q2\_DCOST = **1.**

#######################################################################

# OUTPUT WIDGETS CRIATION

#######################################################################

style = {'description\_width': 'initial'}

self.\_PT1 = wi.FloatProgress(value=self.\_Tc0[**0**]-**273.15**, min=**0**,

max=**100.0**,

description='<b>PV (°C):</b>',

bar\_style='warning',

orientation='horizontal',

style=style)

self.\_LT1 = wi.Label(value=str(self.\_PT1.value))

wi.jslink((self.\_PT1, 'value'), (self.\_LT1, 'value'))

self.\_PT2 = wi.FloatProgress(value=self.\_Tc0[**1**]-**273.15**, min=**0**,

max=**100.0**,

description='<b>PV (°C):</b>',

bar\_style='warning',

orientation='horizontal',

style=style)

self.\_LT2 = wi.Label(value=str(self.\_PT2.value))

wi.jslink((self.\_PT2, 'value'), (self.\_LT2, 'value'))

#######################################################################

# INTERACTION WIDGETS CREATION

#######################################################################

self.\_wQ1 = wi.FloatSlider(value=self.\_Q10, min=**0**, max=**100.0**, step=**0.5**,

description='<b>Q1 (%):</b>',

continuous\_update=False,

orientation='horizontal',

readout=False, style=style,

layout=wi.Layout(width='230px'))

self.\_tQ1 = wi.BoundedFloatText(value=self.\_Q10, min=**0**, max=**100.0**,

step=**0.5**,

layout=wi.Layout(width='60px'))

wi.jslink((self.\_wQ1, 'value'), (self.\_tQ1, 'value'))

self.\_bQ1 = wi.Button(description='Set',

layout=wi.Layout(width='50px'))

self.\_bQ1.on\_click(self.\_Q1\_click)

self.\_wQ2 = wi.FloatSlider(value=self.\_Q20, min=**0**, max=**100.0**, step=**0.5**,

description='<b>Q2 (%):</b>',

continuous\_update=False,

orientation='horizontal',

readout=False, style=style,

layout=wi.Layout(width='230px'))

self.\_tQ2 = wi.BoundedFloatText(value=self.\_Q20, min=**0**, max=**100.0**,

step=**0.5**,

layout=wi.Layout(width='60px'))

wi.jslink((self.\_wQ2, 'value'), (self.\_tQ2, 'value'))

self.\_bQ2 = wi.Button(description='Set',

layout=wi.Layout(width='50px'))

self.\_bQ2.on\_click(self.\_Q2\_click)

self.\_wT1 = wi.FloatSlider(value=self.\_T1\_SP, min=**20**, max=**60.0**,

step=**0.5**, description='<b>T1 SP:</b>',

continuous\_update=False,

orientation='horizontal',

readout=False, style=style,

disabled=True,

layout=wi.Layout(width='230px'))

self.\_tT1 = wi.BoundedFloatText(value=self.\_T1\_SP, min=**20**, max=**60.0**,

step=**0.5**, disabled=True,

layout=wi.Layout(width='60px'))

wi.jslink((self.\_wT1, 'value'), (self.\_tT1, 'value'))

self.\_bT1 = wi.Button(description='Set',

layout=wi.Layout(width='50px'),

disabled=True)

self.\_bT1.on\_click(self.\_T1\_click)

self.\_wT2 = wi.FloatSlider(value=self.\_T2\_SP, min=**20**, max=**60.0**,

step=**0.5**, description='<b>T2 SP:</b>',

continuous\_update=False,

orientation='horizontal', disabled=True,

readout=False, style=style,

layout=wi.Layout(width='230px'))

self.\_tT2 = wi.BoundedFloatText(value=self.\_T2\_SP, min=**20**, max=**60.0**,

step=**0.5**, disabled=True,

layout=wi.Layout(width='60px'))

wi.jslink((self.\_wT2, 'value'), (self.\_tT2, 'value'))

self.\_bT2 = wi.Button(description='Set',

layout=wi.Layout(width='50px'), disabled=True)

self.\_bT2.on\_click(self.\_T2\_click)

#######################################################################

# MODE SELECTION

#######################################################################

self.\_mode = wi.ToggleButtons(options=['Manual',

'On-Off',

'PID',

'MPC'],

style={'button\_width': '100px'})

self.\_mode.observe(self.\_mode\_switch, names='value')

#######################################################################

# STOP THREAD

#######################################################################

self.\_b\_stop = wi.Button(description='Stop', button\_style='warning',

icon='stop', layout=wi.Layout(width='100px',

height='32px'))

self.\_b\_stop.on\_click(self.\_stop\_click)

#######################################################################

# START THREAD - OPEN LOOP

#######################################################################

self.\_b\_play = wi.Button(description='Start', button\_style='success',

icon='play', layout=wi.Layout(width='100px',

height='32px'))

self.\_b\_play.on\_click(self.\_play\_click)

# Join Buttons

buttons = wi.HBox((self.\_b\_play, h\_space, self.\_b\_stop,

wi.Label(value="", layout=wi.Layout(width='165px')),

self.\_mode))

#######################################################################

# LAYOUT

#######################################################################

Q1\_Set = wi.HBox((h\_space, self.\_wQ1, self.\_tQ1, self.\_bQ1),

layout=wi.Layout(border='solid 2px gray',

width='360px'))

T1\_Set = wi.HBox((h\_space, self.\_wT1, self.\_tT1, self.\_bT1),

layout=wi.Layout(border='solid 2px gray',

width='360px'))

T1\_View = wi.HBox((h\_space, self.\_PT1, self.\_LT1),

layout=wi.Layout(border='solid 2px gray',

width='360px'))

Q2\_Set = wi.HBox((h\_space, self.\_wQ2, self.\_tQ2, self.\_bQ2),

layout=wi.Layout(border='solid 2px gray',

width='360px'))

T2\_Set = wi.HBox((h\_space, self.\_wT2, self.\_tT2, self.\_bT2),

layout=wi.Layout(border='solid 2px gray',

width='360px'))

T2\_View = wi.HBox((h\_space, self.\_PT2, self.\_LT2),

layout=wi.Layout(border='solid 2px gray',

width='360px'))

co1 = wi.VBox((T1\_Set, v\_space, T1\_View, v\_space, Q1\_Set))

co2 = wi.VBox((T2\_Set, v\_space, T2\_View, v\_space, Q2\_Set))

co = wi.VBox((v\_space, v\_space, co1, v\_space, v\_space, v\_space, co2))

fig1x = wi.HBox((fig1, fig2))

fig2x = wi.HBox((fig3, h\_space, h\_space, co))

figy = wi.VBox((fig1x, v\_space, fig2x),

layout=wi.Layout(border='solid 2px gray',

width='800px'))

#######################################################################

# APPLICATION TO DISPLAY

#######################################################################

self.\_gui = wi.VBox((buttons, v\_space, figy))

#######################################################################

# CONFIGURATOR

#######################################################################

style = {'description\_width': 'initial'}

lay = wi.Layout(width='120px', margin='0 20px 0 0')

#######################################################################

# GENERAL OPTIONS

#######################################################################

self.\_conf11 = wi.HBox((

wi.HTML(

value='<p style="text-align: right;"><b>&Delta;t (s):</b></p>',

layout=lay),

wi.FloatSlider(value=**4.0**, min=**1.0**, max=**10.0**, step=**0.5**,

description='', style=style)))

self.\_conf12 = wi.HBox((

wi.HTML(value='<p style="text-align: right;">'

'<b>Sleep time (s):</b></p>',

layout=lay),

wi.FloatSlider(value=**0.5**, min=**0.25**, max=**1.5**, step=**0.05**,

description='', style=style)))

but11 = wi.Button(description='Apply', icon='check',

layout=wi.Layout(width='100px', height='32px'))

but11.on\_click(self.\_conf\_general)

but12 = wi.Button(description='Reset', icon='refresh',

layout=wi.Layout(width='100px', height='32px'))

but12.on\_click(self.\_reset\_general)

conf13 = wi.HBox((but11, but12), layout=wi.Layout(margin='10px 0 0 0'))

#######################################################################

# ON-OFF OPTIONS

#######################################################################

lay5 = wi.Layout(width='90px', margin='0 10px 0 15px')

conf20 = wi.Button(description='TEMPERATURE 01 / HEATER 01',

disabled=True,

layout=wi.Layout(width='320px', margin='0 0 0 0'))

self.\_conf21 = wi.HBox((

wi.HTML(value='<p style="text-align: right;">'

'<b>Deadband (K):</b></p>',

layout=lay5),

wi.FloatSlider(value=**0.1**, min=**0.0**, max=**2.0**, step=**0.1**,

description='', style=style,

layout=wi.Layout(width='200px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

box21 = wi.VBox((conf20, self.\_conf21),

layout=wi.Layout(border='solid 2px gray',

width='325px',

margin='10px 0 0 0px'))

conf22 = wi.Button(description='TEMPERATURE 02 / HEATER 02',

disabled=True,

layout=wi.Layout(width='320px', margin='0 0 0 0'))

self.\_conf23 = wi.HBox((

wi.HTML(value='<p style="text-align: right;">'

'<b>Deadband (K):</b></p>',

layout=lay5),

wi.FloatSlider(value=**0.1**, min=**0.0**, max=**2.0**, step=**0.1**,

description='', style=style,

layout=wi.Layout(width='200px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

box22 = wi.VBox((conf22, self.\_conf23),

layout=wi.Layout(border='solid 2px gray',

width='325px',

margin='10px 0 0 10px'))

but21 = wi.Button(description='Apply', icon='check',

layout=wi.Layout(width='100px', height='32px'))

but21.on\_click(self.\_conf\_on\_off)

but22 = wi.Button(description='Reset', icon='refresh',

layout=wi.Layout(width='100px', height='32px'))

but22.on\_click(self.\_reset\_on\_off)

conf24 = wi.HBox((but21, but22), layout=wi.Layout(margin='10px 0 0 0'))

#######################################################################

# PID OPTIONS

#######################################################################

lay4 = wi.Layout(width='100px', margin='0 10px 0 15px')

conf30 = wi.Button(description='TEMPERATURE 01 / HEATER 01',

disabled=True,

layout=wi.Layout(width='330px', margin='0 0 0 0'))

self.\_conf31 = wi.HBox((

wi.HTML(value='<p style="text-align: right;">'

'<b>Kc (K/%Heater):</b></p>',

layout=lay4),

wi.FloatSlider(value=**10.0**, min=**0.0**, max=**20.0**, step=**0.5**,

description='', style=style,

layout=wi.Layout(width='200px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

self.\_conf32 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>tauI (s):</b></p>',

layout=lay4),

wi.FloatSlider(value=**50.0**, min=**0.0**, max=**200.0**, step=**1.0**,

description='', style=style,

layout=wi.Layout(width='200px'))))

self.\_conf33 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>tauD (s):</b></p>',

layout=lay4),

wi.FloatSlider(value=**1.0**, min=**0.0**, max=**10.0**, step=**0.5**,

description='', style=style,

layout=wi.Layout(width='200px'))))

box31 = wi.VBox((conf30, self.\_conf31, self.\_conf32, self.\_conf33),

layout=wi.Layout(border='solid 2px gray',

width='335px',

margin='10px 0 0 0px'))

conf34 = wi.Button(description='TEMPERATURE 02 / HEATER 02',

disabled=True,

layout=wi.Layout(width='330px', margin='0 0 0 0'))

self.\_conf35 = wi.HBox((

wi.HTML(value='<p style="text-align: right;">'

'<b>Kc (K/%Heater):</b></p>',

layout=lay4),

wi.FloatSlider(value=**10.0**, min=**0.0**, max=**20.0**, step=**0.5**,

description='', style=style,

layout=wi.Layout(width='200px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

self.\_conf36 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>tauI (s):</b></p>',

layout=lay4),

wi.FloatSlider(value=**50.0**, min=**0.0**, max=**200.0**, step=**1.0**,

description='', style=style,

layout=wi.Layout(width='200px'))))

self.\_conf37 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>tauD (s):</b></p>',

layout=lay4),

wi.FloatSlider(value=**1.0**, min=**0.0**, max=**10.0**, step=**0.5**,

description='', style=style,

layout=wi.Layout(width='200px'))))

box32 = wi.VBox((conf34, self.\_conf35, self.\_conf36, self.\_conf37),

layout=wi.Layout(border='solid 2px gray',

width='335px',

margin='10px 0 0 10px'))

but31 = wi.Button(description='Apply', icon='check',

layout=wi.Layout(width='100px', height='32px'))

but31.on\_click(self.\_conf\_pid)

but32 = wi.Button(description='Reset', icon='refresh',

layout=wi.Layout(width='100px', height='32px'))

but32.on\_click(self.\_reset\_pid)

conf38 = wi.HBox((but31, but32), layout=wi.Layout(margin='10px 0 0 0'))

#######################################################################

# MPC OPTIONS

#######################################################################

lay1 = wi.Layout(width='60px', margin='0 10px 0 25px')

lay2 = wi.Layout(width='95px', margin='0 20px 0 10px')

lay3 = wi.Layout(width='55px', margin='0 20px 0 10px')

self.\_conf40 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>SOLVER:</b></p>',

layout=lay1),

wi.Dropdown(options=['1 - APOPT', '2 - BPOPT', '3 - IPOPT'],

value='1 - APOPT',

layout=wi.Layout(width='100px')),

wi.HTML(value='<p style="text-align: right;"><b>CVTYPE:</b></p>',

layout=lay1),

wi.Dropdown(options=['1 - Deadband', '2 - Trajectory'],

value='1 - Deadband',

layout=wi.Layout(width='130px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

conf41 = wi.Button(description='TEMPERATURE 01',

disabled=True,

layout=wi.Layout(width='360px', margin='0 0 0 0'))

self.\_conf42 = wi.HBox((

wi.HTML(value='<p style="text-align: right;">'

'<b>Deadband (K):</b></p>',

layout=lay2),

wi.FloatSlider(value=**0.1**, min=**0.1**, max=**1.0**, step=**0.1**,

description='', style=style,

layout=wi.Layout(width='250px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

self.\_conf43 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>TAU:</b></p>',

layout=lay2),

wi.FloatSlider(value=**10.0**, min=**1.0**, max=**100.0**, step=**1.0**,

description='', style=style,

layout=wi.Layout(width='250px'))))

box41 = wi.VBox((conf41, self.\_conf42, self.\_conf43),

layout=wi.Layout(border='solid 2px gray',

width='365px',

margin='10px 0 0 0'))

conf44 = wi.Button(description='TEMPERATURE 02',

disabled=True,

layout=wi.Layout(width='360px', margin='0 0 0 0'))

self.\_conf45 = wi.HBox((

wi.HTML(value='<p style="text-align: right;">'

'<b>Deadband (K):</b></p>',

layout=lay2),

wi.FloatSlider(value=**0.1**, min=**0.1**, max=**1.0**, step=**0.1**,

description='', style=style,

layout=wi.Layout(width='250px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

self.\_conf46 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>TAU:</b></p>',

layout=lay2),

wi.FloatSlider(value=**10.0**, min=**1.0**, max=**100.0**, step=**1.0**,

description='', style=style,

layout=wi.Layout(width='250px'))))

box42 = wi.VBox((conf44, self.\_conf45, self.\_conf46),

layout=wi.Layout(border='solid 2px gray',

width='365px',

margin='10px 0 0 0'))

conf47 = wi.Button(description='HEATER 01',

disabled=True,

layout=wi.Layout(width='320px', margin='0 0 0 0'))

self.\_conf48 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>DMAX:</b></p>',

layout=lay3),

wi.FloatSlider(value=**30.0**, min=**1.0**, max=**100.0**, step=**0.5**,

description='', style=style,

layout=wi.Layout(width='250px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

self.\_conf49 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>DCOST:</b></p>',

layout=lay3),

wi.FloatSlider(value=**1.0**, min=**0.0**, max=**10.0**, step=**0.1**,

description='', style=style,

layout=wi.Layout(width='250px'))))

box43 = wi.VBox((conf47, self.\_conf48, self.\_conf49),

layout=wi.Layout(border='solid 2px gray',

width='325px',

margin='10px 0 0 10px'))

conf410 = wi.Button(description='HEATER 02',

disabled=True,

layout=wi.Layout(width='320px', margin='0 0 0 0'))

self.\_conf411 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>DMAX:</b></p>',

layout=lay3),

wi.FloatSlider(value=**30.0**, min=**1.0**, max=**100.0**, step=**0.5**,

description='', style=style,

layout=wi.Layout(width='250px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

self.\_conf412 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>DCOST:</b></p>',

layout=lay3),

wi.FloatSlider(value=**1.0**, min=**0.0**, max=**10.0**, step=**0.1**,

description='', style=style,

layout=wi.Layout(width='250px'))))

box44 = wi.VBox((conf410, self.\_conf411, self.\_conf412),

layout=wi.Layout(border='solid 2px gray',

width='325px',

margin='10px 0 0 10px'))

but41 = wi.Button(description='Apply', icon='check',

layout=wi.Layout(width='100px', height='32px'))

but41.on\_click(self.\_conf\_mpc)

but42 = wi.Button(description='Reset', icon='refresh',

layout=wi.Layout(width='100px', height='32px'))

but42.on\_click(self.\_reset\_mpc)

conf413 = wi.HBox((but41, but42),

layout=wi.Layout(margin='10px 0 0 0'))

#######################################################################

# CONFIGURATOR LAYOUT

#######################################################################

tab = wi.Tab([wi.VBox((self.\_conf11, self.\_conf12,

wi.Label(layout=wi.Layout(height='206px')),

conf13)),

wi.VBox((wi.HBox((box21, box22)),

wi.Label(layout=wi.Layout(height='191px')),

conf24)),

wi.VBox((wi.HBox((box31, box32)),

wi.Label(layout=wi.Layout(height='127px')),

conf38)),

wi.VBox((self.\_conf40,

wi.HBox((box41, box43)),

wi.HBox((box42, box44)),

wi.Label(layout=wi.Layout(height='11px')),

conf413))],

layout=wi.Layout(width='800px', height='380px'))

tab.set\_title(**0**, 'General Options')

tab.set\_title(**1**, 'On-Off Options')

tab.set\_title(**2**, 'PID Options')

tab.set\_title(**3**, 'MPC Options')

#######################################################################

# DISPLAY CONFIGURATOR

#######################################################################

self.\_conf = tab

**def** **app**(self):

display(self.\_gui)

**def** **config**(self):

display(self.\_conf)

**def** **\_conf\_general**(self, b):

self.\_delta\_t = self.\_conf11.children[**1**].value

self.\_maxtime = int(**500**/self.\_delta\_t)

self.\_sleep = self.\_conf12.children[**1**].value

**def** **\_reset\_general**(self, b):

self.\_conf11.children[**1**].value = **4.0**

self.\_delta\_t = self.\_conf11.children[**1**].value

self.\_maxtime = int(**500**/self.\_delta\_t)

self.\_conf12.children[**1**].value = **0.5**

self.\_sleep = self.\_conf12.children[**1**].value

**def** **\_conf\_on\_off**(self, b):

self.\_q1\_dt\_on\_off = self.\_conf21.children[**1**].value

self.\_q2\_dt\_on\_off = self.\_conf23.children[**1**].value

**def** **\_reset\_on\_off**(self, b):

self.\_conf21.children[**1**].value = **0.1**

self.\_q1\_dt\_on\_off = self.\_conf21.children[**1**].value

self.\_conf23.children[**1**].value = **0.1**

self.\_q2\_dt\_on\_off = self.\_conf23.children[**1**].value

**def** **\_conf\_pid**(self, b):

self.\_pid1\_gain = self.\_conf31.children[**1**].value

self.\_pid1\_reset = self.\_conf32.children[**1**].value

self.\_pid1\_rate = self.\_conf33.children[**1**].value

self.\_pid2\_gain = self.\_conf35.children[**1**].value

self.\_pid2\_reset = self.\_conf36.children[**1**].value

self.\_pid2\_rate = self.\_conf37.children[**1**].value

**def** **\_reset\_pid**(self, b):

self.\_conf31.children[**1**].value = **10.0** # gain

self.\_conf32.children[**1**].value = **50.0** # reset

self.\_conf33.children[**1**].value = **1.0** # rate

self.\_pid1\_gain = self.\_conf31.children[**1**].value

self.\_pid1\_reset = self.\_conf32.children[**1**].value

self.\_pid1\_rate = self.\_conf33.children[**1**].value

self.\_conf35.children[**1**].value = **10.0** # gain

self.\_conf36.children[**1**].value = **50.0** # reset

self.\_conf37.children[**1**].value = **1.0** # rate

self.\_pid2\_gain = self.\_conf35.children[**1**].value

self.\_pid2\_reset = self.\_conf36.children[**1**].value

self.\_pid2\_rate = self.\_conf37.children[**1**].valu

**def** **\_conf\_mpc**(self, b):

self.\_SOLVER = self.\_conf40.children[**1**].value

self.\_CVTYPE = self.\_conf40.children[**3**].value

self.\_T1\_dt = self.\_conf42.children[**1**].value

self.\_T1\_tau = self.\_conf43.children[**1**].value

self.\_T2\_dt = self.\_conf45.children[**1**].value

self.\_T2\_tau = self.\_conf46.children[**1**].value

self.\_Q1\_DMAX = self.\_conf48.children[**1**].value

self.\_Q1\_DCOST = self.\_conf49.children[**1**].value

self.\_Q2\_DMAX = self.\_conf411.children[**1**].value

self.\_Q2\_DCOST = self.\_conf412.children[**1**].value

**def** **\_reset\_mpc**(self, b):

self.\_conf40.children[**1**].value = '1 - APOPT'

self.\_conf40.children[**3**].value = '1 - Deadband'

self.\_conf42.children[**1**].value = **0.1**

self.\_conf43.children[**1**].value = **30.**

self.\_conf45.children[**1**].value = **0.1**

self.\_conf46.children[**1**].value = **30.**

self.\_conf48.children[**1**].value = **30.**

self.\_conf49.children[**1**].value = **1.**

self.\_conf411.children[**1**].value = **30.**

self.\_conf412.children[**1**].value = **1.**

self.\_SOLVER = self.\_conf40.children[**1**].value

self.\_CVTYPE = self.\_conf40.children[**3**].value

self.\_T1\_dt = self.\_conf42.children[**1**].value

self.\_T1\_tau = self.\_conf43.children[**1**].value

self.\_T2\_dt = self.\_conf45.children[**1**].value

self.\_T2\_tau = self.\_conf46.children[**1**].value

self.\_Q1\_DMAX = self.\_conf48.children[**1**].value

self.\_Q1\_DCOST = self.\_conf49.children[**1**].value

self.\_Q2\_DMAX = self.\_conf411.children[**1**].value

self.\_Q2\_DCOST = self.\_conf412.children[**1**].value

**def** **\_Q1\_click**(self, b):

self.\_Q10 = self.\_wQ1.value

**def** **\_Q2\_click**(self, b):

self.\_Q20 = self.\_wQ2.value

**def** **\_T1\_click**(self, b):

self.\_T1\_SP = self.\_wT1.value

**def** **\_T2\_click**(self, b):

self.\_T2\_SP = self.\_wT2.value

**def** **\_stop\_click**(self, b):

self.\_flag = False

self.\_mode.disabled = False

**def** **\_play\_click**(self, b):

**if** **not** self.\_flag:

**if** self.\_mode.value == "Manual":

self.\_flag = True

self.\_mode.disabled = True

thread = threading.Thread(target=self.\_work\_man)

thread.start()

**elif** self.\_mode.value == "On-Off":

self.\_flag = True

self.\_mode.disabled = True

thread = threading.Thread(target=self.\_work\_on\_off)

thread.start()

**elif** self.\_mode.value == "PID":

self.\_flag = True

self.\_mode.disabled = True

thread = threading.Thread(target=self.\_work\_pid)

thread.start()

**elif** self.\_mode.value == "MPC":

self.\_flag = True

self.\_mode.disabled = True

thread = threading.Thread(target=self.\_work\_mpc)

thread.start()

**def** **\_mode\_switch**(self, value):

# Reinitialize parameters

self.\_T1\_SP = **30**

self.\_T2\_SP = **30**

self.\_Q10 = **0**

self.\_Q20 = **0**

# Reset figures

self.\_T1\_meas.x = []

self.\_T1\_meas.y = []

self.\_T2\_meas.x = []

self.\_T2\_meas.y = []

self.\_T1\_set\_point.x = []

self.\_T1\_set\_point.y = []

self.\_T2\_set\_point.x = []

self.\_T2\_set\_point.y = []

self.\_u1.x = []

self.\_u1.y = []

self.\_u2.x = []

self.\_u2.y = []

# Reset controls

self.\_PT1.value = self.\_Tc0[**0**]-**273.15**

self.\_PT2.value = self.\_Tc0[**1**]-**273.15**

self.\_wT1.value = self.\_T1\_SP

self.\_wT2.value = self.\_T2\_SP

self.\_wQ1.value = self.\_Q10

self.\_wQ2.value = self.\_Q20

**if** value['new'] == "Manual":

self.\_wQ1.disabled = False

self.\_tQ1.disabled = False

self.\_bQ1.disabled = False

self.\_wT1.disabled = True

self.\_tT1.disabled = True

self.\_bT1.disabled = True

self.\_wQ2.disabled = False

self.\_tQ2.disabled = False

self.\_bQ2.disabled = False

self.\_wT2.disabled = True

self.\_tT2.disabled = True

self.\_bT2.disabled = True

**else**:

self.\_wQ1.disabled = True

self.\_tQ1.disabled = True

self.\_bQ1.disabled = True

self.\_wT1.disabled = False

self.\_tT1.disabled = False

self.\_bT1.disabled = False

self.\_wQ2.disabled = True

self.\_tQ2.disabled = True

self.\_bQ2.disabled = True

self.\_wT2.disabled = False

self.\_tT2.disabled = False

self.\_bT2.disabled = False

###########################################################################

# PID CONTROLLER

###########################################################################

# inputs -----------------------------------

# sp = setpoint

# pv = current temperature

# pv\_last = prior temperature

# ierr = integral error

# dt = time increment between measurements

# outputs ----------------------------------

# op = output of the PID controller

# I = integral contribution

**def** **\_PID**(self, sp, pv, pv\_last, ierr, dt, Kc=**10.0**, tauI=**50.0**, tauD=**1.0**):

# Default Parameters

# Kc = 10.0 # K/%Heater

# tauI = 50.0 # sec

# tauD = 1.0 # sec

# Parameters in terms of PID coefficients

KP = Kc

**if** tauI == **0**:

KI = **1e5**

**else**:

KI = Kc/tauI

KD = Kc\*tauD

# ubias for controller (initial heater)

op0 = **0**

# upper and lower bounds on heater level

ophi = **100**

oplo = **0**

# calculate the error

error = sp-pv

# calculate the integral error

ierr = ierr + KI \* error \* dt

# calculate the measurement derivative

dpv = (pv - pv\_last) / dt

# calculate the PID output

P = KP \* error

I = ierr

D = -KD \* dpv

op = op0 + P + I + D

# implement anti-reset windup

**if** op < oplo **or** op > ophi:

I = I - KI \* error \* dt

# clip output

op = max(oplo, min(ophi, op))

# return the controller output and PID terms

**return** [op, I]

###########################################################################

# MPC

###########################################################################

**def** **\_MPC**(self):

m = GEKKO(remote=False)

# 60 second time horizon, 4 sec cycle time, non-uniform

m.time = [**0**, **4**, **8**, **12**, **15**, **20**, **25**, **30**, **35**, **40**, **50**, **60**, **70**, **80**, **90**]

# Parameters

m.U = m.FV(value=**10**)

m.tau = m.FV(value=**5**)

m.alpha1 = m.FV(value=**0.01**) # W / % heater

m.alpha2 = m.FV(value=**0.0075**) # W / % heater

# Manipulated variables

m.Q1 = m.MV(value=**0**)

m.Q1.STATUS = **1** # use to control temperature

m.Q1.FSTATUS = **0** # no feedback measurement

m.Q1.LOWER = **0.0**

m.Q1.UPPER = **100.0**

m.Q1.DMAX = **20.0**

m.Q1.COST = **0.0**

m.Q1.DCOST = **2.0**

m.Q2 = m.MV(value=**0**)

m.Q2.STATUS = **1** # use to control temperature

m.Q2.FSTATUS = **0** # no feedback measurement

m.Q2.LOWER = **0.0**

m.Q2.UPPER = **100.0**

m.Q2.DMAX = **20.0**

m.Q2.COST = **0.0**

m.Q2.DCOST = **2.0**

# Controlled variable

m.TC1 = m.CV(value=**22**)

m.TC1.STATUS = **1** # minimize error with setpoint range

m.TC1.FSTATUS = **1** # receive measurement

m.TC1.TR\_INIT = **1** # reference trajectory

m.TC1.TAU = **10** # time constant for response

# Controlled variable

m.TC2 = m.CV(value=**22**)

m.TC2.STATUS = **1** # minimize error with setpoint range

m.TC2.FSTATUS = **1** # receive measurement

m.TC2.TR\_INIT = **1** # reference trajectory

m.TC2.TAU = **10** # time constant for response

# State variables

m.TH1 = m.SV(value=**22**)

m.TH2 = m.SV(value=**22**)

m.Ta = m.Param(value=**23.0**+**273.15**) # K

m.mass = m.Param(value=**4.0**/**1000.0**) # kg

m.Cp = m.Param(value=**0.5**\***1000.0**) # J/kg-K

m.A = m.Param(value=**10.0**/**100.0**\*\***2**) # Area not between heaters in m^2

m.As = m.Param(value=**2.0**/**100.0**\*\***2**) # Area between heaters in m^2

m.eps = m.Param(value=**0.9**) # Emissivity

m.sigma = m.Const(**5.67e-8**) # Stefan-Boltzmann

# Heater temperatures

m.T1i = m.Intermediate(m.TH1+**273.15**)

m.T2i = m.Intermediate(m.TH2+**273.15**)

# Heat transfer between two heaters

m.Q\_C12 = m.Intermediate(m.U\*m.As\*(m.T2i-m.T1i)) # Conv

m.Q\_R12 = m.Intermediate(m.eps\*m.sigma\*m.As\*(m.T2i\*\***4**-m.T1i\*\***4**)) # Rad

# Semi-fundamental correlations (energy balances)

m.Equation(m.TH1.dt() == (**1.0**/(m.mass\*m.Cp)) \*

(m.U\*m.A\*(m.Ta-m.T1i) +

m.eps \* m.sigma \* m.A \*

(m.Ta\*\***4** - m.T1i\*\***4**) + m.Q\_C12 +

m.Q\_R12 + m.alpha1\*m.Q1))

m.Equation(m.TH2.dt() == (**1.0**/(m.mass\*m.Cp)) \*

(m.U\*m.A\*(m.Ta-m.T2i) +

m.eps \* m.sigma \* m.A \*

(m.Ta\*\***4** - m.T2i\*\***4**) - m.Q\_C12 -

m.Q\_R12 + m.alpha2\*m.Q2))

# Empirical correlations (lag equations to emulate conduction)

m.Equation(m.tau \* m.TC1.dt() == -m.TC1 + m.TH1)

m.Equation(m.tau \* m.TC2.dt() == -m.TC2 + m.TH2)

# Global Options

m.options.IMODE = **6** # MPC

m.options.CV\_TYPE = **1** # Objective type

m.options.NODES = **3** # Collocation nodes

m.options.SOLVER = **3** # 1=APOPT, 3=IPOPT

**return** m

###########################################################################

# THREADING FUNCTION - OPEN LOOP

###########################################################################

**def** **\_work\_man**(self):

**try**:

a = iTCLab()

**except**:

a.close()

a = iTCLab()

# Parater to start each cycle

self.\_Tc0 = np.array([

a.T1 + **273.15**,

a.T2 + **273.15**

])

# arrays to store data

t = np.array([])

Q1 = np.array([])

Q2 = np.array([])

T = np.array([[]]).reshape((**0**, **2**))

t = np.append(t, np.array([**0**]), axis=**0**)

T = np.append(T, self.\_Tc0.reshape((**1**, **2**)), axis=**0**)

Q1 = np.append(Q1, np.array([self.\_Q10]), axis=**0**)

Q2 = np.append(Q2, np.array([self.\_Q20]), axis=**0**)

# Main Loop

start\_time = time.time()

prev\_time = start\_time

**while** self.\_flag:

# Sleep time

sleep\_max = self.\_delta\_t

sleep = sleep\_max - (time.time() - prev\_time)

**if** sleep >= **0.01**:

time.sleep(sleep-**0.01**)

**else**:

time.sleep(**0.01**)

# Read temperatures in Celsius

self.\_Tc0 = np.array([

a.T1 + **273.15**,

a.T2 + **273.15**

])

# Write new heater values (0-100)

a.Q1(self.\_Q10)

a.Q2(self.\_Q20)

**if** len(t) >= self.\_maxtime:

t = np.delete(t, **0**, **0**)

T = np.delete(T, **0**, **0**)

Q1 = np.delete(Q1, **0**, **0**)

Q2 = np.delete(Q2, **0**, **0**)

# Record time and change in time

tm = time.time()

prev\_time = tm

tm = tm - start\_time

t = np.append(t, np.array([tm]), axis=**0**)

T = np.append(T, self.\_Tc0.reshape((**1**, **2**)), axis=**0**)

Q1 = np.append(Q1, np.array([self.\_Q10]), axis=**0**)

Q2 = np.append(Q2, np.array([self.\_Q20]), axis=**0**)

self.\_T1\_meas.x = t/**60**

self.\_T1\_meas.y = T[:, **0**] - **273.15**

self.\_PT1.value = np.round(T[-**1**, **0**]-**273.15**, **1**)

self.\_wT1.value = np.round(T[-**1**, **0**]-**273.15**, **1**)

self.\_T2\_meas.x = t/**60**

self.\_T2\_meas.y = T[:, **1**] - **273.15**

self.\_PT2.value = np.round(T[-**1**, **1**]-**273.15**, **1**)

self.\_wT2.value = np.round(T[-**1**, **1**]-**273.15**, **1**)

self.\_u1.x = t/**60**

self.\_u1.y = Q1

self.\_u2.x = t/**60**

self.\_u2.y = Q2

time.sleep(self.\_sleep)

a.Q1(**0**)

a.Q2(**0**)

a.close()

###########################################################################

# THREADING FUNCTION - ON-OFF

###########################################################################

**def** **\_work\_on\_off**(self):

**try**:

a = iTCLab()

**except**:

a.close()

a = iTCLab()

# Parater to start each cycle

self.\_Tc0 = np.array([

a.T1 + **273.15**,

a.T2 + **273.15**

])

# arrays to store data

t = np.array([])

Q1 = np.array([])

Q2 = np.array([])

T = np.array([[]]).reshape((**0**, **2**))

SP\_T1 = np.array([])

SP\_T2 = np.array([])

t = np.append(t, np.array([**0**]), axis=**0**)

T = np.append(T, self.\_Tc0.reshape((**1**, **2**)), axis=**0**)

Q1 = np.append(Q1, np.array([self.\_Q10]), axis=**0**)

Q2 = np.append(Q2, np.array([self.\_Q20]), axis=**0**)

SP\_T1 = np.append(SP\_T1, np.array([self.\_T1\_SP]), axis=**0**)

SP\_T2 = np.append(SP\_T2, np.array([self.\_T2\_SP]), axis=**0**)

# Main Loop

start\_time = time.time()

prev\_time = start\_time

**while** self.\_flag:

# Sleep time

sleep\_max = self.\_delta\_t

sleep = sleep\_max - (time.time() - prev\_time)

**if** sleep >= **0.01**:

time.sleep(sleep-**0.01**)

**else**:

time.sleep(**0.01**)

# Read temperatures in Celsius

self.\_Tc0 = np.array([

a.T1 + **273.15**,

a.T2 + **273.15**

])

# apply ON/OFF controller

# heater 1

**if** (self.\_Tc0[**0**]-**273.15**) < (self.\_T1\_SP - self.\_q1\_dt\_on\_off):

self.\_Q10 = **100.0**

**elif** (self.\_Tc0[**0**]-**273.15**) > (self.\_T1\_SP + self.\_q1\_dt\_on\_off):

self.\_Q10 = **0.0**

# heater 2

**if** (self.\_Tc0[**1**]-**273.15**) < (self.\_T2\_SP - self.\_q2\_dt\_on\_off):

self.\_Q20 = **100.0**

**elif** (self.\_Tc0[**1**]-**273.15**) > (self.\_T2\_SP + self.\_q2\_dt\_on\_off):

self.\_Q20 = **0.0**

# Write new heater values (0-100)

a.Q1(self.\_Q10)

a.Q2(self.\_Q20)

**if** len(t) >= self.\_maxtime:

t = np.delete(t, **0**, **0**)

T = np.delete(T, **0**, **0**)

Q1 = np.delete(Q1, **0**, **0**)

Q2 = np.delete(Q2, **0**, **0**)

SP\_T1 = np.delete(SP\_T1, **0**, **0**)

SP\_T2 = np.delete(SP\_T2, **0**, **0**)

# Record time and change in time

tm = time.time()

prev\_time = tm

tm = tm - start\_time

t = np.append(t, np.array([tm]), axis=**0**)

T = np.append(T, self.\_Tc0.reshape((**1**, **2**)), axis=**0**)

Q1 = np.append(Q1, np.array([self.\_Q10]), axis=**0**)

Q2 = np.append(Q2, np.array([self.\_Q20]), axis=**0**)

SP\_T1 = np.append(SP\_T1, np.array([self.\_T1\_SP]), axis=**0**)

SP\_T2 = np.append(SP\_T2, np.array([self.\_T2\_SP]), axis=**0**)

self.\_T1\_meas.x = t/**60**

self.\_T1\_meas.y = T[:, **0**] - **273.15**

self.\_PT1.value = np.round(T[-**1**, **0**]-**273.15**, **1**)

self.\_T1\_set\_point.x = t/**60**

self.\_T1\_set\_point.y = SP\_T1

self.\_T2\_meas.x = t/**60**

self.\_T2\_meas.y = T[:, **1**] - **273.15**

self.\_PT2.value = np.round(T[-**1**, **1**]-**273.15**, **1**)

self.\_T2\_set\_point.x = t/**60**

self.\_T2\_set\_point.y = SP\_T2

self.\_u1.x = t/**60**

self.\_u1.y = Q1

self.\_wQ1.value = np.round(Q1[-**1**], **1**)

self.\_u2.x = t/**60**

self.\_u2.y = Q2

self.\_wQ2.value = np.round(Q2[-**1**], **1**)

time.sleep(self.\_sleep)

a.Q1(**0**)

a.Q2(**0**)

a.close()

###########################################################################

# THREADING FUNCTION - PID

###########################################################################

**def** **\_work\_pid**(self):

**try**:

a = iTCLab()

**except**:

a.close()

a = iTCLab()

# Parater to start each cycle

self.\_Tc0 = np.array([

a.T1 + **273.15**,

a.T2 + **273.15**

])

# arrays to store data

t = np.array([])

Q1 = np.array([])

Q2 = np.array([])

T = np.array([[]]).reshape((**0**, **2**))

SP\_T1 = np.array([])

SP\_T2 = np.array([])

t = np.append(t, np.array([**0**]), axis=**0**)

T = np.append(T, self.\_Tc0.reshape((**1**, **2**)), axis=**0**)

Q1 = np.append(Q1, np.array([self.\_Q10]), axis=**0**)

Q2 = np.append(Q2, np.array([self.\_Q20]), axis=**0**)

SP\_T1 = np.append(SP\_T1, np.array([self.\_T1\_SP]), axis=**0**)

SP\_T2 = np.append(SP\_T2, np.array([self.\_T2\_SP]), axis=**0**)

# Integral error

ierr1 = **0.0**

ierr2 = **0.0**

# Main Loop

start\_time = time.time()

prev\_time = start\_time

**while** self.\_flag:

# Sleep time

sleep\_max = self.\_delta\_t

sleep = sleep\_max - (time.time() - prev\_time)

**if** sleep >= **0.01**:

time.sleep(sleep-**0.01**)

**else**:

time.sleep(**0.01**)

# Read temperatures in Celsius

self.\_Tc0 = np.array([

a.T1 + **273.15**,

a.T2 + **273.15**

])

**if** len(t) >= self.\_maxtime:

t = np.delete(t, **0**, **0**)

T = np.delete(T, **0**, **0**)

Q1 = np.delete(Q1, **0**, **0**)

Q2 = np.delete(Q2, **0**, **0**)

SP\_T1 = np.delete(SP\_T1, **0**, **0**)

SP\_T2 = np.delete(SP\_T2, **0**, **0**)

# Record time and change in time

tm = time.time()

prev\_time = tm

tm = tm - start\_time

t = np.append(t, np.array([tm]), axis=**0**)

T = np.append(T, self.\_Tc0.reshape((**1**, **2**)), axis=**0**)

Q1 = np.append(Q1, np.array([self.\_Q10]), axis=**0**)

Q2 = np.append(Q2, np.array([self.\_Q20]), axis=**0**)

SP\_T1 = np.append(SP\_T1, np.array([self.\_T1\_SP]), axis=**0**)

SP\_T2 = np.append(SP\_T2, np.array([self.\_T2\_SP]), axis=**0**)

# Calculate PID output

[self.\_Q10, ierr1] = self.\_PID(

self.\_T1\_SP, T[-**1**, **0**]-**273.15**, T[-**2**, **0**]-**273.15**, ierr1,

self.\_delta\_t, self.\_pid1\_gain, self.\_pid1\_reset,

self.\_pid1\_rate)

[self.\_Q20, ierr2] = self.\_PID(

self.\_T2\_SP, T[-**1**, **1**]-**273.15**, T[-**2**, **1**]-**273.15**, ierr2,

self.\_delta\_t, self.\_pid2\_gain, self.\_pid2\_reset,

self.\_pid2\_rate)

# Write new heater values (0-100)

a.Q1(self.\_Q10)

a.Q2(self.\_Q20)

self.\_T1\_meas.x = t/**60**

self.\_T1\_meas.y = T[:, **0**] - **273.15**

self.\_PT1.value = np.round(T[-**1**, **0**]-**273.15**, **1**)

self.\_T1\_set\_point.x = t/**60**

self.\_T1\_set\_point.y = SP\_T1

self.\_T2\_meas.x = t/**60**

self.\_T2\_meas.y = T[:, **1**] - **273.15**

self.\_PT2.value = np.round(T[-**1**, **1**]-**273.15**, **1**)

self.\_T2\_set\_point.x = t/**60**

self.\_T2\_set\_point.y = SP\_T2

self.\_u1.x = t/**60**

self.\_u1.y = Q1

self.\_wQ1.value = np.round(Q1[-**1**], **1**)

self.\_u2.x = t/**60**

self.\_u2.y = Q2

self.\_wQ2.value = np.round(Q2[-**1**], **1**)

time.sleep(self.\_sleep)

a.Q1(**0**)

a.Q2(**0**)

a.close()

###########################################################################

# THREADING FUNCTION - MPC

###########################################################################

**def** **\_work\_mpc**(self):

**try**:

a = iTCLab()

**except**:

a.close()

a = iTCLab()

# Parater to start each cycle

self.\_Tc0 = np.array([

a.T1 + **273.15**,

a.T2 + **273.15**

])

# arrays to store data

t = np.array([])

Q1 = np.array([])

Q2 = np.array([])

T = np.array([[]]).reshape((**0**, **2**))

SP\_T1 = np.array([])

SP\_T2 = np.array([])

t = np.append(t, np.array([**0**]), axis=**0**)

T = np.append(T, self.\_Tc0.reshape((**1**, **2**)), axis=**0**)

Q1 = np.append(Q1, np.array([self.\_Q10]), axis=**0**)

Q2 = np.append(Q2, np.array([self.\_Q20]), axis=**0**)

SP\_T1 = np.append(SP\_T1, np.array([self.\_T1\_SP]), axis=**0**)

SP\_T2 = np.append(SP\_T2, np.array([self.\_T2\_SP]), axis=**0**)

# Create MPC object

m = self.\_MPC()

# Main Loop

start\_time = time.time()

prev\_time = start\_time

**while** self.\_flag:

# Sleep time

sleep\_max = self.\_delta\_t

sleep = sleep\_max - (time.time() - prev\_time)

**if** sleep >= **0.01**:

time.sleep(sleep-**0.01**)

**else**:

time.sleep(**0.01**)

# Read temperatures in Celsius

self.\_Tc0 = np.array([

a.T1 + **273.15**,

a.T2 + **273.15**

])

**if** len(t) >= self.\_maxtime:

t = np.delete(t, **0**, **0**)

T = np.delete(T, **0**, **0**)

Q1 = np.delete(Q1, **0**, **0**)

Q2 = np.delete(Q2, **0**, **0**)

SP\_T1 = np.delete(SP\_T1, **0**, **0**)

SP\_T2 = np.delete(SP\_T2, **0**, **0**)

# Record time and change in time

tm = time.time()

prev\_time = tm

tm = tm - start\_time

t = np.append(t, np.array([tm]), axis=**0**)

T = np.append(T, self.\_Tc0.reshape((**1**, **2**)), axis=**0**)

Q1 = np.append(Q1, np.array([self.\_Q10]), axis=**0**)

Q2 = np.append(Q2, np.array([self.\_Q20]), axis=**0**)

SP\_T1 = np.append(SP\_T1, np.array([self.\_T1\_SP]), axis=**0**)

SP\_T2 = np.append(SP\_T2, np.array([self.\_T2\_SP]), axis=**0**)

# Change SOLVER

**if** self.\_SOLVER == '1 - APOPT':

m.options.SOLVER = **1**

**elif** self.\_SOLVER == '2 - BPOPT':

m.options.SOLVER = **2**

**else**:

m.options.SOLVER = **3**

# Change CVTYPE

**if** self.\_CVTYPE == '1 - Deadband':

m.options.CV\_TYPE = **1**

**else**:

m.options.CV\_TYPE = **2**

# Add measurements to the MPC

m.TC1.MEAS = self.\_Tc0[**0**] - **273.15**

m.TC2.MEAS = self.\_Tc0[**1**] - **273.15**

# Update Parameters

m.TC1.TAU = self.\_T1\_tau

m.TC2.TAU = self.\_T2\_tau

m.Q1.DMAX = self.\_Q1\_DMAX

m.Q1.DCOST = self.\_Q1\_DCOST

m.Q2.DMAX = self.\_Q2\_DMAX

m.Q2.DCOST = self.\_Q2\_DCOST

# Update prediction horizon

DT = self.\_delta\_t

m.time = [

**0**,

DT,

DT\***2**,

DT\***3**,

DT\***4**,

DT\***5**,

DT\***6**,

DT\***7**,

DT\***8**,

DT\***10**,

DT\***12**,

DT\***15**,

DT\***18**,

DT\***20**,

DT\***25**]

**if** m.options.CV\_TYPE == **1**:

# Input setpoint with deadband +/- DT

DT1 = self.\_T1\_dt

m.TC1.SPHI = self.\_T1\_SP + DT1

m.TC1.SPLO = self.\_T1\_SP - DT1

DT2 = self.\_T2\_dt

m.TC2.SPHI = self.\_T2\_SP + DT2

m.TC2.SPLO = self.\_T2\_SP - DT2

**else**:

m.TC1.SP = self.\_T1\_SP

m.TC2.SP = self.\_T2\_SP

**try**:

# Solve MPC

m.solve(disp=False)

# Check if successful solution

**if** (m.options.APPSTATUS == **1**):

# retrieve new value

self.\_Q10 = m.Q1.NEWVAL

self.\_Q20 = m.Q2.NEWVAL

**except**:

# Keep previous value

**pass**

# Write new heater values (0-100)

a.Q1(self.\_Q10)

a.Q2(self.\_Q20)

self.\_T1\_meas.x = t/**60**

self.\_T1\_meas.y = T[:, **0**] - **273.15**

self.\_PT1.value = np.round(T[-**1**, **0**]-**273.15**, **1**)

self.\_T1\_set\_point.x = t/**60**

self.\_T1\_set\_point.y = SP\_T1

self.\_T2\_meas.x = t/**60**

self.\_T2\_meas.y = T[:, **1**] - **273.15**

self.\_PT2.value = np.round(T[-**1**, **1**]-**273.15**, **1**)

self.\_T2\_set\_point.x = t/**60**

self.\_T2\_set\_point.y = SP\_T2

self.\_u1.x = t/**60**

self.\_u1.y = Q1

self.\_wQ1.value = np.round(Q1[-**1**], **1**)

self.\_u2.x = t/**60**

self.\_u2.y = Q2

self.\_wQ2.value = np.round(Q2[-**1**], **1**)

time.sleep(self.\_sleep)

a.Q1(**0**)

a.Q2(**0**)

a.close()

**Code control\_demo.py**

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Wed Jan 23 19:02:13 2019

@author: evertoncolling

@email: evertoncolling@gmail.com

@licence: MIT

"""

**from** **\_\_future\_\_** **import** print\_function, division

**from** **ipywidgets** **import** widgets **as** wi

**from** **IPython.display** **import** display

**import** **threading**

**import** **time**

**import** **numpy** **as** **np**

**from** **gekko** **import** GEKKO

**from** **scipy.integrate** **import** odeint

**import** **bqplot** **as** **bq**

**class** **GUI**(object):

"""

Class that defines the \_GUI applications

"""

**def** **\_\_init\_\_**(self):

"""

Initialize the \_GUI elements

"""

#######################################################################

# PLOTTING CONFIGURATION

#######################################################################

colors = [

'#1f77b4', # T1

'#5ebdff', # T1\_SP

'#d62728', # T2

'#ff7f0e', # T2\_SP

'#1f77b4', # Q1

'#d62728' # Q2

]

# Create scales

t\_sc = bq.LinearScale()

T\_sc = bq.LinearScale(min=**20.**, max=**60.**)

Q\_sc = bq.LinearScale(min=**0.**, max=**100.**)

T\_sc\_set = {'x': t\_sc, 'y': T\_sc}

Q\_sc\_set = {'x': t\_sc, 'y': Q\_sc}

# Create Axis

ax\_x1 = bq.Axis(label="", scale=t\_sc, visible=False) # upper

ax\_x2 = bq.Axis(label="time [min]", scale=t\_sc) # down

ax\_T1 = bq.Axis(label="Temperature 1 [C]", scale=T\_sc,

orientation="vertical", label\_color=colors[**0**])

ax\_T2 = bq.Axis(label="Temperature 2 [C]", scale=T\_sc,

orientation="vertical", label\_color=colors[**2**])

ax\_Q = bq.Axis(label="Heater Output [%]", scale=Q\_sc,

orientation="vertical", label\_color="black")

# Create Lines/Markers

self.\_T1\_meas = bq.Scatter(x=[], y=[], scales=T\_sc\_set,

marker='circle', colors=[colors[**0**]],

display\_legend=False, default\_size=**20**)

self.\_T2\_meas = bq.Scatter(x=[], y=[], scales=T\_sc\_set,

marker='circle', colors=[colors[**2**]],

display\_legend=False, default\_size=**20**)

self.\_T1\_set\_point = bq.Lines(x=[], y=[], scales=T\_sc\_set,

stroke\_width=**4**, colors=[colors[**1**]],

interpolation='step-before',

display\_legend=False)

self.\_T2\_set\_point = bq.Lines(x=[], y=[], scales=T\_sc\_set,

stroke\_width=**4**, colors=[colors[**3**]],

interpolation='step-before',

display\_legend=False)

self.\_u1 = bq.Lines(x=[], y=[], scales=Q\_sc\_set,

stroke\_width=**2**, colors=[colors[**4**]],

interpolation='step-before',

display\_legend=True, labels=['Heater 1'])

self.\_u2 = bq.Lines(x=[], y=[], scales=Q\_sc\_set,

stroke\_width=**2**, colors=[colors[**5**]],

interpolation='step-before',

display\_legend=True, labels=['Heater 2'])

# Mix everything and create figures

fig\_lay\_up = wi.Layout(width="400px", height="240px")

fig\_lay\_down = wi.Layout(width="400px", height="290px")

box1 = dict(top=**10**, bottom=**0**, left=**60**, right=**5**)

box2 = dict(top=**10**, bottom=**0**, left=**60**, right=**20**)

box3 = dict(top=**10**, bottom=**50**, left=**60**, right=**10**)

fig1 = bq.Figure(layout=fig\_lay\_up, axes=[ax\_x1, ax\_T1],

marks=[self.\_T1\_meas, self.\_T1\_set\_point],

fig\_margin=box1)

fig2 = bq.Figure(layout=fig\_lay\_up, axes=[ax\_x1, ax\_T2],

marks=[self.\_T2\_meas, self.\_T2\_set\_point],

fig\_margin=box2)

fig3 = bq.Figure(layout=fig\_lay\_down, axes=[ax\_x2, ax\_Q],

marks=[self.\_u1, self.\_u2], fig\_margin=box3,

legend\_location="left",

legend\_style={"fill": "white",

"fill-opacity": "0.7",

"width": "130px"})

#######################################################################

# SPACING WIDGETS CRIATION

#######################################################################

v\_space = wi.Label(value="", layout=wi.Layout(height='2px'))

h\_space = wi.Label(value="", layout=wi.Layout(width='2px'))

#######################################################################

# VARYING ARRAYS TO STORE DATA

#######################################################################

self.\_t = np.array([])

self.\_Q1 = np.array([])

self.\_Q2 = np.array([])

self.\_T = np.array([[]]).reshape((**0**, **2**))

self.\_SP\_T1 = np.array([])

self.\_SP\_T2 = np.array([])

#######################################################################

# PARAMETERS

#######################################################################

self.\_delta\_t = **4.0**

self.\_maxtime = int(**500**/self.\_delta\_t)

self.\_Th0 = np.array([**293.15**, **293.15**])

self.\_Tc0 = np.array([**293.15**, **293.15**])

self.\_T1\_SP = **30**

self.\_T2\_SP = **30**

self.\_Q10 = **0**

self.\_Q20 = **0**

self.\_flag = False

self.\_sleep = **0.5**

self.\_q1\_dt\_on\_off = **0.1**

self.\_q2\_dt\_on\_off = **0.1**

self.\_pid1\_gain = **10.**

self.\_pid1\_reset = **50.**

self.\_pid1\_rate = **1.**

self.\_pid2\_gain = **10.**

self.\_pid2\_reset = **50.**

self.\_pid2\_rate = **1.**

self.\_SOLVER = '1 - APOPT'

self.\_CVTYPE = '1 - Deadband'

self.\_T1\_dt = **0.1**

self.\_T1\_tau = **10.**

self.\_T2\_dt = **0.1**

self.\_T2\_tau = **10.**

self.\_Q1\_DMAX = **30.**

self.\_Q1\_DCOST = **1.**

self.\_Q2\_DMAX = **30.**

self.\_Q2\_DCOST = **1.**

#######################################################################

# OUTPUT WIDGETS CRIATION

#######################################################################

style = {'description\_width': 'initial'}

self.\_PT1 = wi.FloatProgress(value=self.\_Tc0[**0**]-**273.15**, min=**0**,

max=**100.0**,

description='<b>PV (°C):</b>',

bar\_style='warning',

orientation='horizontal',

style=style)

self.\_LT1 = wi.Label(value=str(self.\_PT1.value))

wi.jslink((self.\_PT1, 'value'), (self.\_LT1, 'value'))

self.\_PT2 = wi.FloatProgress(value=self.\_Tc0[**1**]-**273.15**, min=**0**,

max=**100.0**,

description='<b>PV (°C):</b>',

bar\_style='warning',

orientation='horizontal',

style=style)

self.\_LT2 = wi.Label(value=str(self.\_PT2.value))

wi.jslink((self.\_PT2, 'value'), (self.\_LT2, 'value'))

#######################################################################

# INTERACTION WIDGETS CREATION

#######################################################################

self.\_wQ1 = wi.FloatSlider(value=self.\_Q10, min=**0**, max=**100.0**, step=**0.5**,

description='<b>Q1 (%):</b>',

continuous\_update=False,

orientation='horizontal',

readout=False, style=style,

layout=wi.Layout(width='230px'))

self.\_tQ1 = wi.BoundedFloatText(value=self.\_Q10, min=**0**, max=**100.0**,

step=**0.5**,

layout=wi.Layout(width='60px'))

wi.jslink((self.\_wQ1, 'value'), (self.\_tQ1, 'value'))

self.\_bQ1 = wi.Button(description='Set',

layout=wi.Layout(width='50px'))

self.\_bQ1.on\_click(self.\_Q1\_click)

self.\_wQ2 = wi.FloatSlider(value=self.\_Q20, min=**0**, max=**100.0**, step=**0.5**,

description='<b>Q2 (%):</b>',

continuous\_update=False,

orientation='horizontal',

readout=False, style=style,

layout=wi.Layout(width='230px'))

self.\_tQ2 = wi.BoundedFloatText(value=self.\_Q20, min=**0**, max=**100.0**,

step=**0.5**,

layout=wi.Layout(width='60px'))

wi.jslink((self.\_wQ2, 'value'), (self.\_tQ2, 'value'))

self.\_bQ2 = wi.Button(description='Set',

layout=wi.Layout(width='50px'))

self.\_bQ2.on\_click(self.\_Q2\_click)

self.\_wT1 = wi.FloatSlider(value=self.\_T1\_SP, min=**20**, max=**60.0**,

step=**0.5**, description='<b>T1 SP:</b>',

continuous\_update=False,

orientation='horizontal',

readout=False, style=style,

disabled=True,

layout=wi.Layout(width='230px'))

self.\_tT1 = wi.BoundedFloatText(value=self.\_T1\_SP, min=**20**, max=**60.0**,

step=**0.5**, disabled=True,

layout=wi.Layout(width='60px'))

wi.jslink((self.\_wT1, 'value'), (self.\_tT1, 'value'))

self.\_bT1 = wi.Button(description='Set',

layout=wi.Layout(width='50px'),

disabled=True)

self.\_bT1.on\_click(self.\_T1\_click)

self.\_wT2 = wi.FloatSlider(value=self.\_T2\_SP, min=**20**, max=**60.0**,

step=**0.5**, description='<b>T2 SP:</b>',

continuous\_update=False,

orientation='horizontal', disabled=True,

readout=False, style=style,

layout=wi.Layout(width='230px'))

self.\_tT2 = wi.BoundedFloatText(value=self.\_T2\_SP, min=**20**, max=**60.0**,

step=**0.5**, disabled=True,

layout=wi.Layout(width='60px'))

wi.jslink((self.\_wT2, 'value'), (self.\_tT2, 'value'))

self.\_bT2 = wi.Button(description='Set',

layout=wi.Layout(width='50px'), disabled=True)

self.\_bT2.on\_click(self.\_T2\_click)

#######################################################################

# MODE SELECTION

#######################################################################

self.\_mode = wi.ToggleButtons(options=['Manual',

'On-Off',

'PID',

'MPC'],

style={'button\_width': '100px'})

self.\_mode.observe(self.\_mode\_switch, names='value')

#######################################################################

# STOP THREAD

#######################################################################

self.\_b\_stop = wi.Button(description='Stop', button\_style='warning',

icon='stop', layout=wi.Layout(width='100px',

height='32px'))

self.\_b\_stop.on\_click(self.\_stop\_click)

#######################################################################

# START THREAD - OPEN LOOP

#######################################################################

self.\_b\_play = wi.Button(description='Start', button\_style='success',

icon='play', layout=wi.Layout(width='100px',

height='32px'))

self.\_b\_play.on\_click(self.\_play\_click)

# Join Buttons

buttons = wi.HBox((self.\_b\_play, h\_space, self.\_b\_stop,

wi.Label(value="", layout=wi.Layout(width='165px')),

self.\_mode))

#######################################################################

# LAYOUT

#######################################################################

Q1\_Set = wi.HBox((h\_space, self.\_wQ1, self.\_tQ1, self.\_bQ1),

layout=wi.Layout(border='solid 2px gray',

width='360px'))

T1\_Set = wi.HBox((h\_space, self.\_wT1, self.\_tT1, self.\_bT1),

layout=wi.Layout(border='solid 2px gray',

width='360px'))

T1\_View = wi.HBox((h\_space, self.\_PT1, self.\_LT1),

layout=wi.Layout(border='solid 2px gray',

width='360px'))

Q2\_Set = wi.HBox((h\_space, self.\_wQ2, self.\_tQ2, self.\_bQ2),

layout=wi.Layout(border='solid 2px gray',

width='360px'))

T2\_Set = wi.HBox((h\_space, self.\_wT2, self.\_tT2, self.\_bT2),

layout=wi.Layout(border='solid 2px gray',

width='360px'))

T2\_View = wi.HBox((h\_space, self.\_PT2, self.\_LT2),

layout=wi.Layout(border='solid 2px gray',

width='360px'))

co1 = wi.VBox((T1\_Set, v\_space, T1\_View, v\_space, Q1\_Set))

co2 = wi.VBox((T2\_Set, v\_space, T2\_View, v\_space, Q2\_Set))

co = wi.VBox((v\_space, v\_space, co1, v\_space, v\_space, v\_space, co2))

fig1x = wi.HBox((fig1, fig2))

fig2x = wi.HBox((fig3, h\_space, h\_space, co))

figy = wi.VBox((fig1x, v\_space, fig2x),

layout=wi.Layout(border='solid 2px gray',

width='800px'))

#######################################################################

# APPLICATION TO DISPLAY

#######################################################################

self.\_gui = wi.VBox((buttons, v\_space, figy))

#######################################################################

# CONFIGURATOR

#######################################################################

style = {'description\_width': 'initial'}

lay = wi.Layout(width='120px', margin='0 20px 0 0')

#######################################################################

# GENERAL OPTIONS

#######################################################################

self.\_conf11 = wi.HBox((

wi.HTML(

value='<p style="text-align: right;"><b>&Delta;t (s):</b></p>',

layout=lay),

wi.FloatSlider(value=**4.0**, min=**1.0**, max=**10.0**, step=**0.5**,

description='', style=style)))

self.\_conf12 = wi.HBox((

wi.HTML(value='<p style="text-align: right;">'

'<b>Sleep time (s):</b></p>',

layout=lay),

wi.FloatSlider(value=**0.5**, min=**0.25**, max=**1.5**, step=**0.05**,

description='', style=style)))

but11 = wi.Button(description='Apply', icon='check',

layout=wi.Layout(width='100px', height='32px'))

but11.on\_click(self.\_conf\_general)

but12 = wi.Button(description='Reset', icon='refresh',

layout=wi.Layout(width='100px', height='32px'))

but12.on\_click(self.\_reset\_general)

conf13 = wi.HBox((but11, but12), layout=wi.Layout(margin='10px 0 0 0'))

#######################################################################

# ON-OFF OPTIONS

#######################################################################

lay5 = wi.Layout(width='90px', margin='0 10px 0 15px')

conf20 = wi.Button(description='TEMPERATURE 01 / HEATER 01',

disabled=True,

layout=wi.Layout(width='320px', margin='0 0 0 0'))

self.\_conf21 = wi.HBox((

wi.HTML(value='<p style="text-align: right;">'

'<b>Deadband (K):</b></p>',

layout=lay5),

wi.FloatSlider(value=**0.1**, min=**0.0**, max=**2.0**, step=**0.1**,

description='', style=style,

layout=wi.Layout(width='200px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

box21 = wi.VBox((conf20, self.\_conf21),

layout=wi.Layout(border='solid 2px gray',

width='325px',

margin='10px 0 0 0px'))

conf22 = wi.Button(description='TEMPERATURE 02 / HEATER 02',

disabled=True,

layout=wi.Layout(width='320px', margin='0 0 0 0'))

self.\_conf23 = wi.HBox((

wi.HTML(value='<p style="text-align: right;">'

'<b>Deadband (K):</b></p>',

layout=lay5),

wi.FloatSlider(value=**0.1**, min=**0.0**, max=**2.0**, step=**0.1**,

description='', style=style,

layout=wi.Layout(width='200px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

box22 = wi.VBox((conf22, self.\_conf23),

layout=wi.Layout(border='solid 2px gray',

width='325px',

margin='10px 0 0 10px'))

but21 = wi.Button(description='Apply', icon='check',

layout=wi.Layout(width='100px', height='32px'))

but21.on\_click(self.\_conf\_on\_off)

but22 = wi.Button(description='Reset', icon='refresh',

layout=wi.Layout(width='100px', height='32px'))

but22.on\_click(self.\_reset\_on\_off)

conf24 = wi.HBox((but21, but22), layout=wi.Layout(margin='10px 0 0 0'))

#######################################################################

# PID OPTIONS

#######################################################################

lay4 = wi.Layout(width='100px', margin='0 10px 0 15px')

conf30 = wi.Button(description='TEMPERATURE 01 / HEATER 01',

disabled=True,

layout=wi.Layout(width='330px', margin='0 0 0 0'))

self.\_conf31 = wi.HBox((

wi.HTML(value='<p style="text-align: right;">'

'<b>Kc (K/%Heater):</b></p>',

layout=lay4),

wi.FloatSlider(value=**10.0**, min=**0.0**, max=**20.0**, step=**0.5**,

description='', style=style,

layout=wi.Layout(width='200px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

self.\_conf32 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>tauI (s):</b></p>',

layout=lay4),

wi.FloatSlider(value=**50.0**, min=**0.0**, max=**200.0**, step=**1.0**,

description='', style=style,

layout=wi.Layout(width='200px'))))

self.\_conf33 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>tauD (s):</b></p>',

layout=lay4),

wi.FloatSlider(value=**1.0**, min=**0.0**, max=**10.0**, step=**0.5**,

description='', style=style,

layout=wi.Layout(width='200px'))))

box31 = wi.VBox((conf30, self.\_conf31, self.\_conf32, self.\_conf33),

layout=wi.Layout(border='solid 2px gray',

width='335px',

margin='10px 0 0 0px'))

conf34 = wi.Button(description='TEMPERATURE 02 / HEATER 02',

disabled=True,

layout=wi.Layout(width='330px', margin='0 0 0 0'))

self.\_conf35 = wi.HBox((

wi.HTML(value='<p style="text-align: right;">'

'<b>Kc (K/%Heater):</b></p>',

layout=lay4),

wi.FloatSlider(value=**10.0**, min=**0.0**, max=**20.0**, step=**0.5**,

description='', style=style,

layout=wi.Layout(width='200px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

self.\_conf36 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>tauI (s):</b></p>',

layout=lay4),

wi.FloatSlider(value=**50.0**, min=**0.0**, max=**200.0**, step=**1.0**,

description='', style=style,

layout=wi.Layout(width='200px'))))

self.\_conf37 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>tauD (s):</b></p>',

layout=lay4),

wi.FloatSlider(value=**1.0**, min=**0.0**, max=**10.0**, step=**0.5**,

description='', style=style,

layout=wi.Layout(width='200px'))))

box32 = wi.VBox((conf34, self.\_conf35, self.\_conf36, self.\_conf37),

layout=wi.Layout(border='solid 2px gray',

width='335px',

margin='10px 0 0 10px'))

but31 = wi.Button(description='Apply', icon='check',

layout=wi.Layout(width='100px', height='32px'))

but31.on\_click(self.\_conf\_pid)

but32 = wi.Button(description='Reset', icon='refresh',

layout=wi.Layout(width='100px', height='32px'))

but32.on\_click(self.\_reset\_pid)

conf38 = wi.HBox((but31, but32), layout=wi.Layout(margin='10px 0 0 0'))

#######################################################################

# MPC OPTIONS

#######################################################################

lay1 = wi.Layout(width='60px', margin='0 10px 0 25px')

lay2 = wi.Layout(width='95px', margin='0 20px 0 10px')

lay3 = wi.Layout(width='55px', margin='0 20px 0 10px')

self.\_conf40 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>SOLVER:</b></p>',

layout=lay1),

wi.Dropdown(options=['1 - APOPT', '2 - BPOPT', '3 - IPOPT'],

value='1 - APOPT',

layout=wi.Layout(width='100px')),

wi.HTML(value='<p style="text-align: right;"><b>CVTYPE:</b></p>',

layout=lay1),

wi.Dropdown(options=['1 - Deadband', '2 - Trajectory'],

value='1 - Deadband',

layout=wi.Layout(width='130px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

conf41 = wi.Button(description='TEMPERATURE 01',

disabled=True,

layout=wi.Layout(width='360px', margin='0 0 0 0'))

self.\_conf42 = wi.HBox((

wi.HTML(value='<p style="text-align: right;">'

'<b>Deadband (K):</b></p>',

layout=lay2),

wi.FloatSlider(value=**0.1**, min=**0.1**, max=**1.0**, step=**0.1**,

description='', style=style,

layout=wi.Layout(width='250px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

self.\_conf43 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>TAU:</b></p>',

layout=lay2),

wi.FloatSlider(value=**10.0**, min=**1.0**, max=**100.0**, step=**1.0**,

description='', style=style,

layout=wi.Layout(width='250px'))))

box41 = wi.VBox((conf41, self.\_conf42, self.\_conf43),

layout=wi.Layout(border='solid 2px gray',

width='365px',

margin='10px 0 0 0'))

conf44 = wi.Button(description='TEMPERATURE 02',

disabled=True,

layout=wi.Layout(width='360px', margin='0 0 0 0'))

self.\_conf45 = wi.HBox((

wi.HTML(value='<p style="text-align: right;">'

'<b>Deadband (K):</b></p>',

layout=lay2),

wi.FloatSlider(value=**0.1**, min=**0.1**, max=**1.0**, step=**0.1**,

description='', style=style,

layout=wi.Layout(width='250px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

self.\_conf46 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>TAU:</b></p>',

layout=lay2),

wi.FloatSlider(value=**10.0**, min=**1.0**, max=**100.0**, step=**1.0**,

description='', style=style,

layout=wi.Layout(width='250px'))))

box42 = wi.VBox((conf44, self.\_conf45, self.\_conf46),

layout=wi.Layout(border='solid 2px gray',

width='365px',

margin='10px 0 0 0'))

conf47 = wi.Button(description='HEATER 01',

disabled=True,

layout=wi.Layout(width='320px', margin='0 0 0 0'))

self.\_conf48 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>DMAX:</b></p>',

layout=lay3),

wi.FloatSlider(value=**30.0**, min=**1.0**, max=**100.0**, step=**0.5**,

description='', style=style,

layout=wi.Layout(width='250px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

self.\_conf49 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>DCOST:</b></p>',

layout=lay3),

wi.FloatSlider(value=**1.0**, min=**0.0**, max=**10.0**, step=**0.1**,

description='', style=style,

layout=wi.Layout(width='250px'))))

box43 = wi.VBox((conf47, self.\_conf48, self.\_conf49),

layout=wi.Layout(border='solid 2px gray',

width='325px',

margin='10px 0 0 10px'))

conf410 = wi.Button(description='HEATER 02',

disabled=True,

layout=wi.Layout(width='320px', margin='0 0 0 0'))

self.\_conf411 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>DMAX:</b></p>',

layout=lay3),

wi.FloatSlider(value=**30.0**, min=**1.0**, max=**100.0**, step=**0.5**,

description='', style=style,

layout=wi.Layout(width='250px'))),

layout=wi.Layout(margin='5px 0 0 0')

)

self.\_conf412 = wi.HBox((

wi.HTML(value='<p style="text-align: right;"><b>DCOST:</b></p>',

layout=lay3),

wi.FloatSlider(value=**1.0**, min=**0.0**, max=**10.0**, step=**0.1**,

description='', style=style,

layout=wi.Layout(width='250px'))))

box44 = wi.VBox((conf410, self.\_conf411, self.\_conf412),

layout=wi.Layout(border='solid 2px gray',

width='325px',

margin='10px 0 0 10px'))

but41 = wi.Button(description='Apply', icon='check',

layout=wi.Layout(width='100px', height='32px'))

but41.on\_click(self.\_conf\_mpc)

but42 = wi.Button(description='Reset', icon='refresh',

layout=wi.Layout(width='100px', height='32px'))

but42.on\_click(self.\_reset\_mpc)

conf413 = wi.HBox((but41, but42),

layout=wi.Layout(margin='10px 0 0 0'))

#######################################################################

# CONFIGURATOR LAYOUT

#######################################################################

tab = wi.Tab([wi.VBox((self.\_conf11, self.\_conf12,

wi.Label(layout=wi.Layout(height='206px')),

conf13)),

wi.VBox((wi.HBox((box21, box22)),

wi.Label(layout=wi.Layout(height='191px')),

conf24)),

wi.VBox((wi.HBox((box31, box32)),

wi.Label(layout=wi.Layout(height='127px')),

conf38)),

wi.VBox((self.\_conf40,

wi.HBox((box41, box43)),

wi.HBox((box42, box44)),

wi.Label(layout=wi.Layout(height='11px')),

conf413))],

layout=wi.Layout(width='800px', height='380px'))

tab.set\_title(**0**, 'General Options')

tab.set\_title(**1**, 'On-Off Options')

tab.set\_title(**2**, 'PID Options')

tab.set\_title(**3**, 'MPC Options')

#######################################################################

# DISPLAY CONFIGURATOR

#######################################################################

self.\_conf = tab

**def** **app**(self):

display(self.\_gui)

**def** **config**(self):

display(self.\_conf)

**def** **\_conf\_general**(self, b):

self.\_delta\_t = self.\_conf11.children[**1**].value

self.\_maxtime = int(**500**/self.\_delta\_t)

self.\_sleep = self.\_conf12.children[**1**].value

**def** **\_reset\_general**(self, b):

self.\_conf11.children[**1**].value = **4.0**

self.\_delta\_t = self.\_conf11.children[**1**].value

self.\_maxtime = int(**500**/self.\_delta\_t)

self.\_conf12.children[**1**].value = **0.5**

self.\_sleep = self.\_conf12.children[**1**].value

**def** **\_conf\_on\_off**(self, b):

self.\_q1\_dt\_on\_off = self.\_conf21.children[**1**].value

self.\_q2\_dt\_on\_off = self.\_conf23.children[**1**].value

**def** **\_reset\_on\_off**(self, b):

self.\_conf21.children[**1**].value = **0.1**

self.\_q1\_dt\_on\_off = self.\_conf21.children[**1**].value

self.\_conf23.children[**1**].value = **0.1**

self.\_q2\_dt\_on\_off = self.\_conf23.children[**1**].value

**def** **\_conf\_pid**(self, b):

self.\_pid1\_gain = self.\_conf31.children[**1**].value

self.\_pid1\_reset = self.\_conf32.children[**1**].value

self.\_pid1\_rate = self.\_conf33.children[**1**].value

self.\_pid2\_gain = self.\_conf35.children[**1**].value

self.\_pid2\_reset = self.\_conf36.children[**1**].value

self.\_pid2\_rate = self.\_conf37.children[**1**].value

**def** **\_reset\_pid**(self, b):

self.\_conf31.children[**1**].value = **10.0** # gain

self.\_conf32.children[**1**].value = **50.0** # reset

self.\_conf33.children[**1**].value = **1.0** # rate

self.\_pid1\_gain = self.\_conf31.children[**1**].value

self.\_pid1\_reset = self.\_conf32.children[**1**].value

self.\_pid1\_rate = self.\_conf33.children[**1**].value

self.\_conf35.children[**1**].value = **10.0** # gain

self.\_conf36.children[**1**].value = **50.0** # reset

self.\_conf37.children[**1**].value = **1.0** # rate

self.\_pid2\_gain = self.\_conf35.children[**1**].value

self.\_pid2\_reset = self.\_conf36.children[**1**].value

self.\_pid2\_rate = self.\_conf37.children[**1**].valu

**def** **\_conf\_mpc**(self, b):

self.\_SOLVER = self.\_conf40.children[**1**].value

self.\_CVTYPE = self.\_conf40.children[**3**].value

self.\_T1\_dt = self.\_conf42.children[**1**].value

self.\_T1\_tau = self.\_conf43.children[**1**].value

self.\_T2\_dt = self.\_conf45.children[**1**].value

self.\_T2\_tau = self.\_conf46.children[**1**].value

self.\_Q1\_DMAX = self.\_conf48.children[**1**].value

self.\_Q1\_DCOST = self.\_conf49.children[**1**].value

self.\_Q2\_DMAX = self.\_conf411.children[**1**].value

self.\_Q2\_DCOST = self.\_conf412.children[**1**].value

**def** **\_reset\_mpc**(self, b):

self.\_conf40.children[**1**].value = '1 - APOPT'

self.\_conf40.children[**3**].value = '1 - Deadband'

self.\_conf42.children[**1**].value = **0.1**

self.\_conf43.children[**1**].value = **30.**

self.\_conf45.children[**1**].value = **0.1**

self.\_conf46.children[**1**].value = **30.**

self.\_conf48.children[**1**].value = **30.**

self.\_conf49.children[**1**].value = **1.**

self.\_conf411.children[**1**].value = **30.**

self.\_conf412.children[**1**].value = **1.**

self.\_SOLVER = self.\_conf40.children[**1**].value

self.\_CVTYPE = self.\_conf40.children[**3**].value

self.\_T1\_dt = self.\_conf42.children[**1**].value

self.\_T1\_tau = self.\_conf43.children[**1**].value

self.\_T2\_dt = self.\_conf45.children[**1**].value

self.\_T2\_tau = self.\_conf46.children[**1**].value

self.\_Q1\_DMAX = self.\_conf48.children[**1**].value

self.\_Q1\_DCOST = self.\_conf49.children[**1**].value

self.\_Q2\_DMAX = self.\_conf411.children[**1**].value

self.\_Q2\_DCOST = self.\_conf412.children[**1**].value

**def** **\_Q1\_click**(self, b):

self.\_Q10 = self.\_wQ1.value

**def** **\_Q2\_click**(self, b):

self.\_Q20 = self.\_wQ2.value

**def** **\_T1\_click**(self, b):

self.\_T1\_SP = self.\_wT1.value

**def** **\_T2\_click**(self, b):

self.\_T2\_SP = self.\_wT2.value

**def** **\_stop\_click**(self, b):

self.\_flag = False

self.\_mode.disabled = False

**def** **\_play\_click**(self, b):

**if** **not** self.\_flag:

**if** self.\_mode.value == "Manual":

self.\_flag = True

self.\_mode.disabled = True

thread = threading.Thread(target=self.\_work\_man)

thread.start()

**elif** self.\_mode.value == "On-Off":

self.\_flag = True

self.\_mode.disabled = True

thread = threading.Thread(target=self.\_work\_on\_off)

thread.start()

**elif** self.\_mode.value == "PID":

self.\_flag = True

self.\_mode.disabled = True

thread = threading.Thread(target=self.\_work\_pid)

thread.start()

**elif** self.\_mode.value == "MPC":

self.\_flag = True

self.\_mode.disabled = True

thread = threading.Thread(target=self.\_work\_mpc)

thread.start()

**def** **\_mode\_switch**(self, value):

# Reinitialize parameters

self.\_Th0 = np.array([**293.15**, **293.15**])

self.\_Tc0 = np.array([**293.15**, **293.15**])

self.\_T1\_SP = **30**

self.\_T2\_SP = **30**

self.\_Q10 = **0**

self.\_Q20 = **0**

# Reset figures

self.\_T1\_meas.x = []

self.\_T1\_meas.y = []

self.\_T2\_meas.x = []

self.\_T2\_meas.y = []

self.\_T1\_set\_point.x = []

self.\_T1\_set\_point.y = []

self.\_T2\_set\_point.x = []

self.\_T2\_set\_point.y = []

self.\_u1.x = []

self.\_u1.y = []

self.\_u2.x = []

self.\_u2.y = []

# Reset controls

self.\_PT1.value = self.\_Tc0[**0**]-**273.15**

self.\_PT2.value = self.\_Tc0[**1**]-**273.15**

self.\_wT1.value = self.\_T1\_SP

self.\_wT2.value = self.\_T2\_SP

self.\_wQ1.value = self.\_Q10

self.\_wQ2.value = self.\_Q20

**if** value['new'] == "Manual":

self.\_wQ1.disabled = False

self.\_tQ1.disabled = False

self.\_bQ1.disabled = False

self.\_wT1.disabled = True

self.\_tT1.disabled = True

self.\_bT1.disabled = True

self.\_wQ2.disabled = False

self.\_tQ2.disabled = False

self.\_bQ2.disabled = False

self.\_wT2.disabled = True

self.\_tT2.disabled = True

self.\_bT2.disabled = True

**else**:

self.\_wQ1.disabled = True

self.\_tQ1.disabled = True

self.\_bQ1.disabled = True

self.\_wT1.disabled = False

self.\_tT1.disabled = False

self.\_bT1.disabled = False

self.\_wQ2.disabled = True

self.\_tQ2.disabled = True

self.\_bQ2.disabled = True

self.\_wT2.disabled = False

self.\_tT2.disabled = False

self.\_bT2.disabled = False

###########################################################################

# \_MODEL TO SIMULATE

###########################################################################

**def** **\_heater**(self, x, t, Q1, Q2):

# Parameters

U = **4.87519009** + (np.random.rand()-**0.5**) # variable convection

alpha1 = **0.00640897365**

alpha2 = **0.00310952441**

Ta = **23** + **273.15** # K

m = **4.0**/**1000.0** # kg

Cp = **0.5** \* **1000.0** # J/kg-K

A = **10.0** / **100.0**\*\***2** # Area in m^2

As = **2.0** / **100.0**\*\***2** # Area in m^2

eps = **0.9** # Emissivity

sigma = **5.67e-8** # Stefan-Boltzman

# Temperature States

Th1 = x[**0**]

Th2 = x[**1**]

# Heat Transfer Exchange Between 1 and 2

conv12 = U\*As\*(Th2-Th1)

rad12 = eps\*sigma\*As \* (Th2\*\***4** - Th1\*\***4**)

# Nonlinear Energy Balances

dTh1dt = (**1.0**/(m\*Cp)) \* \

(U\*A\*(Ta-Th1) +

eps \* sigma \* A \* (Ta\*\***4** - Th1\*\***4**) +

conv12 + rad12 + alpha1\*Q1)

dTh2dt = (**1.0**/(m\*Cp)) \* \

(U\*A\*(Ta-Th2) +

eps \* sigma \* A \* (Ta\*\***4** - Th2\*\***4**) -

conv12 - rad12 + alpha2\*Q2)

**return** [dTh1dt, dTh2dt]

**def** **\_sensor**(self, x, t, Th1, Th2):

# Parameter

tau = **17.7176964**

# Temperature States

Tc1 = x[**0**]

Tc2 = x[**1**]

# lag equations to emulate conduction

dTc1dt = (-Tc1 + Th1)/tau

dTc2dt = (-Tc2 + Th2)/tau

**return** [dTc1dt, dTc2dt]

###########################################################################

# PID CONTROLLER

###########################################################################

# inputs -----------------------------------

# sp = setpoint

# pv = current temperature

# pv\_last = prior temperature

# ierr = integral error

# dt = time increment between measurements

# outputs ----------------------------------

# op = output of the PID controller

# I = integral contribution

**def** **\_PID**(self, sp, pv, pv\_last, ierr, dt, Kc=**10.0**, tauI=**50.0**, tauD=**1.0**):

# Default Parameters

# Kc = 10.0 # K/%Heater

# tauI = 50.0 # sec

# tauD = 1.0 # sec

# Parameters in terms of PID coefficients

KP = Kc

**if** tauI == **0**:

KI = **1e5**

**else**:

KI = Kc/tauI

KD = Kc\*tauD

# ubias for controller (initial heater)

op0 = **0**

# upper and lower bounds on heater level

ophi = **100**

oplo = **0**

# calculate the error

error = sp-pv

# calculate the integral error

ierr = ierr + KI \* error \* dt

# calculate the measurement derivative

dpv = (pv - pv\_last) / dt

# calculate the PID output

P = KP \* error

I = ierr

D = -KD \* dpv

op = op0 + P + I + D

# implement anti-reset windup

**if** op < oplo **or** op > ophi:

I = I - KI \* error \* dt

# clip output

op = max(oplo, min(ophi, op))

# return the controller output and PID terms

**return** [op, I]

###########################################################################

# MPC

###########################################################################

**def** **\_MPC**(self):

m = GEKKO(remote=False)

# 60 second time horizon, 4 sec cycle time, non-uniform

m.time = [**0**, **4**, **8**, **12**, **15**, **20**, **25**, **30**, **35**, **40**, **50**, **60**, **70**, **80**, **90**]

# Parameters

m.U = m.FV(value=**10**)

m.tau = m.FV(value=**5**)

m.alpha1 = m.FV(value=**0.01**) # W / % heater

m.alpha2 = m.FV(value=**0.0075**) # W / % heater

# Manipulated variables

m.Q1 = m.MV(value=**0**)

m.Q1.STATUS = **1** # use to control temperature

m.Q1.FSTATUS = **0** # no feedback measurement

m.Q1.LOWER = **0.0**

m.Q1.UPPER = **100.0**

m.Q1.DMAX = **20.0**

m.Q1.COST = **0.0**

m.Q1.DCOST = **2.0**

m.Q2 = m.MV(value=**0**)

m.Q2.STATUS = **1** # use to control temperature

m.Q2.FSTATUS = **0** # no feedback measurement

m.Q2.LOWER = **0.0**

m.Q2.UPPER = **100.0**

m.Q2.DMAX = **20.0**

m.Q2.COST = **0.0**

m.Q2.DCOST = **2.0**

# Controlled variable

m.TC1 = m.CV(value=**22**)

m.TC1.STATUS = **1** # minimize error with setpoint range

m.TC1.FSTATUS = **1** # receive measurement

m.TC1.TR\_INIT = **1** # reference trajectory

m.TC1.TAU = **10** # time constant for response

# Controlled variable

m.TC2 = m.CV(value=**22**)

m.TC2.STATUS = **1** # minimize error with setpoint range

m.TC2.FSTATUS = **1** # receive measurement

m.TC2.TR\_INIT = **1** # reference trajectory

m.TC2.TAU = **10** # time constant for response

# State variables

m.TH1 = m.SV(value=**22**)

m.TH2 = m.SV(value=**22**)

m.Ta = m.Param(value=**23.0**+**273.15**) # K

m.mass = m.Param(value=**4.0**/**1000.0**) # kg

m.Cp = m.Param(value=**0.5**\***1000.0**) # J/kg-K

m.A = m.Param(value=**10.0**/**100.0**\*\***2**) # Area not between heaters in m^2

m.As = m.Param(value=**2.0**/**100.0**\*\***2**) # Area between heaters in m^2

m.eps = m.Param(value=**0.9**) # Emissivity

m.sigma = m.Const(**5.67e-8**) # Stefan-Boltzmann

# Heater temperatures

m.T1i = m.Intermediate(m.TH1+**273.15**)

m.T2i = m.Intermediate(m.TH2+**273.15**)

# Heat transfer between two heaters

m.Q\_C12 = m.Intermediate(m.U\*m.As\*(m.T2i-m.T1i)) # Conv

m.Q\_R12 = m.Intermediate(m.eps\*m.sigma\*m.As\*(m.T2i\*\***4**-m.T1i\*\***4**)) # Rad

# Semi-fundamental correlations (energy balances)

m.Equation(m.TH1.dt() == (**1.0**/(m.mass\*m.Cp)) \*

(m.U\*m.A\*(m.Ta-m.T1i) +

m.eps \* m.sigma \* m.A \*

(m.Ta\*\***4** - m.T1i\*\***4**) + m.Q\_C12 +

m.Q\_R12 + m.alpha1\*m.Q1))

m.Equation(m.TH2.dt() == (**1.0**/(m.mass\*m.Cp)) \*

(m.U\*m.A\*(m.Ta-m.T2i) +

m.eps \* m.sigma \* m.A \*

(m.Ta\*\***4** - m.T2i\*\***4**) - m.Q\_C12 -

m.Q\_R12 + m.alpha2\*m.Q2))

# Empirical correlations (lag equations to emulate conduction)

m.Equation(m.tau \* m.TC1.dt() == -m.TC1 + m.TH1)

m.Equation(m.tau \* m.TC2.dt() == -m.TC2 + m.TH2)

# Global Options

m.options.IMODE = **6** # MPC

m.options.CV\_TYPE = **1** # Objective type

m.options.NODES = **3** # Collocation nodes

m.options.SOLVER = **3** # 1=APOPT, 3=IPOPT

**return** m

###########################################################################

# THREADING FUNCTION - OPEN LOOP

###########################################################################

**def** **\_work\_man**(self):

# Paraters to start each cycle

Th0 = self.\_Th0

Tc0 = self.\_Tc0

# arrays to store data

t = np.array([])

Q1 = np.array([])

Q2 = np.array([])

T = np.array([[]]).reshape((**0**, **2**))

t = np.append(t, np.array([**0**]), axis=**0**)

T = np.append(T, Tc0.reshape((**1**, **2**)), axis=**0**)

Q1 = np.append(Q1, np.array([self.\_Q10]), axis=**0**)

Q2 = np.append(Q2, np.array([self.\_Q20]), axis=**0**)

**while** self.\_flag:

ts = [t[-**1**], t[-**1**]+self.\_delta\_t]

y = odeint(self.\_heater, Th0, ts, args=(self.\_Q10, self.\_Q20))

Th0 = y[-**1**]

z = odeint(self.\_sensor, Tc0, ts, args=(Th0[**0**], Th0[**1**]))

Tc0 = z[-**1**]

# Measurement noise

Tc\_noise = np.array([

Tc0[**0**] + (np.random.rand()-**0.5**),

Tc0[**1**] + (np.random.rand()-**0.5**)

])

**if** len(t) >= self.\_maxtime:

t = np.delete(t, **0**, **0**)

T = np.delete(T, **0**, **0**)

Q1 = np.delete(Q1, **0**, **0**)

Q2 = np.delete(Q2, **0**, **0**)

t = np.append(t, np.array([ts[-**1**]]), axis=**0**)

T = np.append(T, Tc\_noise.reshape((**1**, **2**)), axis=**0**)

Q1 = np.append(Q1, np.array([self.\_Q10]), axis=**0**)

Q2 = np.append(Q2, np.array([self.\_Q20]), axis=**0**)

self.\_T1\_meas.x = t/**60**

self.\_T1\_meas.y = T[:, **0**] - **273.15**

self.\_PT1.value = np.round(T[-**1**, **0**]-**273.15**, **1**)

self.\_wT1.value = np.round(T[-**1**, **0**]-**273.15**, **1**)

self.\_T2\_meas.x = t/**60**

self.\_T2\_meas.y = T[:, **1**] - **273.15**

self.\_PT2.value = np.round(T[-**1**, **1**]-**273.15**, **1**)

self.\_wT2.value = np.round(T[-**1**, **1**]-**273.15**, **1**)

self.\_u1.x = t/**60**

self.\_u1.y = Q1

self.\_u2.x = t/**60**

self.\_u2.y = Q2

time.sleep(self.\_sleep)

###########################################################################

# THREADING FUNCTION - ON-OFF

###########################################################################

**def** **\_work\_on\_off**(self):

# Paraters to start each cycle

Th0 = self.\_Th0

Tc0 = self.\_Tc0

Q10 = self.\_Q10

Q20 = self.\_Q20

# arrays to store data

t = np.array([])

Q1 = np.array([])

Q2 = np.array([])

T = np.array([[]]).reshape((**0**, **2**))

SP\_T1 = np.array([])

SP\_T2 = np.array([])

t = np.append(t, np.array([**0**]), axis=**0**)

T = np.append(T, Tc0.reshape((**1**, **2**)), axis=**0**)

Q1 = np.append(Q1, np.array([Q10]), axis=**0**)

Q2 = np.append(Q2, np.array([Q20]), axis=**0**)

SP\_T1 = np.append(SP\_T1, np.array([self.\_T1\_SP]), axis=**0**)

SP\_T2 = np.append(SP\_T2, np.array([self.\_T2\_SP]), axis=**0**)

**while** self.\_flag:

# apply ON/OFF controller

# heater 1

**if** (Tc0[**0**]-**273.15**) < (self.\_T1\_SP - self.\_q1\_dt\_on\_off):

Q10 = **100.0**

**elif** (Tc0[**0**]-**273.15**) > (self.\_T1\_SP + self.\_q1\_dt\_on\_off):

Q10 = **0.0**

# heater 2

**if** (Tc0[**1**]-**273.15**) < (self.\_T2\_SP - self.\_q2\_dt\_on\_off):

Q20 = **100.0**

**elif** (Tc0[**1**]-**273.15**) > (self.\_T2\_SP + self.\_q2\_dt\_on\_off):

Q20 = **0.0**

ts = [t[-**1**], t[-**1**]+self.\_delta\_t]

y = odeint(self.\_heater, Th0, ts, args=(Q10, Q20))

Th0 = y[-**1**]

z = odeint(self.\_sensor, Tc0, ts, args=(Th0[**0**], Th0[**1**]))

Tc0 = z[-**1**]

# Measurement noise

Tc\_noise = np.array([

Tc0[**0**] + (np.random.rand()-**0.5**),

Tc0[**1**] + (np.random.rand()-**0.5**)

])

**if** len(t) >= self.\_maxtime:

t = np.delete(t, **0**, **0**)

T = np.delete(T, **0**, **0**)

Q1 = np.delete(Q1, **0**, **0**)

Q2 = np.delete(Q2, **0**, **0**)

SP\_T1 = np.delete(SP\_T1, **0**, **0**)

SP\_T2 = np.delete(SP\_T2, **0**, **0**)

t = np.append(t, np.array([ts[-**1**]]), axis=**0**)

T = np.append(T, Tc\_noise.reshape((**1**, **2**)), axis=**0**)

Q1 = np.append(Q1, np.array([Q10]), axis=**0**)

Q2 = np.append(Q2, np.array([Q20]), axis=**0**)

SP\_T1 = np.append(SP\_T1, np.array([self.\_T1\_SP]), axis=**0**)

SP\_T2 = np.append(SP\_T2, np.array([self.\_T2\_SP]), axis=**0**)

self.\_T1\_meas.x = t/**60**

self.\_T1\_meas.y = T[:, **0**] - **273.15**

self.\_PT1.value = np.round(T[-**1**, **0**]-**273.15**, **1**)

self.\_T1\_set\_point.x = t/**60**

self.\_T1\_set\_point.y = SP\_T1

self.\_T2\_meas.x = t/**60**

self.\_T2\_meas.y = T[:, **1**] - **273.15**

self.\_PT2.value = np.round(T[-**1**, **1**]-**273.15**, **1**)

self.\_T2\_set\_point.x = t/**60**

self.\_T2\_set\_point.y = SP\_T2

self.\_u1.x = t/**60**

self.\_u1.y = Q1

self.\_wQ1.value = np.round(Q1[-**1**], **1**)

self.\_u2.x = t/**60**

self.\_u2.y = Q2

self.\_wQ2.value = np.round(Q2[-**1**], **1**)

time.sleep(self.\_sleep)

###########################################################################

# THREADING FUNCTION - PID

###########################################################################

**def** **\_work\_pid**(self):

# Paraters to start each cycle

Th0 = self.\_Th0

Tc0 = self.\_Tc0

Q10 = self.\_Q10

Q20 = self.\_Q20

# arrays to store data

t = np.array([])

Q1 = np.array([])

Q2 = np.array([])

T = np.array([[]]).reshape((**0**, **2**))

SP\_T1 = np.array([])

SP\_T2 = np.array([])

t = np.append(t, np.array([**0**]), axis=**0**)

T = np.append(T, Tc0.reshape((**1**, **2**)), axis=**0**)

Q1 = np.append(Q1, np.array([Q10]), axis=**0**)

Q2 = np.append(Q2, np.array([Q20]), axis=**0**)

SP\_T1 = np.append(SP\_T1, np.array([self.\_T1\_SP]), axis=**0**)

SP\_T2 = np.append(SP\_T2, np.array([self.\_T2\_SP]), axis=**0**)

# Integral error

ierr1 = **0.0**

ierr2 = **0.0**

**while** self.\_flag:

ts = [t[-**1**], t[-**1**]+self.\_delta\_t]

y = odeint(self.\_heater, Th0, ts, args=(Q10, Q20))

Th0 = y[-**1**]

z = odeint(self.\_sensor, Tc0, ts, args=(Th0[**0**], Th0[**1**]))

Tc0 = z[-**1**]

# Measurement noise

Tc\_noise = np.array([

Tc0[**0**] + (np.random.rand()-**0.5**),

Tc0[**1**] + (np.random.rand()-**0.5**)

])

**if** len(t) >= self.\_maxtime:

t = np.delete(t, **0**, **0**)

T = np.delete(T, **0**, **0**)

Q1 = np.delete(Q1, **0**, **0**)

Q2 = np.delete(Q2, **0**, **0**)

SP\_T1 = np.delete(SP\_T1, **0**, **0**)

SP\_T2 = np.delete(SP\_T2, **0**, **0**)

t = np.append(t, np.array([ts[-**1**]]), axis=**0**)

T = np.append(T, Tc\_noise.reshape((**1**, **2**)), axis=**0**)

Q1 = np.append(Q1, np.array([Q10]), axis=**0**)

Q2 = np.append(Q2, np.array([Q20]), axis=**0**)

SP\_T1 = np.append(SP\_T1, np.array([self.\_T1\_SP]), axis=**0**)

SP\_T2 = np.append(SP\_T2, np.array([self.\_T2\_SP]), axis=**0**)

# Calculate PID output

[Q10, ierr1] = self.\_PID(self.\_T1\_SP, T[-**1**, **0**]-**273.15**,

T[-**2**, **0**]-**273.15**, ierr1, self.\_delta\_t,

self.\_pid1\_gain, self.\_pid1\_reset,

self.\_pid1\_rate)

[Q20, ierr2] = self.\_PID(self.\_T2\_SP, T[-**1**, **1**]-**273.15**,

T[-**2**, **1**]-**273.15**, ierr2, self.\_delta\_t,

self.\_pid2\_gain, self.\_pid2\_reset,

self.\_pid2\_rate)

self.\_T1\_meas.x = t/**60**

self.\_T1\_meas.y = T[:, **0**] - **273.15**

self.\_PT1.value = np.round(T[-**1**, **0**]-**273.15**, **1**)

self.\_T1\_set\_point.x = t/**60**

self.\_T1\_set\_point.y = SP\_T1

self.\_T2\_meas.x = t/**60**

self.\_T2\_meas.y = T[:, **1**] - **273.15**

self.\_PT2.value = np.round(T[-**1**, **1**]-**273.15**, **1**)

self.\_T2\_set\_point.x = t/**60**

self.\_T2\_set\_point.y = SP\_T2

self.\_u1.x = t/**60**

self.\_u1.y = Q1

self.\_wQ1.value = np.round(Q1[-**1**], **1**)

self.\_u2.x = t/**60**

self.\_u2.y = Q2

self.\_wQ2.value = np.round(Q2[-**1**], **1**)

time.sleep(self.\_sleep)

###########################################################################

# THREADING FUNCTION - MPC

###########################################################################

**def** **\_work\_mpc**(self):

Th0 = self.\_Th0

Tc0 = self.\_Tc0

Q10 = self.\_Q10

Q20 = self.\_Q20

# arrays to store data

t = np.array([])

Q1 = np.array([])

Q2 = np.array([])

T = np.array([[]]).reshape((**0**, **2**))

SP\_T1 = np.array([])

SP\_T2 = np.array([])

t = np.append(t, np.array([**0**]), axis=**0**)

T = np.append(T, Tc0.reshape((**1**, **2**)), axis=**0**)

Q1 = np.append(Q1, np.array([Q10]), axis=**0**)

Q2 = np.append(Q2, np.array([Q20]), axis=**0**)

SP\_T1 = np.append(SP\_T1, np.array([self.\_T1\_SP]), axis=**0**)

SP\_T2 = np.append(SP\_T2, np.array([self.\_T2\_SP]), axis=**0**)

# Create MPC object

m = self.\_MPC()

**while** self.\_flag:

# Change SOLVER

**if** self.\_SOLVER == '1 - APOPT':

m.options.SOLVER = **1**

**elif** self.\_SOLVER == '2 - BPOPT':

m.options.SOLVER = **2**

**else**:

m.options.SOLVER = **3**

# Change CVTYPE

**if** self.\_CVTYPE == '1 - Deadband':

m.options.CV\_TYPE = **1**

**else**:

m.options.CV\_TYPE = **2**

# Add measurements to the MPC

m.TC1.MEAS = T[-**1**, **0**] - **273.15**

m.TC2.MEAS = T[-**1**, **1**] - **273.15**

# Update Parameters

m.TC1.TAU = self.\_T1\_tau

m.TC2.TAU = self.\_T2\_tau

m.Q1.DMAX = self.\_Q1\_DMAX

m.Q1.DCOST = self.\_Q1\_DCOST

m.Q2.DMAX = self.\_Q2\_DMAX

m.Q2.DCOST = self.\_Q2\_DCOST

# Update prediction horizon

DT = self.\_delta\_t

m.time = [

**0**,

DT,

DT\***2**,

DT\***3**,

DT\***4**,

DT\***5**,

DT\***6**,

DT\***7**,

DT\***8**,

DT\***10**,

DT\***12**,

DT\***15**,

DT\***18**,

DT\***20**,

DT\***25**]

**if** m.options.CV\_TYPE == **1**:

# Input setpoint with deadband +/- DT

DT1 = self.\_T1\_dt

m.TC1.SPHI = self.\_T1\_SP + DT1

m.TC1.SPLO = self.\_T1\_SP - DT1

DT2 = self.\_T2\_dt

m.TC2.SPHI = self.\_T2\_SP + DT2

m.TC2.SPLO = self.\_T2\_SP - DT2

**else**:

m.TC1.SP = self.\_T1\_SP

m.TC2.SP = self.\_T2\_SP

**try**:

# Solve MPC

m.solve(disp=False)

# Check if successful solution

**if** (m.options.APPSTATUS == **1**):

# retrieve new value

Q10 = m.Q1.NEWVAL

Q20 = m.Q2.NEWVAL

**except**:

# Keep previous value

**pass**

ts = [t[-**1**], t[-**1**]+self.\_delta\_t]

y = odeint(self.\_heater, Th0, ts, args=(Q10, Q20))

Th0 = y[-**1**]

z = odeint(self.\_sensor, Tc0, ts, args=(Th0[**0**], Th0[**1**]))

Tc0 = z[-**1**]

# Measurement noise

Tc\_noise = np.array([

Tc0[**0**] + (np.random.rand()-**0.5**),

Tc0[**1**] + (np.random.rand()-**0.5**)

])

**if** len(t) >= self.\_maxtime:

t = np.delete(t, **0**, **0**)

T = np.delete(T, **0**, **0**)

Q1 = np.delete(Q1, **0**, **0**)

Q2 = np.delete(Q2, **0**, **0**)

SP\_T1 = np.delete(SP\_T1, **0**, **0**)

SP\_T2 = np.delete(SP\_T2, **0**, **0**)

t = np.append(t, np.array([ts[-**1**]]), axis=**0**)

T = np.append(T, Tc\_noise.reshape((**1**, **2**)), axis=**0**)

Q1 = np.append(Q1, np.array([Q10]), axis=**0**)

Q2 = np.append(Q2, np.array([Q20]), axis=**0**)

SP\_T1 = np.append(SP\_T1, np.array([self.\_T1\_SP]), axis=**0**)

SP\_T2 = np.append(SP\_T2, np.array([self.\_T2\_SP]), axis=**0**)

self.\_T1\_meas.x = t/**60**

self.\_T1\_meas.y = T[:, **0**] - **273.15**

self.\_PT1.value = np.round(T[-**1**, **0**]-**273.15**, **1**)

self.\_T1\_set\_point.x = t/**60**

self.\_T1\_set\_point.y = SP\_T1

self.\_T2\_meas.x = t/**60**

self.\_T2\_meas.y = T[:, **1**] - **273.15**

self.\_PT2.value = np.round(T[-**1**, **1**]-**273.15**, **1**)

self.\_T2\_set\_point.x = t/**60**

self.\_T2\_set\_point.y = SP\_T2

self.\_u1.x = t/**60**

self.\_u1.y = Q1

self.\_wQ1.value = np.round(Q1[-**1**], **1**)

self.\_u2.x = t/**60**

self.\_u2.y = Q2

self.\_wQ2.value = np.round(Q2[-**1**], **1**)

time.sleep(self.\_sleep)

**Code itclab\_PID.ipynb**

**import** **control\_arduino** **as** **cd**

run\_control = cd.GUI()

run\_control.app()

run\_control.config()

**Code demo.ipynb**

**import** **control\_demo** **as** **cd**

demo = cd.GUI()

demo.app()

demo.config()

## iTCLab 8

**Code IoT\_OnOff.ino**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Program : iTCLab Kit temperature monitoring **and** control On/Off via IoT

\* By : Assoc. Prof. Dr. Basuki Rahmat, S.Si, MT, ITS-AI,

\* Assoc. Prof. Dr. Muljono, S.Si, M.Kom, et al

\* Pro. Team : i-ot.net, io-t.net

\* R. Group : Intelligent Control, Robotics **and** Automation Systems Research Group

\* Univ. : Universitas Pembangunan Nasional "Veteran" Jawa Timur

\* Country : Indonesia

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <WiFi.h>

#include <PubSubClient.h>

#include <Arduino.h>

const char\* ssid = "Redmi-9"; // Enter your WiFi name

const char\* password = "20062004"; // Enter WiFi password

#define mqttServer "broker.hivemq.com"

#define mqttPort 1883

WiFiServer server(**80**);

WiFiClient espClient;

PubSubClient client(espClient);

String Topic;

String Payload;

// constants

const int baud = **115200**; // serial baud rate

// pin numbers corresponding to signals on the iTCLab Shield

const int pinT1 = **34**; // T1

const int pinT2 = **35**; // T2

const int pinQ1 = **32**; // Q1

const int pinQ2 = **33**; // Q2

const int pinLED = **26**; // LED

// setting PWM properties

const int freq = **5000**; //**5000**

const int ledChannel = **0**;

const int Q1Channel = **1**;

const int Q2Channel = **2**;

const int resolutionLedChannel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ1Channel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ2Channel = **8**; //Resolution **8**, **10**, **12**, **15**

float cel, cel1, degC, degC1;

const float upper\_temperature\_limit = **58**;

// **global** variables

float Q1 = **0**; // value written to Q1 pin

float Q2 = **0**; // value written to Q2 pin

int iwrite\_value = **25**; // integer value **for** writing

int iwrite\_min = **0**; // integer value **for** writing

void setup() {

// put your setup code here, to run once:

Serial.begin(baud);

**while** (!Serial) {

; // wait **for** serial port to connect.

}

// configure pinQ1 PWM functionalitites

ledcSetup(Q1Channel, freq, resolutionQ1Channel);

// attach the channel to the pinQ1 to be controlled

ledcAttachPin(pinQ1, Q1Channel);

// configure pinQ2 PWM functionalitites

ledcSetup(Q2Channel, freq, resolutionQ2Channel);

// attach the channel to the pinQ2 to be controlled

ledcAttachPin(pinQ2, Q2Channel);

// configure pinLED PWM functionalitites

ledcSetup(ledChannel, freq, resolutionLedChannel);

// attach the channel to the pinLED to be controlled

ledcAttachPin(pinLED, ledChannel);

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

ledcWrite(ledChannel,**0**);

// Connect to WiFi network

Serial.println();

Serial.println();

Serial.**print**("Connecting to ");

Serial.println(ssid);

WiFi.begin(ssid, password);

**while** (WiFi.status() != WL\_CONNECTED) {

delay(**500**);

Serial.**print**(".");

}

Serial.println("");

Serial.println("WiFi connected");

// Connect to Server IoT (CloudMQTT)

client.setServer(mqttServer, mqttPort);

client.setCallback(receivedCallback);

**while** (!client.connected()) {

Serial.println("Connecting to CLoud IoT ...");

// **if** (client.connect("ESP32Client", mqttUser, mqttPassword )) {

**if** (client.connect("iTCLab Suhu On/Off")) {

Serial.println("connected");

Serial.**print**("Message received: ");

} **else** {

Serial.**print**("failed with state ");

Serial.**print**(client.state());

delay(**2000**);

}

client.subscribe("heater1bas");

client.subscribe("heater2bas");

}

}

void Q1on(){

ledcWrite(Q1Channel,iwrite\_value);

//Q1 = iwrite\_value/**255**\***100**;

//Serial.println(Q1);

}

void Q1off(){

ledcWrite(Q1Channel,iwrite\_min);

//Q1 = iwrite\_min/**255**\***100**;

//Serial.println(Q1);

}

void Q2on(){

ledcWrite(Q2Channel,iwrite\_value);

//Q2 = iwrite\_value/**255**\***100**;

//Serial.println(Q2);

}

void Q2off(){

ledcWrite(Q2Channel,iwrite\_min);

//Q2 = iwrite\_min/**255**\***100**;

//Serial.println(Q2);

}

void ledon(){

ledcWrite(ledChannel,iwrite\_value);

}

void ledoff(){

ledcWrite(ledChannel,iwrite\_min);

}

void cektemp(){

degC = analogRead(pinT1) \* **0.322265625** ; // use **for** **3.3**v AREF

cel = degC/**10**;

degC1 = analogRead(pinT2) \* **0.322265625** ; // use **for** **3.3**v AREF

cel1 = degC1/**10**;

Serial.**print**("Temperature: ");

Serial.**print**(cel); // **print** the temperature T1 **in** Celsius

Serial.**print**("°C");

Serial.**print**(" ~ "); // separator between Celsius **and** Fahrenheit

Serial.**print**(cel1); // **print** the temperature T2 **in** Celsius

Serial.println("°C");

}

void receivedCallback(char\* topic, byte\* payload, unsigned int length) {

/\* we got '1' -> Q1\_on \*/

**if** ((char)payload[**0**] == '1') {

Q1on();

Serial.println("Q1 On");

}

/\* we got '2' -> Q1\_off \*/

**if** ((char)payload[**0**] == '2') {

Q1off();

Serial.println("Q1 Off");

}

/\* we got '3' -> Q2\_on \*/

**if** ((char)payload[**0**] == '3') {

Q2on();

Serial.println("Q2 On");

}

/\* we got '4' -> Q2\_off \*/

**if** ((char)payload[**0**] == '4') {

Q2off();

Serial.println("Q2 Off");

}

}

void loop() {

char suhu1[**4**];

char suhu2[**4**];

client.loop();

// put your main code here, to run repeatedly:

cektemp();

**if** (cel > upper\_temperature\_limit){

Q1off();

ledon();

}

**else** {

Q1on();

ledoff();

}

**if** (cel1 > upper\_temperature\_limit){

Q2off();

ledon();

}

**else** {

Q2on();

ledoff();

}

delay (**100**);

Serial.**print**("Temperature T1: ");

Serial.**print**(cel);

Serial.**print**(" Celcius ");

Serial.println(" send to Broker MQTT");

dtostrf(cel, **1**, **0**, suhu1);

client.publish("Suhu1",suhu1);

delay (**200**);

Serial.**print**("Temperature T2: ");

Serial.**print**(cel1);

Serial.**print**(" Celcius ");

Serial.println(" send to Broker MQTT");

dtostrf(cel1, **1**, **0**, suhu2);

client.publish("Suhu2",suhu2);

delay (**200**);

}

## iTCLab 9

**Code IoT\_Monitor.ino**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Program : PID-iTCLab Monitoring Using IoT

\* By : Assoc. Prof. Dr. Basuki Rahmat, S.Si, MT, ITS-AI,

\* Assoc. Prof. Dr. Muljono, S.Si, M.Kom, et al

\* Pro. Team : i-ot.net, io-t.net

\* R. Group : Intelligent Control, Robotics **and** Automation Systems Research Group

\* Univ. : Universitas Pembangunan Nasional "Veteran" Jawa Timur

\* Country : Indonesia

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <WiFi.h>

#include <PubSubClient.h>

#include <Arduino.h>

const char\* ssid = "Redmi-9"; // Enter your WiFi name

const char\* password = "20062004"; // Enter WiFi password

#define mqttServer "broker.hivemq.com"

#define mqttPort 1883

WiFiServer server(**80**);

WiFiClient espClient;

PubSubClient client(espClient);

String Topic;

String Payload;

// constants

const int baud = **115200**; // serial baud rate

// pin numbers corresponding to signals on the iTCLab Shield

const int pinT1 = **34**; // T1

const int pinT2 = **35**; // T2

const int pinQ1 = **32**; // Q1

const int pinQ2 = **33**; // Q2

const int pinLED = **26**; // LED

// setting PWM properties

const int freq = **5000**; //**5000**

const int ledChannel = **0**;

const int Q1Channel = **1**;

const int Q2Channel = **2**;

const int resolutionLedChannel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ1Channel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ2Channel = **8**; //Resolution **8**, **10**, **12**, **15**

float cel, cel1, degC, degC1;

float P, I, D, Kc, tauI, tauD;

float KP, KI, KD, op0, ophi, oplo, error, dpv;

float sp = **35**, //set point

pv = **0**, //current temperature

pv\_last = **0**, //prior temperature

ierr = **0**, //integral error

dt = **0**, //time between measurements

op = **0**; //PID controller output

unsigned long ts = **0**, new\_ts = **0**; //timestamp

const float upper\_temperature\_limit = **58**;

// **global** variables

float Q1 = **0**; // value written to Q1 pin

float Q2 = **0**; // value written to Q2 pin

int iwrite\_value = **25**; // integer value **for** writing

int iwrite\_led = **255**; // integer value **for** writing

int iwrite\_min = **0**; // integer value **for** writing

void setup() {

// put your setup code here, to run once:

ts = millis();

Serial.begin(baud);

**while** (!Serial) {

; // wait **for** serial port to connect.

}

// configure pinQ1 PWM functionalitites

ledcSetup(Q1Channel, freq, resolutionQ1Channel);

// attach the channel to the pinQ1 to be controlled

ledcAttachPin(pinQ1, Q1Channel);

// configure pinQ2 PWM functionalitites

ledcSetup(Q2Channel, freq, resolutionQ2Channel);

// attach the channel to the pinQ2 to be controlled

ledcAttachPin(pinQ2, Q2Channel);

// configure pinLED PWM functionalitites

ledcSetup(ledChannel, freq, resolutionLedChannel);

// attach the channel to the pinLED to be controlled

ledcAttachPin(pinLED, ledChannel);

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

ledcWrite(ledChannel,**0**);

// Connect to WiFi network

Serial.println();

Serial.println();

Serial.**print**("Connecting to ");

Serial.println(ssid);

WiFi.begin(ssid, password);

**while** (WiFi.status() != WL\_CONNECTED) {

delay(**500**);

Serial.**print**(".");

}

Serial.println("");

Serial.println("WiFi connected");

// Connect to Server IoT (CloudMQTT)

client.setServer(mqttServer, mqttPort);

client.setCallback(receivedCallback);

**while** (!client.connected()) {

Serial.println("Connecting to MQTT Broker ...");

**if** (client.connect("PID-iTCLab Monitoring Using IoT...")) {

Serial.println("connected");

Serial.**print**("Message received: ");

} **else** {

Serial.**print**("failed with state ");

Serial.**print**(client.state());

delay(**2000**);

}

//client.subscribe("heater1");

//client.subscribe("heater2");

}

}

void Q1on(){

ledcWrite(Q1Channel,iwrite\_value);

//Q1 = iwrite\_value/**255**\***100**;

//Serial.println(Q1);

}

void Q1off(){

ledcWrite(Q1Channel,iwrite\_min);

//Q1 = iwrite\_min/**255**\***100**;

//Serial.println(Q1);

}

void Q2on(){

ledcWrite(Q2Channel,iwrite\_value);

//Q2 = iwrite\_value/**255**\***100**;

//Serial.println(Q2);

}

void Q2off(){

ledcWrite(Q2Channel,iwrite\_min);

//Q2 = iwrite\_min/**255**\***100**;

//Serial.println(Q2);

}

void ledon(){

ledcWrite(ledChannel,iwrite\_led);

}

void ledoff(){

ledcWrite(ledChannel,iwrite\_min);

}

void cektemp(){

degC = analogRead(pinT1) \* **0.322265625** ; // use **for** **3.3**v AREF

cel = degC/**10**;

degC1 = analogRead(pinT2) \* **0.322265625** ; // use **for** **3.3**v AREF

cel1 = degC1/**10**;

Serial.**print**("Temperature T1: ");

Serial.**print**(cel); // **print** the temperature T1 **in** Celsius

Serial.**print**("°C");

Serial.**print**(" ~ "); // separator between Celsius **and** Fahrenheit

Serial.**print**("Temperature T2: ");

Serial.**print**(cel1); // **print** the temperature T2 **in** Celsius

Serial.println("°C");

}

float pid(float sp, float pv, float pv\_last, float& ierr, float dt) {

float Kc = **10.0**; // K / %Heater

float tauI = **50.0**; // sec

float tauD = **1.0**; // sec

// PID coefficients

float KP = Kc;

float KI = Kc / tauI;

float KD = Kc\*tauD;

// upper **and** lower bounds on heater level

float ophi = **100**;

float oplo = **0**;

// calculate the error

float error = sp - pv;

// calculate the integral error

ierr = ierr + KI \* error \* dt;

// calculate the measurement derivative

float dpv = (pv - pv\_last) / dt;

// calculate the PID output

float P = KP \* error; //proportional contribution

float I = ierr; //integral contribution

float D = -KD \* dpv; //derivative contribution

float op = P + I + D;

// implement anti-reset windup

**if** ((op < oplo) || (op > ophi)) {

I = I - KI \* error \* dt;

// clip output

op = max(oplo, min(ophi, op));

}

ierr = I;

Serial.println("sp="+String(sp) + " pv=" + String(pv) + " dt=" + String(dt) + " op=" + String(op) + " P=" + String(P) + " I=" + String(I) + " D=" + String(D));

**return** op;

}

void receivedCallback(char\* topic, byte\* payload, unsigned int length) {

/\* we got '1' -> Q1\_on \*/

**if** ((char)payload[**0**] == '1') {

Q1on();

Serial.println("Q1 On");

}

/\* we got '2' -> Q1\_off \*/

**if** ((char)payload[**0**] == '2') {

Q1off();

Serial.println("Q1 Off");

}

/\* we got '3' -> Q2\_on \*/

**if** ((char)payload[**0**] == '3') {

Q2on();

Serial.println("Q2 On");

}

/\* we got '4' -> Q2\_off \*/

**if** ((char)payload[**0**] == '4') {

Q2off();

Serial.println("Q2 Off");

}

}

void loop() {

new\_ts = millis();

**if** (new\_ts - ts > **1000**) {

char suhu1[**4**];

char suhu2[**4**];

char SetPoint[**4**];

char Nilai\_op[**4**];

char Nilai\_P[**4**];

char Nilai\_I[**4**];

char Nilai\_D[**4**];

client.loop();

// put your main code here, to run repeatedly:

cektemp();

**if** (cel > upper\_temperature\_limit){

Q1off();

ledon();

}

**else** {

Q1on();

ledoff();

}

**if** (cel1 > upper\_temperature\_limit){

Q2off();

ledon();

}

**else** {

Q2on();

ledoff();

}

//delay (**100**);

pv = cel; // Temperature T1

dt = (new\_ts - ts) / **1000.0**;

ts = new\_ts;

op = pid(sp,pv,pv\_last,ierr,dt);

ledcWrite(Q1Channel,op);

pv\_last = pv;

dtostrf(cel, **1**, **0**, suhu1);

client.publish("Suhu1",suhu1);

dtostrf(sp, **1**, **0**, SetPoint);

client.publish("SetPoint",SetPoint);

dtostrf(op, **1**, **0**, Nilai\_op);

client.publish("Nilai\_op",Nilai\_op);

delay (**200**);

dtostrf(cel1, **1**, **0**, suhu2);

client.publish("Suhu2",suhu2);

delay (**200**);

}

}

## iTCLab 10

**Code IoT\_Control.ino**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Program : PID-iTCLab Controlling Using IoT

\* By : Assoc. Prof. Dr. Basuki Rahmat, S.Si, MT, ITS-AI,

\* Assoc. Prof. Dr. Muljono, S.Si, M.Kom, et al

\* Pro. Team : i-ot.net, io-t.net

\* R. Group : Intelligent Control, Robotics **and** Automation Systems Research Group

\* Univ. : Universitas Pembangunan Nasional "Veteran" Jawa Timur

\* Country : Indonesia

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <WiFi.h>

#include <PubSubClient.h>

#include <Arduino.h>

const char\* ssid = "Redmi-9"; // Enter your WiFi name

const char\* password = "20062004"; // Enter WiFi password

#define mqttServer "broker.hivemq.com"

#define mqttPort 1883

WiFiServer server(**80**);

WiFiClient espClient;

PubSubClient client(espClient);

String Topic;

String Payload;

// constants

const int baud = **115200**; // serial baud rate

// pin numbers corresponding to signals on the iTCLab Shield

const int pinT1 = **34**; // T1

const int pinT2 = **35**; // T2

const int pinQ1 = **32**; // Q1

const int pinQ2 = **33**; // Q2

const int pinLED = **26**; // LED

// setting PWM properties

const int freq = **5000**; //**5000**

const int ledChannel = **0**;

const int Q1Channel = **1**;

const int Q2Channel = **2**;

const int resolutionLedChannel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ1Channel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ2Channel = **8**; //Resolution **8**, **10**, **12**, **15**

float cel, cel1, degC, degC1;

float P, I, D;

float KP, KI, KD, op0, ophi, oplo, error, dpv;

float sp = **30**, //set point

pv = **0**, //current temperature

pv\_last = **0**, //prior temperature

ierr = **0**, //integral error

dt = **0**, //time between measurements

op = **0**; //PID controller output

int autoSet = **0**; // autoSet = **1** otomatis sesuai Default

//float Kc = **0**;

//float tauI = **0**;

//float tauD = **0**;

// Default = autoset = **1** otomatis sesuai Default

float Kc = **10.0**; // K / %Heater

float tauI = **50.0**; // sec

float tauD = **1.0**; // sec

unsigned long ts = **0**, new\_ts = **0**; //timestamp

const float upper\_temperature\_limit = **58**;

// **global** variables

float Q1 = **0**; // value written to Q1 pin

float Q2 = **0**; // value written to Q2 pin

int iwrite\_value = **25**; // integer value **for** writing

int iwrite\_led = **255**; // integer value **for** writing

int iwrite\_min = **0**; // integer value **for** writing

void setup() {

// put your setup code here, to run once:

ts = millis();

Serial.begin(baud);

**while** (!Serial) {

; // wait **for** serial port to connect.

}

// configure pinQ1 PWM functionalitites

ledcSetup(Q1Channel, freq, resolutionQ1Channel);

// attach the channel to the pinQ1 to be controlled

ledcAttachPin(pinQ1, Q1Channel);

// configure pinQ2 PWM functionalitites

ledcSetup(Q2Channel, freq, resolutionQ2Channel);

// attach the channel to the pinQ2 to be controlled

ledcAttachPin(pinQ2, Q2Channel);

// configure pinLED PWM functionalitites

ledcSetup(ledChannel, freq, resolutionLedChannel);

// attach the channel to the pinLED to be controlled

ledcAttachPin(pinLED, ledChannel);

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

ledcWrite(ledChannel,**0**);

// Connect to WiFi network

Serial.println();

Serial.println();

Serial.**print**("Connecting to ");

Serial.println(ssid);

WiFi.begin(ssid, password);

**while** (WiFi.status() != WL\_CONNECTED) {

delay(**500**);

Serial.**print**(".");

}

Serial.println("");

Serial.println("WiFi connected");

// Connect to Server IoT (CloudMQTT)

client.setServer(mqttServer, mqttPort);

client.setCallback(receivedCallback);

**while** (!client.connected()) {

Serial.println("Connecting to MQTT Broker ...");

**if** (client.connect("PID-iTCLab Controlling Using IoT...")) {

Serial.println("connected");

Serial.**print**("Message received: ");

} **else** {

Serial.**print**("failed with state ");

Serial.**print**(client.state());

delay(**1000**);

}

client.subscribe("autoSet");

client.subscribe("SetPoint");

client.subscribe("Nilai\_Kc");

client.subscribe("Nilai\_tauI");

client.subscribe("Nilai\_tauD");

}

}

void Q1on(){

ledcWrite(Q1Channel,iwrite\_value);

//Q1 = iwrite\_value/**255**\***100**;

//Serial.println(Q1);

}

void Q1off(){

ledcWrite(Q1Channel,iwrite\_min);

//Q1 = iwrite\_min/**255**\***100**;

//Serial.println(Q1);

}

void Q2on(){

ledcWrite(Q2Channel,iwrite\_value);

//Q2 = iwrite\_value/**255**\***100**;

//Serial.println(Q2);

}

void Q2off(){

ledcWrite(Q2Channel,iwrite\_min);

//Q2 = iwrite\_min/**255**\***100**;

//Serial.println(Q2);

}

void ledon(){

ledcWrite(ledChannel,iwrite\_led);

}

void ledoff(){

ledcWrite(ledChannel,iwrite\_min);

}

void cektemp(){

degC = analogRead(pinT1) \* **0.322265625** ; // use **for** **3.3**v AREF

cel = degC/**10**;

degC1 = analogRead(pinT2) \* **0.322265625** ; // use **for** **3.3**v AREF

cel1 = degC1/**10**;

}

float pid(float sp, float Kc, float tauI, float tauD, float pv, float pv\_last, float& ierr, float dt) {

// PID coefficients

float KP = Kc;

float KI = Kc / tauI;

float KD = Kc\*tauD;

// upper **and** lower bounds on heater level

float ophi = **100**;

float oplo = **0**;

// calculate the error

float error = sp - pv;

// calculate the integral error

ierr = ierr + KI \* error \* dt;

// calculate the measurement derivative

float dpv = (pv - pv\_last) / dt;

// calculate the PID output

float P = KP \* error; //proportional contribution

float I = ierr; //integral contribution

float D = -KD \* dpv; //derivative contribution

float op = P + I + D;

// implement anti-reset windup

**if** ((op < oplo) || (op > ophi)) {

I = I - KI \* error \* dt;

// clip output

op = max(oplo, min(ophi, op));

}

ierr = I;

Serial.println("sp="+String(sp) + " pv=" + String(pv) + " dt=" + String(dt) + " op=" + String(op) + " P=" + String(P) + " I=" + String(I) + " D=" + String(D));

**return** op;

}

void receivedCallback(char\* topic, byte\* payload, unsigned int length) {

Topic = topic;

char autoS[**60**];

int i;

**for** (i=**0**;i<length;i++){

autoS[i] = payload[i];

}

autoS[i] = '**\0**';

Payload = String(autoS);

}

void loop() {

new\_ts = millis();

**if** (new\_ts - ts > **1000**) {

char suhu1[**4**];

char suhu2[**4**];

char Nilai\_op[**4**];

char Tampil\_SP[**4**];

char Tampil\_Kc[**4**];

char Tampil\_tauI[**4**];

char Tampil\_tauD[**4**];

client.loop();

// put your main code here, to run repeatedly:

cektemp();

**if** (cel > upper\_temperature\_limit){

Q1off();

ledon();

}

**else** {

Q1on();

ledoff();

}

**if** (cel1 > upper\_temperature\_limit){

Q2off();

ledon();

}

**else** {

Q2on();

ledoff();

}

**if**(Topic=="autoSet"){

autoSet=Payload.toInt();

}

**if**(Topic=="Nilai\_Kc"){

Kc=Payload.toFloat();

}

**if**(Topic=="Nilai\_tauI"){

tauI=Payload.toFloat();

}

**if**(Topic=="Nilai\_tauD"){

tauD=Payload.toFloat()/**6**;

}

**if**(Topic=="SetPoint"){

sp=Payload.toFloat();

}

Serial.println("<-------------------------->");

Serial.**print**("autoSet: ");

Serial.println(autoSet);

Serial.**print**("SetPoint: ");

Serial.println(sp);

Serial.**print**("Nilai\_Kc: ");

Serial.println(Kc);

Serial.**print**("Nilai\_tauI: ");

Serial.println(tauI);

Serial.**print**("Nilai\_tauD: ");

Serial.println(tauD);

Serial.println("<-------------------------->");

dtostrf(cel, **1**, **0**, suhu1);

client.publish("Suhu1",suhu1);

dtostrf(cel1, **1**, **0**, suhu2);

client.publish("Suhu2",suhu2);

dtostrf(sp, **1**, **0**, Tampil\_SP);

client.publish("Tampil\_SP",Tampil\_SP);

dtostrf(Kc, **1**, **0**, Tampil\_Kc);

client.publish("Tampil\_Kc",Tampil\_Kc);

dtostrf(tauI, **1**, **0**, Tampil\_tauI);

client.publish("Tampil\_tauI",Tampil\_tauI);

dtostrf(tauD, **1**, **0**, Tampil\_tauD);

client.publish("Tampil\_tauD",Tampil\_tauD);

**if**(autoSet==**1**){

sp = **35**;

Kc = **10.0**; // K / %Heater

tauI = **50.0**; // sec

tauD = **1.0**; // sec

}**else** **if**(autoSet == **0**){

// bisa diubah2, sesuai yg muncul terakhir, dan setelah diubah2

}

pv = cel; // Temperature T1

dt = (new\_ts - ts) / **1000.0**;

ts = new\_ts;

op = pid(sp,Kc,tauI,tauD,pv,pv\_last,ierr,dt); // PID Process

ledcWrite(Q1Channel,op);

pv\_last = pv;

dtostrf(op, **1**, **0**, Nilai\_op);

client.publish("Nilai\_op",Nilai\_op);

}

}

## iTCLab 11

**Code XOR\_Gate\_Programming\_using\_Deep\_Learning-checkpoint.ipynb**

# Library yg dibutuhkan

**import** **numpy** **as** **np**

# Pasangan data latih

XOR\_X = np.array([

[**0**, **0**],

[**0**, **1**],

[**1**, **0**],

[**1**, **1**]

])

XOR\_Y = np.array([

[**0**],

[**1**],

[**1**],

[**0**]

])

# 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(**2**, activation='sigmoid'))

# Tambahkan lapisan keluaran

model.add(Dense(**1**, activation='sigmoid'))

# Bentuk keluaran model

model.output\_shape

# Ringkasan model

model.summary()

# Konfigurasi model

model.get\_config()

# Buat daftar semua tensor bobot

model.get\_weights()

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

model.fit(XOR\_X, XOR\_Y,epochs=**1000**, batch\_size=**1**, verbose=**1**)

Hasil\_Prediksi\_Keras = model.predict(XOR\_X)

**print**(Hasil\_Prediksi\_Keras)

**import** **matplotlib.pyplot** **as** **plt**

plt.plot(XOR\_Y, 'bo', label='Target', linewidth=**2**, markersize=**12**)

plt.plot(Hasil\_Prediksi\_Keras, 'r+', label='Keras Output', linewidth=**2**, markersize=**12**)

plt.legend(loc='upper right')

plt.show()

**from** **sklearn.metrics** **import** mean\_squared\_error

**from** **math** **import** sqrt

mse2 = mean\_squared\_error(XOR\_Y, Hasil\_Prediksi\_Keras)

rmse2 = sqrt(mean\_squared\_error(XOR\_Y, Hasil\_Prediksi\_Keras))

**print**('MSE =',mse2)

**print**('RMSE =',rmse2)

## iTCLab 12

**Code Deep\_PID.ipynb**

**port** **numpy** **as** **np** # For matrix math

**import** **matplotlib.pyplot** **as** **plt** # For plotting

**import** **sys** # For printing

# 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**]

])

# 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'))

# Bentuk keluaran model

model.output\_shape

# Ringkasan model

model.summary()

# Konfigurasi model

model.get\_config()

# Buat daftar semua tensor bobot

model.get\_weights()

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

model.fit(X, y,epochs=**100**, batch\_size=**1**, verbose=**1**)

Hasil\_Prediksi\_Keras = model.predict(X)

**print**(Hasil\_Prediksi\_Keras)

ujicoba1 = np.array([

[**1**, **1**]

])

ujicoba1

outDL = model.predict(ujicoba1)

outDL

result\_Kc = outDL[**0**,**0**]

result\_tauI = outDL[**0**,**1**]

result\_tauD = outDL[**0**,**2**]

result\_Kc

result\_tauI

result\_tauD

# 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()

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **scipy.integrate** **import** odeint

**import** **ipywidgets** **as** **wg**

**from** **IPython.display** **import** display

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 = result\_Kc

tauI\_slide = result\_tauI

tauD\_slide = result\_tauD

wg.interact(pidPlot, Kc=Kc\_slide, tauI=tauI\_slide, tauD=tauD\_slide)

ujicoba2 = np.array([

[**0.4**, **1.2**]

])

ujicoba2

outDL = model.predict(ujicoba2)

outDL

result\_Kc = outDL[**0**,**0**]

result\_tauI = outDL[**0**,**1**]

result\_tauD = outDL[**0**,**2**]

result\_Kc

result\_tauI

result\_tauD

# 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()

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **scipy.integrate** **import** odeint

**import** **ipywidgets** **as** **wg**

**from** **IPython.display** **import** display

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 = result\_Kc

tauI\_slide = result\_tauI

tauD\_slide = result\_tauD

wg.interact(pidPlot, Kc=Kc\_slide, tauI=tauI\_slide, tauD=tauD\_slide)

ujicoba3 = np.array([

[**1.2**, **0.1**]

])

ujicoba3

outDL = model.predict(ujicoba3)

outDL

result\_Kc = outDL[**0**,**0**]

result\_tauI = outDL[**0**,**1**]

result\_tauD = outDL[**0**,**2**]

result\_Kc

result\_tauI

result\_tauD

# 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()

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **scipy.integrate** **import** odeint

**import** **ipywidgets** **as** **wg**

**from** **IPython.display** **import** display

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 = result\_Kc

tauI\_slide = result\_tauI

tauD\_slide = result\_tauD

wg.interact(pidPlot, Kc=Kc\_slide, tauI=tauI\_slide, tauD=tauD\_slide)

## iTCLab 13

**Code Deep\_PID\_iTCLab.ino**

/\*

iTCLab Internet-Based Temperature Control Lab Firmware

Jeffrey Kantor, Initial Version

John Hedengren, Modified

Oct **2017**

Basuki Rahmat, Modified

April **2022**

This firmware **is** loaded into the Internet-Based Temperature Control Laboratory ESP32 to

provide a high level interface to the Internet-Based Temperature Control Lab. The firmware

scans the serial port looking **for** case-insensitive commands:

Q1 set Heater **1**, range **0** to **100**% subject to limit (**0**-**255** int)

Q2 set Heater **2**, range **0** to **100**% subject to limit (**0**-**255** int)

T1 get Temperature T1, returns deg C **as** string

T2 get Temperature T2, returns dec C **as** string

VER get firmware version string

X stop, enter sleep mode

Limits on the heater can be configured **with** the constants below.

\*/

#include <Arduino.h>

// constants

const String vers = "1.04"; // version of this firmware

const int baud = **115200**; // serial baud rate

const char sp = ' '; // command separator

const char nl = '**\n**'; // command terminator

// pin numbers corresponding to signals on the iTCLab Shield

const int pinT1 = **34**; // T1

const int pinT2 = **35**; // T2

const int pinQ1 = **32**; // Q1

const int pinQ2 = **33**; // Q2

const int pinLED = **26**; // LED

// setting PWM properties

const int freq = **5000**; //**5000**

const int ledChannel = **0**;

const int Q1Channel = **1**;

const int Q2Channel = **2**;

const int resolutionLedChannel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ1Channel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ2Channel = **8**; //Resolution **8**, **10**, **12**, **15**

const double batas\_suhu\_atas = **59**;

// **global** variables

char Buffer[**64**]; // buffer **for** parsing serial input

String cmd; // command

double pv = **0**; // pin value

float level; // LED level (**0**-**100**%)

double Q1 = **0**; // value written to Q1 pin

double Q2 = **0**; // value written to Q2 pin

int iwrite = **0**; // integer value **for** writing

float dwrite = **0**; // float value **for** writing

int n = **10**; // number of samples **for** each temperature measurement

void parseSerial(void) {

int ByteCount = Serial.readBytesUntil(nl,Buffer,sizeof(Buffer));

String read\_ = String(Buffer);

memset(Buffer,**0**,sizeof(Buffer));

// separate command **from** **associated** **data**

int idx = read\_.indexOf(sp);

cmd = read\_.substring(**0**,idx);

cmd.trim();

cmd.toUpperCase();

// extract data. toInt() returns **0** on error

String data = read\_.substring(idx+**1**);

data.trim();

pv = data.toFloat();

}

// Q1\_max = **100**%

// Q2\_max = **100**%

void dispatchCommand(void) {

**if** (cmd == "Q1") {

Q1 = max(**0.0**, min(**25.0**, pv));

iwrite = int(Q1 \* **2.0**); // **10.**? max

iwrite = max(**0**, min(**255**, iwrite));

ledcWrite(Q1Channel,iwrite);

Serial.println(Q1);

}

**else** **if** (cmd == "Q2") {

Q2 = max(**0.0**, min(**25.0**, pv));

iwrite = int(Q2 \* **2.0**); // **10.**? max

iwrite = max(**0**, min(**255**, iwrite));

ledcWrite(Q2Channel,iwrite);

Serial.println(Q2);

}

**else** **if** (cmd == "T1") {

float mV = **0.0**;

float degC = **0.0**;

**for** (int i = **0**; i < n; i++) {

mV = (float) analogRead(pinT1) \* **0.322265625**;

degC = degC + mV/**10.0**;

}

degC = degC / float(n);

Serial.println(degC);

}

**else** **if** (cmd == "T2") {

float mV = **0.0**;

float degC = **0.0**;

**for** (int i = **0**; i < n; i++) {

mV = (float) analogRead(pinT2) \* **0.322265625**;

degC = degC + mV/**10.0**;

}

degC = degC / float(n);

Serial.println(degC);

}

**else** **if** ((cmd == "V") **or** (cmd == "VER")) {

Serial.println("TCLab Firmware Version " + vers);

}

**else** **if** (cmd == "LED") {

level = max(**0.0**, min(**100.0**, pv));

iwrite = int(level \* **0.5**);

iwrite = max(**0**, min(**50**, iwrite));

ledcWrite(ledChannel, iwrite);

Serial.println(level);

}

**else** **if** (cmd == "X") {

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

Serial.println("Stop");

}

}

// check temperature **and** shut-off heaters **if** above high limit

void checkTemp(void) {

float mV = (float) analogRead(pinT1) \* **0.322265625**;

//float degC = (mV - **500.0**)/**10.0**;

float degC = mV/**10.0**;

**if** (degC >= batas\_suhu\_atas) {

Q1 = **0.0**;

Q2 = **0.0**;

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

//Serial.println("High Temp 1 (> batas\_suhu\_atas): ");

Serial.println(degC);

}

mV = (float) analogRead(pinT2) \* **0.322265625**;

//degC = (mV - **500.0**)/**10.0**;

degC = mV/**10.0**;

**if** (degC >= batas\_suhu\_atas) {

Q1 = **0.0**;

Q2 = **0.0**;

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

//Serial.println("High Temp 2 (> batas\_suhu\_atas): ");

Serial.println(degC);

}

}

// arduino startup

void setup() {

//analogReference(EXTERNAL);

Serial.begin(baud);

**while** (!Serial) {

; // wait **for** serial port to connect.

}

// configure pinQ1 PWM functionalitites

ledcSetup(Q1Channel, freq, resolutionQ1Channel);

// attach the channel to the pinQ1 to be controlled

ledcAttachPin(pinQ1, Q1Channel);

// configure pinQ2 PWM functionalitites

ledcSetup(Q2Channel, freq, resolutionQ2Channel);

// attach the channel to the pinQ2 to be controlled

ledcAttachPin(pinQ2, Q2Channel);

// configure pinLED PWM functionalitites

ledcSetup(ledChannel, freq, resolutionLedChannel);

// attach the channel to the pinLED to be controlled

ledcAttachPin(pinLED, ledChannel);

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

}

// arduino main event loop

void loop() {

parseSerial();

dispatchCommand();

checkTemp();

}

**Code itclab.py**

**import** **sys**

**import** **time**

**import** **numpy** **as** **np**

**try**:

**import** **serial**

**except**:

**import** **pip**

pip.main(['install','pyserial'])

**import** **serial**

**from** **serial.tools** **import** list\_ports

**class** **iTCLab**(object):

**def** **\_\_init\_\_**(self, port=None, baud=**115200**):

port = self.findPort()

**print**('Opening connection')

self.sp = serial.Serial(port=port, baudrate=baud, timeout=**2**)

self.sp.flushInput()

self.sp.flushOutput()

time.sleep(**3**)

**print**('iTCLab connected via Arduino on port ' + port)

**def** **findPort**(self):

found = False

**for** port **in** list(list\_ports.comports()):

# Arduino Uno

**if** port[**2**].startswith('USB VID:PID=16D0:0613'):

port = port[**0**]

found = True

# Arduino HDuino

**if** port[**2**].startswith('USB VID:PID=1A86:7523'):

port = port[**0**]

found = True

# Arduino Leonardo

**if** port[**2**].startswith('USB VID:PID=2341:8036'):

port = port[**0**]

found = True

# Arduino ESP32

**if** port[**2**].startswith('USB VID:PID=10C4:EA60'):

port = port[**0**]

found = True

# Arduino ESP32 - Tipe yg berbeda

**if** port[**2**].startswith('USB VID:PID=1A86:55D4'):

port = port[**0**]

found = True

**if** (**not** found):

**print**('Arduino COM port not found')

**print**('Please ensure that the USB cable is connected')

**print**('--- Printing Serial Ports ---')

**for** port **in** list(serial.tools.list\_ports.comports()):

**print**(port[**0**] + ' ' + port[**1**] + ' ' + port[**2**])

**print**('For Windows:')

**print**(' Open device manager, select "Ports (COM & LPT)"')

**print**(' Look for COM port of Arduino such as COM4')

**print**('For MacOS:')

**print**(' Open terminal and type: ls /dev/\*.')

**print**(' Search for /dev/tty.usbmodem\* or /dev/tty.usbserial\*. The port number is \*.')

**print**('For Linux')

**print**(' Open terminal and type: ls /dev/tty\*')

**print**(' Search for /dev/ttyUSB\* or /dev/ttyACM\*. The port number is \*.')

**print**('')

port = input('Input port: ')

# or hard-code it here

#port = 'COM3' # for Windows

#port = '/dev/tty.wchusbserial1410' # for MacOS

**return** port

**def** **stop**(self):

**return** self.read('X')

**def** **version**(self):

**return** self.read('VER')

**@property**

**def** **T1**(self):

self.\_T1 = float(self.read('T1'))

**return** self.\_T1

**@property**

**def** **T2**(self):

self.\_T2 = float(self.read('T2'))

**return** self.\_T2

**def** **LED**(self,pwm):

pwm = max(**0.0**,min(**100.0**,pwm))/**2.0**

self.write('LED',pwm)

**return** pwm

**def** **Q1**(self,pwm):

pwm = max(**0.0**,min(**100.0**,pwm))

self.write('Q1',pwm)

**return** pwm

**def** **Q2**(self,pwm):

pwm = max(**0.0**,min(**100.0**,pwm))

self.write('Q2',pwm)

**return** pwm

# save txt file with data and set point

# t = time

# u1,u2 = heaters

# y1,y2 = tempeatures

# sp1,sp2 = setpoints

**def** **save\_txt**(self,t,u1,u2,y1,y2,sp1,sp2):

data = np.vstack((t,u1,u2,y1,y2,sp1,sp2)) # vertical stack

data = data.T # transpose data

top = 'Time (sec), Heater 1 (%), Heater 2 (%), ' \

+ 'Temperature 1 (degC), Temperature 2 (degC), ' \

+ 'Set Point 1 (degC), Set Point 2 (degC)'

np.savetxt('data.txt',data,delimiter=',',header=top,comments='')

**def** **read**(self,cmd):

cmd\_str = self.build\_cmd\_str(cmd,'')

**try**:

self.sp.write(cmd\_str.encode())

self.sp.flush()

**except** **Exception**:

**return** None

**return** self.sp.readline().decode('UTF-8').replace("**\r\n**", "")

**def** **write**(self,cmd,pwm):

cmd\_str = self.build\_cmd\_str(cmd,(pwm,))

**try**:

self.sp.write(cmd\_str.encode())

self.sp.flush()

**except**:

**return** None

**return** self.sp.readline().decode('UTF-8').replace("**\r\n**", "")

**def** **build\_cmd\_str**(self,cmd, args=None):

"""

Build a command string that can be sent to the arduino.

Input:

cmd (str): the command to send to the arduino, must not

contain a % character

args (iterable): the arguments to send to the command

"""

**if** args:

args = ' '.join(map(str, args))

**else**:

args = ''

**return** "{cmd} {args}**\n**".format(cmd=cmd, args=args)

**def** **close**(self):

**try**:

self.sp.close()

**print**('Arduino disconnected successfully')

**except**:

**print**('Problems disconnecting from Arduino.')

**print**('Please unplug and reconnect Arduino.')

**return** True

**Code Deep\_PID\_iTCLab.ipynb**

**import** **itclab**

**import** **numpy** **as** **np**

**import** **time**

**import** **matplotlib.pyplot** **as** **plt**

**from** **scipy.integrate** **import** odeint

**import** **random**

# Machine Learning - Building Datasets and Model

# 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'))

# 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**]

])

# Bentuk keluaran model

model.output\_shape

# Ringkasan model

model.summary()

# Konfigurasi model

model.get\_config()

# Buat daftar semua tensor bobot

model.get\_weights()

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

model.fit(X, y,epochs=**100**, batch\_size=**1**, verbose=**1**)

######################################################

# Use this script for evaluating model predictions #

# and PID controller performance for the TCLab #

# Adjust only PID and model sections #

######################################################

######################################################

# PID Controller #

######################################################

# inputs -----------------------------------

# sp = setpoint

# pv = current temperature

# pv\_last = prior temperature

# ierr = integral error

# dt = time increment between measurements

# outputs ----------------------------------

# op = output of the PID controller

# P = proportional contribution

# I = integral contribution

# D = derivative contribution

**def** **pid**(sp,pv,pv\_last,ierr,dt):

Kc = **10.0** # K/%Heater

tauI = **50.0** # sec

tauD = **1.0** # sec

# Parameters in terms of PID coefficients

KP = Kc

KI = Kc/tauI

KD = Kc\*tauD

# ubias for controller (initial heater)

op0 = **0**

# upper and lower bounds on heater level

ophi = **100**

oplo = **0**

# calculate the error

error = sp-pv

# calculate the integral error

ierr = ierr + KI \* error \* dt

# calculate the measurement derivative

dpv = (pv - pv\_last) / dt

# calculate the PID output

P = KP \* error

I = ierr

D = -KD \* dpv

op = op0 + P + I + D

# implement anti-reset windup

**if** op < oplo **or** op > ophi:

I = I - KI \* error \* dt

# clip output

op = max(oplo,min(ophi,op))

# return the controller output and PID terms

**return** [op,P,I,D]

######################################################

# PID Controller using Deep Learning #

######################################################

# inputs -----------------------------------

# sp = setpoint

# pv = current temperature

# pv\_last = prior temperature

# ierr = integral error

# dt = time increment between measurements

# outputs ----------------------------------

# op = output of the PID controller

# P = proportional contribution

# I = integral contribution

# D = derivative contribution

**def** **pid\_dl**(sp,pv,pv\_last,ierr,dt):

# calculate the error

error = sp-pv

d\_error = sp-pv\_last

delta\_error = (error - d\_error)

outDL = model.predict(np.array([[error,delta\_error]]))

Kc = outDL[**0**,**0**]

tauI = outDL[**0**,**1**]

tauD = outDL[**0**,**2**]

# Parameters in terms of PID coefficients

KP = Kc

KI = Kc/tauI

KD = Kc\*tauD

# ubias for controller (initial heater)

op0 = **0**

# upper and lower bounds on heater level

ophi = **100**

oplo = **0**

# calculate the integral error

ierr = ierr + KI \* error \* dt

# calculate the measurement derivative

dpv = (pv - pv\_last) / dt

# calculate the PID output

P = KP \* error

I = ierr

D = -KD \* dpv

op = op0 + P + I + D

# implement anti-reset windup

**if** op < oplo **or** op > ophi:

I = I - KI \* error \* dt

# clip output

op = max(oplo,min(ophi,op))

# return the controller output and PID terms

**return** [op,P,I,D]

######################################################

# FOPDT model #

######################################################

Kp = **0.5** # degC/%

tauP = **120.0** # seconds

thetaP = **10** # seconds (integer)

Tss = **23** # degC (ambient temperature)

Qss = **0** # % heater

######################################################

# Energy balance model #

######################################################

**def** **heat**(x,t,Q):

# Parameters

Ta = **23** + **273.15** # K

U = **10.0** # W/m^2-K

m = **4.0**/**1000.0** # kg

Cp = **0.5** \* **1000.0** # J/kg-K

A = **12.0** / **100.0**\*\***2** # Area in m^2

alpha = **0.01** # W / % heater

eps = **0.9** # Emissivity

sigma = **5.67e-8** # Stefan-Boltzman

# Temperature State

T = x[**0**]

# Nonlinear Energy Balance

dTdt = (**1.0**/(m\*Cp))\*(U\*A\*(Ta-T) \

+ eps \* sigma \* A \* (Ta\*\***4** - T\*\***4**) \

+ alpha\*Q)

**return** dTdt

######################################################

# Do not adjust anything below this point #

######################################################

# Connect to Arduino

a = itclab.iTCLab()

#a.encode('utf-8').strip()#modification error

# Turn LED on

**print**('LED On')

a.LED(**100**)

# Run time in minutes

run\_time = **15.0**

# Number of cycles

loops = int(**60.0**\*run\_time)

tm = np.zeros(loops)

# Temperature

# set point (degC)

Tsp1 = np.ones(loops) \* **25.0**

Tsp1[**60**:] = **45.0**

Tsp1[**360**:] = **30.0**

Tsp1[**660**:] = **35.0**

T1 = np.ones(loops) \* a.T1 # measured T (degC)

error\_sp = np.zeros(loops)

Tsp2 = np.ones(loops) \* **23.0** # set point (degC)

T2 = np.ones(loops) \* a.T2 # measured T (degC)

# Predictions

Tp = np.ones(loops) \* a.T1

error\_eb = np.zeros(loops)

Tpl = np.ones(loops) \* a.T1

error\_fopdt = np.zeros(loops)

# impulse tests (0 - 100%)

Q1 = np.ones(loops) \* **0.0**

Q2 = np.ones(loops) \* **0.0**

**print**('Running Main Loop. Ctrl-C to end.')

**print**(' Time SP PV Q1 = P + I + D')

**print**(('{:6.1f} {:6.2f} {:6.2f} ' + \

'{:6.2f} {:6.2f} {:6.2f} {:6.2f}').format( \

tm[**0**],Tsp1[**0**],T1[**0**], \

Q1[**0**],**0.0**,**0.0**,**0.0**))

# Create plot

plt.figure(figsize=(**10**,**7**))

plt.ion()

plt.show()

# Main Loop

start\_time = time.time()

prev\_time = start\_time

# Integral error

ierr = **0.0**

**try**:

**for** i **in** range(**1**,loops):

# Sleep time

sleep\_max = **1.0**

sleep = sleep\_max - (time.time() - prev\_time)

**if** sleep>=**0.01**:

time.sleep(sleep-**0.01**)

**else**:

time.sleep(**0.01**)

# Record time and change in time

t = time.time()

dt = t - prev\_time

prev\_time = t

tm[i] = t - start\_time

# Read temperatures in Kelvin

T1[i] = a.T1

T2[i] = a.T2

# Simulate one time step with Energy Balance

Tnext = odeint(heat,Tp[i-**1**]+**273.15**,[**0**,dt],args=(Q1[i-**1**],))

Tp[i] = Tnext[**1**]-**273.15**

# Simulate one time step with linear FOPDT model

z = np.exp(-dt/tauP)

Tpl[i] = (Tpl[i-**1**]-Tss) \* z \

+ (Q1[max(**0**,i-int(thetaP)-**1**)]-Qss)\*(**1**-z)\*Kp \

+ Tss

# Calculate PID Output (Choose one of them)

# 1. Manually Choosen

# [Q1[i],P,ierr,D] = pid(Tsp1[i],T1[i],T1[i-1],ierr,dt)

# 2. Based on Deep Learning Result

[Q1[i],P,ierr,D] = pid\_dl(Tsp1[i],T1[i],T1[i-**1**],ierr,dt)

# Start setpoint error accumulation after 1 minute (60 seconds)

**if** i>=**60**:

error\_eb[i] = error\_eb[i-**1**] + abs(Tp[i]-T1[i])

error\_fopdt[i] = error\_fopdt[i-**1**] + abs(Tpl[i]-T1[i])

error\_sp[i] = error\_sp[i-**1**] + abs(Tsp1[i]-T1[i])

# Write output (0-100)

a.Q1(Q1[i])

a.Q2(**0.0**)

# Print line of data

**print**(('{:6.1f} {:6.2f} {:6.2f} ' + \

'{:6.2f} {:6.2f} {:6.2f} {:6.2f}').format( \

tm[i],Tsp1[i],T1[i], \

Q1[i],P,ierr,D))

# Plot

plt.clf()

ax=plt.subplot(**4**,**1**,**1**)

ax.grid()

plt.plot(tm[**0**:i],T1[**0**:i],'r.',label=r'$T\_1$ measured')

plt.plot(tm[**0**:i],Tsp1[**0**:i],'k--',label=r'$T\_1$ set point')

plt.ylabel('Temperature (degC)')

plt.legend(loc=**2**)

ax=plt.subplot(**4**,**1**,**2**)

ax.grid()

plt.plot(tm[**0**:i],Q1[**0**:i],'b-',label=r'$Q\_1$')

plt.ylabel('Heater')

plt.legend(loc='best')

ax=plt.subplot(**4**,**1**,**3**)

ax.grid()

plt.plot(tm[**0**:i],T1[**0**:i],'r.',label=r'$T\_1$ measured')

plt.plot(tm[**0**:i],Tp[**0**:i],'k-',label=r'$T\_1$ energy balance')

plt.plot(tm[**0**:i],Tpl[**0**:i],'g-',label=r'$T\_1$ linear model')

plt.ylabel('Temperature (degC)')

plt.legend(loc=**2**)

ax=plt.subplot(**4**,**1**,**4**)

ax.grid()

plt.plot(tm[**0**:i],error\_sp[**0**:i],'r-',label='Set Point Error')

plt.plot(tm[**0**:i],error\_eb[**0**:i],'k-',label='Energy Balance Error')

plt.plot(tm[**0**:i],error\_fopdt[**0**:i],'g-',label='Linear Model Error')

plt.ylabel('Cumulative Error')

plt.legend(loc='best')

plt.xlabel('Time (sec)')

plt.draw()

plt.pause(**0.05**)

# Turn off heaters

a.Q1(**0**)

a.Q2(**0**)

# Save figure

plt.savefig('test\_PID\_dl.png')

# Allow user to end loop with Ctrl-C

**except** **KeyboardInterrupt**:

# Disconnect from Arduino

a.Q1(**0**)

a.Q2(**0**)

**print**('Shutting down')

a.close()

plt.savefig('test\_PID\_dl.png')

# Make sure serial connection still closes when there's an error

**except**:

# Disconnect from Arduino

a.Q1(**0**)

a.Q2(**0**)

**print**('Error: Shutting down')

a.close()

plt.savefig('test\_PID\_dl.png')

**raise**

a.close()

## iTCLab 14

**Code Deep\_PID\_iTCLab\_IoT.ino**

/\*

iTCLab Internet-Based Temperature Control Lab Firmware

Jeffrey Kantor, Initial Version

John Hedengren, Modified

Oct **2017**

Basuki Rahmat, Modified

April **2022**

This firmware **is** loaded into the Internet-Based Temperature Control Laboratory ESP32 to

provide a high level interface to the Internet-Based Temperature Control Lab. The firmware

scans the serial port looking **for** case-insensitive commands:

Q1 set Heater **1**, range **0** to **100**% subject to limit (**0**-**255** int)

Q2 set Heater **2**, range **0** to **100**% subject to limit (**0**-**255** int)

T1 get Temperature T1, returns deg C **as** string

T2 get Temperature T2, returns dec C **as** string

VER get firmware version string

X stop, enter sleep mode

Limits on the heater can be configured **with** the constants below.

\*/

#include <Arduino.h>

// constants

const String vers = "1.04"; // version of this firmware

const int baud = **115200**; // serial baud rate

const char sp = ' '; // command separator

const char nl = '**\n**'; // command terminator

// pin numbers corresponding to signals on the iTCLab Shield

const int pinT1 = **34**; // T1

const int pinT2 = **35**; // T2

const int pinQ1 = **32**; // Q1

const int pinQ2 = **33**; // Q2

const int pinLED = **26**; // LED

// setting PWM properties

const int freq = **5000**; //**5000**

const int ledChannel = **0**;

const int Q1Channel = **1**;

const int Q2Channel = **2**;

const int resolutionLedChannel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ1Channel = **8**; //Resolution **8**, **10**, **12**, **15**

const int resolutionQ2Channel = **8**; //Resolution **8**, **10**, **12**, **15**

const double batas\_suhu\_atas = **59**;

// **global** variables

char Buffer[**64**]; // buffer **for** parsing serial input

String cmd; // command

double pv = **0**; // pin value

float level; // LED level (**0**-**100**%)

double Q1 = **0**; // value written to Q1 pin

double Q2 = **0**; // value written to Q2 pin

int iwrite = **0**; // integer value **for** writing

float dwrite = **0**; // float value **for** writing

int n = **10**; // number of samples **for** each temperature measurement

void parseSerial(void) {

int ByteCount = Serial.readBytesUntil(nl,Buffer,sizeof(Buffer));

String read\_ = String(Buffer);

memset(Buffer,**0**,sizeof(Buffer));

// separate command **from** **associated** **data**

int idx = read\_.indexOf(sp);

cmd = read\_.substring(**0**,idx);

cmd.trim();

cmd.toUpperCase();

// extract data. toInt() returns **0** on error

String data = read\_.substring(idx+**1**);

data.trim();

pv = data.toFloat();

}

// Q1\_max = **100**%

// Q2\_max = **100**%

void dispatchCommand(void) {

**if** (cmd == "Q1") {

Q1 = max(**0.0**, min(**25.0**, pv));

iwrite = int(Q1 \* **2.0**); // **10.**? max

iwrite = max(**0**, min(**255**, iwrite));

ledcWrite(Q1Channel,iwrite);

Serial.println(Q1);

}

**else** **if** (cmd == "Q2") {

Q2 = max(**0.0**, min(**25.0**, pv));

iwrite = int(Q2 \* **2.0**); // **10.**? max

iwrite = max(**0**, min(**255**, iwrite));

ledcWrite(Q2Channel,iwrite);

Serial.println(Q2);

}

**else** **if** (cmd == "T1") {

float mV = **0.0**;

float degC = **0.0**;

**for** (int i = **0**; i < n; i++) {

mV = (float) analogRead(pinT1) \* **0.322265625**;

degC = degC + mV/**10.0**;

}

degC = degC / float(n);

Serial.println(degC);

}

**else** **if** (cmd == "T2") {

float mV = **0.0**;

float degC = **0.0**;

**for** (int i = **0**; i < n; i++) {

mV = (float) analogRead(pinT2) \* **0.322265625**;

degC = degC + mV/**10.0**;

}

degC = degC / float(n);

Serial.println(degC);

}

**else** **if** ((cmd == "V") **or** (cmd == "VER")) {

Serial.println("TCLab Firmware Version " + vers);

}

**else** **if** (cmd == "LED") {

level = max(**0.0**, min(**100.0**, pv));

iwrite = int(level \* **0.5**);

iwrite = max(**0**, min(**50**, iwrite));

ledcWrite(ledChannel, iwrite);

Serial.println(level);

}

**else** **if** (cmd == "X") {

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

Serial.println("Stop");

}

}

// check temperature **and** shut-off heaters **if** above high limit

void checkTemp(void) {

float mV = (float) analogRead(pinT1) \* **0.322265625**;

//float degC = (mV - **500.0**)/**10.0**;

float degC = mV/**10.0**;

**if** (degC >= batas\_suhu\_atas) {

Q1 = **0.0**;

Q2 = **0.0**;

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

//Serial.println("High Temp 1 (> batas\_suhu\_atas): ");

Serial.println(degC);

}

mV = (float) analogRead(pinT2) \* **0.322265625**;

//degC = (mV - **500.0**)/**10.0**;

degC = mV/**10.0**;

**if** (degC >= batas\_suhu\_atas) {

Q1 = **0.0**;

Q2 = **0.0**;

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

//Serial.println("High Temp 2 (> batas\_suhu\_atas): ");

Serial.println(degC);

}

}

// arduino startup

void setup() {

//analogReference(EXTERNAL);

Serial.begin(baud);

**while** (!Serial) {

; // wait **for** serial port to connect.

}

// configure pinQ1 PWM functionalitites

ledcSetup(Q1Channel, freq, resolutionQ1Channel);

// attach the channel to the pinQ1 to be controlled

ledcAttachPin(pinQ1, Q1Channel);

// configure pinQ2 PWM functionalitites

ledcSetup(Q2Channel, freq, resolutionQ2Channel);

// attach the channel to the pinQ2 to be controlled

ledcAttachPin(pinQ2, Q2Channel);

// configure pinLED PWM functionalitites

ledcSetup(ledChannel, freq, resolutionLedChannel);

// attach the channel to the pinLED to be controlled

ledcAttachPin(pinLED, ledChannel);

ledcWrite(Q1Channel,**0**);

ledcWrite(Q2Channel,**0**);

}

// arduino main event loop

void loop() {

parseSerial();

dispatchCommand();

checkTemp();

}

**Code itclab.py**

**import** **sys**

**import** **time**

**import** **numpy** **as** **np**

**try**:

**import** **serial**

**except**:

**import** **pip**

pip.main(['install','pyserial'])

**import** **serial**

**from** **serial.tools** **import** list\_ports

**class** **iTCLab**(object):

**def** **\_\_init\_\_**(self, port=None, baud=**115200**):

port = self.findPort()

**print**('Opening connection')

self.sp = serial.Serial(port=port, baudrate=baud, timeout=**2**)

self.sp.flushInput()

self.sp.flushOutput()

time.sleep(**3**)

**print**('iTCLab connected via Arduino on port ' + port)

**def** **findPort**(self):

found = False

**for** port **in** list(list\_ports.comports()):

# Arduino Uno

**if** port[**2**].startswith('USB VID:PID=16D0:0613'):

port = port[**0**]

found = True

# Arduino HDuino

**if** port[**2**].startswith('USB VID:PID=1A86:7523'):

port = port[**0**]

found = True

# Arduino Leonardo

**if** port[**2**].startswith('USB VID:PID=2341:8036'):

port = port[**0**]

found = True

# Arduino ESP32

**if** port[**2**].startswith('USB VID:PID=10C4:EA60'):

port = port[**0**]

found = True

# Arduino ESP32 - Tipe yg berbeda

**if** port[**2**].startswith('USB VID:PID=1A86:55D4'):

port = port[**0**]

found = True

**if** (**not** found):

**print**('Arduino COM port not found')

**print**('Please ensure that the USB cable is connected')

**print**('--- Printing Serial Ports ---')

**for** port **in** list(serial.tools.list\_ports.comports()):

**print**(port[**0**] + ' ' + port[**1**] + ' ' + port[**2**])

**print**('For Windows:')

**print**(' Open device manager, select "Ports (COM & LPT)"')

**print**(' Look for COM port of Arduino such as COM4')

**print**('For MacOS:')

**print**(' Open terminal and type: ls /dev/\*.')

**print**(' Search for /dev/tty.usbmodem\* or /dev/tty.usbserial\*. The port number is \*.')

**print**('For Linux')

**print**(' Open terminal and type: ls /dev/tty\*')

**print**(' Search for /dev/ttyUSB\* or /dev/ttyACM\*. The port number is \*.')

**print**('')

port = input('Input port: ')

# or hard-code it here

#port = 'COM3' # for Windows

#port = '/dev/tty.wchusbserial1410' # for MacOS

**return** port

**def** **stop**(self):

**return** self.read('X')

**def** **version**(self):

**return** self.read('VER')

**@property**

**def** **T1**(self):

self.\_T1 = float(self.read('T1'))

**return** self.\_T1

**@property**

**def** **T2**(self):

self.\_T2 = float(self.read('T2'))

**return** self.\_T2

**def** **LED**(self,pwm):

pwm = max(**0.0**,min(**100.0**,pwm))/**2.0**

self.write('LED',pwm)

**return** pwm

**def** **Q1**(self,pwm):

pwm = max(**0.0**,min(**100.0**,pwm))

self.write('Q1',pwm)

**return** pwm

**def** **Q2**(self,pwm):

pwm = max(**0.0**,min(**100.0**,pwm))

self.write('Q2',pwm)

**return** pwm

# save txt file with data and set point

# t = time

# u1,u2 = heaters

# y1,y2 = tempeatures

# sp1,sp2 = setpoints

**def** **save\_txt**(self,t,u1,u2,y1,y2,sp1,sp2):

data = np.vstack((t,u1,u2,y1,y2,sp1,sp2)) # vertical stack

data = data.T # transpose data

top = 'Time (sec), Heater 1 (%), Heater 2 (%), ' \

+ 'Temperature 1 (degC), Temperature 2 (degC), ' \

+ 'Set Point 1 (degC), Set Point 2 (degC)'

np.savetxt('data.txt',data,delimiter=',',header=top,comments='')

**def** **read**(self,cmd):

cmd\_str = self.build\_cmd\_str(cmd,'')

**try**:

self.sp.write(cmd\_str.encode())

self.sp.flush()

**except** **Exception**:

**return** None

**return** self.sp.readline().decode('UTF-8').replace("**\r\n**", "")

**def** **write**(self,cmd,pwm):

cmd\_str = self.build\_cmd\_str(cmd,(pwm,))

**try**:

self.sp.write(cmd\_str.encode())

self.sp.flush()

**except**:

**return** None

**return** self.sp.readline().decode('UTF-8').replace("**\r\n**", "")

**def** **build\_cmd\_str**(self,cmd, args=None):

"""

Build a command string that can be sent to the arduino.

Input:

cmd (str): the command to send to the arduino, must not

contain a % character

args (iterable): the arguments to send to the command

"""

**if** args:

args = ' '.join(map(str, args))

**else**:

args = ''

**return** "{cmd} {args}**\n**".format(cmd=cmd, args=args)

**def** **close**(self):

**try**:

self.sp.close()

**print**('Arduino disconnected successfully')

**except**:

**print**('Problems disconnecting from Arduino.')

**print**('Please unplug and reconnect Arduino.')

**return** True

**Code Deep\_PID\_iTCLab\_IoT.ipynb**

**import** **itclab**

**import** **numpy** **as** **np**

**import** **time**

**import** **matplotlib.pyplot** **as** **plt**

**from** **scipy.integrate** **import** odeint

**import** **random**

**from** **paho.mqtt** **import** client **as** mqtt\_client

# Machine Learning - Building Datasets and Model

# 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'))

# 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**]

])

# Bentuk keluaran model

model.output\_shape

# Ringkasan model

model.summary()

# Konfigurasi model

model.get\_config()

# Buat daftar semua tensor bobot

model.get\_weights()

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

model.fit(X, y,epochs=**10**, batch\_size=**1**, verbose=**1**)

######################################################

# Use this script for evaluating model predictions #

# and PID controller performance for the TCLab #

# Adjust only PID and model sections #

######################################################

######################################################

# PID Controller #

######################################################

# inputs -----------------------------------

# sp = setpoint

# pv = current temperature

# pv\_last = prior temperature

# ierr = integral error

# dt = time increment between measurements

# outputs ----------------------------------

# op = output of the PID controller

# P = proportional contribution

# I = integral contribution

# D = derivative contribution

**def** **pid**(sp,pv,pv\_last,ierr,dt):

Kc = **10.0** # K/%Heater

tauI = **50.0** # sec

tauD = **1.0** # sec

# Parameters in terms of PID coefficients

KP = Kc

KI = Kc/tauI

KD = Kc\*tauD

# ubias for controller (initial heater)

op0 = **0**

# upper and lower bounds on heater level

ophi = **100**

oplo = **0**

# calculate the error

error = sp-pv

# calculate the integral error

ierr = ierr + KI \* error \* dt

# calculate the measurement derivative

dpv = (pv - pv\_last) / dt

# calculate the PID output

P = KP \* error

I = ierr

D = -KD \* dpv

op = op0 + P + I + D

# implement anti-reset windup

**if** op < oplo **or** op > ophi:

I = I - KI \* error \* dt

# clip output

op = max(oplo,min(ophi,op))

# return the controller output and PID terms

**return** [op,P,I,D]

######################################################

# PID Controller using Deep Learning #

######################################################

# inputs -----------------------------------

# sp = setpoint

# pv = current temperature

# pv\_last = prior temperature

# ierr = integral error

# dt = time increment between measurements

# outputs ----------------------------------

# op = output of the PID controller

# P = proportional contribution

# I = integral contribution

# D = derivative contribution

**def** **pid\_dl**(sp,pv,pv\_last,ierr,dt):

# calculate the error

error = sp-pv

d\_error = sp-pv\_last

delta\_error = (error - d\_error)

outDL = model.predict(np.array([[error,delta\_error]]))

Kc = outDL[**0**,**0**]

tauI = outDL[**0**,**1**]

tauD = outDL[**0**,**2**]

# Parameters in terms of PID coefficients

KP = Kc

KI = Kc/tauI

KD = Kc\*tauD

# ubias for controller (initial heater)

op0 = **0**

# upper and lower bounds on heater level

ophi = **100**

oplo = **0**

# calculate the integral error

ierr = ierr + KI \* error \* dt

# calculate the measurement derivative

dpv = (pv - pv\_last) / dt

# calculate the PID output

P = KP \* error

I = ierr

D = -KD \* dpv

op = op0 + P + I + D

# implement anti-reset windup

**if** op < oplo **or** op > ophi:

I = I - KI \* error \* dt

# clip output

op = max(oplo,min(ophi,op))

# return the controller output and PID terms

**return** [op,P,I,D]

######################################################

# FOPDT model #

######################################################

Kp = **0.5** # degC/%

tauP = **120.0** # seconds

thetaP = **10** # seconds (integer)

Tss = **23** # degC (ambient temperature)

Qss = **0** # % heater

######################################################

# Energy balance model #

######################################################

**def** **heat**(x,t,Q):

# Parameters

Ta = **23** + **273.15** # K

U = **10.0** # W/m^2-K

m = **4.0**/**1000.0** # kg

Cp = **0.5** \* **1000.0** # J/kg-K

A = **12.0** / **100.0**\*\***2** # Area in m^2

alpha = **0.01** # W / % heater

eps = **0.9** # Emissivity

sigma = **5.67e-8** # Stefan-Boltzman

# Temperature State

T = x[**0**]

# Nonlinear Energy Balance

dTdt = (**1.0**/(m\*Cp))\*(U\*A\*(Ta-T) \

+ eps \* sigma \* A \* (Ta\*\***4** - T\*\***4**) \

+ alpha\*Q)

**return** dTdt

# Connect to MQTT Broker for Monitoring

broker = 'broker.hivemq.com'

port = **1883**

client\_id = f'python-mqtt-{random.randint(0, 1000)}'

**def** **connect\_mqtt**():

**def** **on\_connect**(client, userdata, flags, rc):

**if** rc == **0**:

**print**("Connected to MQTT Broker!")

**else**:

**print**("Failed to connect, return code %d**\n**", rc)

client = mqtt\_client.Client(client\_id)

client.on\_connect = on\_connect

client.connect(broker, port)

**return** client

client = connect\_mqtt()

client.loop\_start()

######################################################

# Do not adjust anything below this point #

######################################################

# Connect to Arduino

a = itclab.iTCLab()

#a.encode('utf-8').strip()#modification error

# Turn LED on

**print**('LED On')

a.LED(**100**)

# Run time in minutes

run\_time = **15.0**

# Number of cycles

loops = int(**60.0**\*run\_time)

tm = np.zeros(loops)

# Temperature

# set point (degC)

Tsp1 = np.ones(loops) \* **25.0**

Tsp1[**60**:] = **45.0**

Tsp1[**360**:] = **30.0**

Tsp1[**660**:] = **35.0**

T1 = np.ones(loops) \* a.T1 # measured T (degC)

error\_sp = np.zeros(loops)

Tsp2 = np.ones(loops) \* **23.0** # set point (degC)

T2 = np.ones(loops) \* a.T2 # measured T (degC)

# Predictions

Tp = np.ones(loops) \* a.T1

error\_eb = np.zeros(loops)

Tpl = np.ones(loops) \* a.T1

error\_fopdt = np.zeros(loops)

# impulse tests (0 - 100%)

Q1 = np.ones(loops) \* **0.0**

Q2 = np.ones(loops) \* **0.0**

**print**('Running Main Loop. Ctrl-C to end.')

**print**(' Time SP PV Q1 = P + I + D')

**print**(('{:6.1f} {:6.2f} {:6.2f} ' + \

'{:6.2f} {:6.2f} {:6.2f} {:6.2f}').format( \

tm[**0**],Tsp1[**0**],T1[**0**], \

Q1[**0**],**0.0**,**0.0**,**0.0**))

# Create plot

plt.figure(figsize=(**10**,**7**))

plt.ion()

plt.show()

# Main Loop

start\_time = time.time()

prev\_time = start\_time

# Integral error

ierr = **0.0**

**try**:

**for** i **in** range(**1**,loops):

# Sleep time

sleep\_max = **1.0**

sleep = sleep\_max - (time.time() - prev\_time)

**if** sleep>=**0.01**:

time.sleep(sleep-**0.01**)

**else**:

time.sleep(**0.01**)

# Record time and change in time

t = time.time()

dt = t - prev\_time

prev\_time = t

tm[i] = t - start\_time

# Read temperatures in Kelvin

T1[i] = a.T1

T2[i] = a.T2

# Simulate one time step with Energy Balance

Tnext = odeint(heat,Tp[i-**1**]+**273.15**,[**0**,dt],args=(Q1[i-**1**],))

Tp[i] = Tnext[**1**]-**273.15**

# Simulate one time step with linear FOPDT model

z = np.exp(-dt/tauP)

Tpl[i] = (Tpl[i-**1**]-Tss) \* z \

+ (Q1[max(**0**,i-int(thetaP)-**1**)]-Qss)\*(**1**-z)\*Kp \

+ Tss

# Calculate PID Output (Choose one of them)

# 1. Manually Choosen

# [Q1[i],P,ierr,D] = pid(Tsp1[i],T1[i],T1[i-1],ierr,dt)

# 2. Based on Deep Learning Result

[Q1[i],P,ierr,D] = pid\_dl(Tsp1[i],T1[i],T1[i-**1**],ierr,dt)

# Start setpoint error accumulation after 1 minute (60 seconds)

**if** i>=**60**:

error\_eb[i] = error\_eb[i-**1**] + abs(Tp[i]-T1[i])

error\_fopdt[i] = error\_fopdt[i-**1**] + abs(Tpl[i]-T1[i])

error\_sp[i] = error\_sp[i-**1**] + abs(Tsp1[i]-T1[i])

# Write output (0-100)

a.Q1(Q1[i])

a.Q2(**0.0**)

# Print line of data

**print**(('{:6.1f} {:6.2f} {:6.2f} ' + \

'{:6.2f} {:6.2f} {:6.2f} {:6.2f}').format( \

tm[i],Tsp1[i],T1[i], \

Q1[i],P,ierr,D))

# Publish data to MQTT Broker

pub\_sp = client.publish('SetPoint', Tsp1[i])

pub\_pv1 = client.publish('Suhu1', T1[i])

pub\_op = client.publish('Nilai\_op', Q1[i])

# Plot

plt.clf()

ax=plt.subplot(**4**,**1**,**1**)

ax.grid()

plt.plot(tm[**0**:i],T1[**0**:i],'r.',label=r'$T\_1$ measured')

plt.plot(tm[**0**:i],Tsp1[**0**:i],'k--',label=r'$T\_1$ set point')

plt.ylabel('Temperature (degC)')

plt.legend(loc=**2**)

ax=plt.subplot(**4**,**1**,**2**)

ax.grid()

plt.plot(tm[**0**:i],Q1[**0**:i],'b-',label=r'$Q\_1$')

plt.ylabel('Heater')

plt.legend(loc='best')

ax=plt.subplot(**4**,**1**,**3**)

ax.grid()

plt.plot(tm[**0**:i],T1[**0**:i],'r.',label=r'$T\_1$ measured')

plt.plot(tm[**0**:i],Tp[**0**:i],'k-',label=r'$T\_1$ energy balance')

plt.plot(tm[**0**:i],Tpl[**0**:i],'g-',label=r'$T\_1$ linear model')

plt.ylabel('Temperature (degC)')

plt.legend(loc=**2**)

ax=plt.subplot(**4**,**1**,**4**)

ax.grid()

plt.plot(tm[**0**:i],error\_sp[**0**:i],'r-',label='Set Point Error')

plt.plot(tm[**0**:i],error\_eb[**0**:i],'k-',label='Energy Balance Error')

plt.plot(tm[**0**:i],error\_fopdt[**0**:i],'g-',label='Linear Model Error')

plt.ylabel('Cumulative Error')

plt.legend(loc='best')

plt.xlabel('Time (sec)')

plt.draw()

plt.pause(**0.05**)

# Turn off heaters

a.Q1(**0**)

a.Q2(**0**)

# Save figure

plt.savefig('test\_PID\_dl.png')

# Allow user to end loop with Ctrl-C

**except** **KeyboardInterrupt**:

# Disconnect from Arduino

a.Q1(**0**)

a.Q2(**0**)

**print**('Shutting down')

a.close()

plt.savefig('test\_PID\_dl.png')

# Make sure serial connection still closes when there's an error

**except**:

# Disconnect from Arduino

a.Q1(**0**)

a.Q2(**0**)

**print**('Error: Shutting down')

a.close()

plt.savefig('test\_PID\_dl.png')

**raise**

a.close()