https://www.halvorsen.blog

Part II

Control Systems with Python

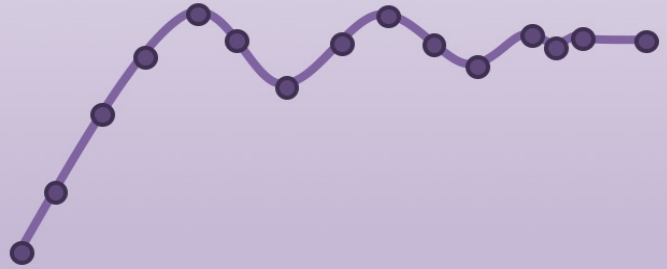
Exemplified by using a Small-scale Industrial Process called “Air Heater System”

Hans-Petter Halvorsen

Free Textbook with lots of Practical Examples

Python for Control Engineering

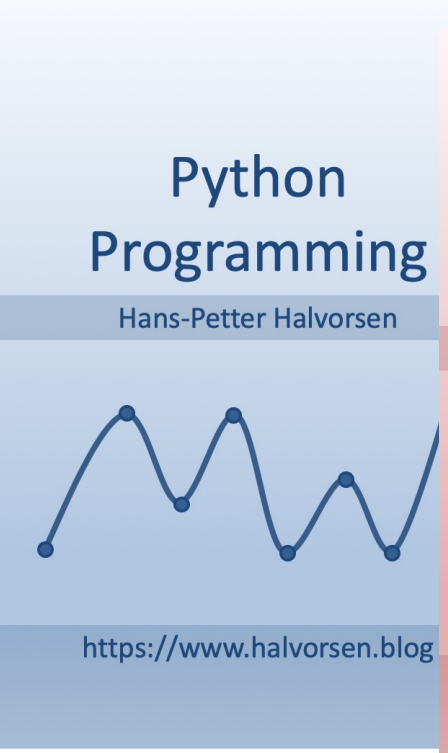
Hans-Petter Halvorsen

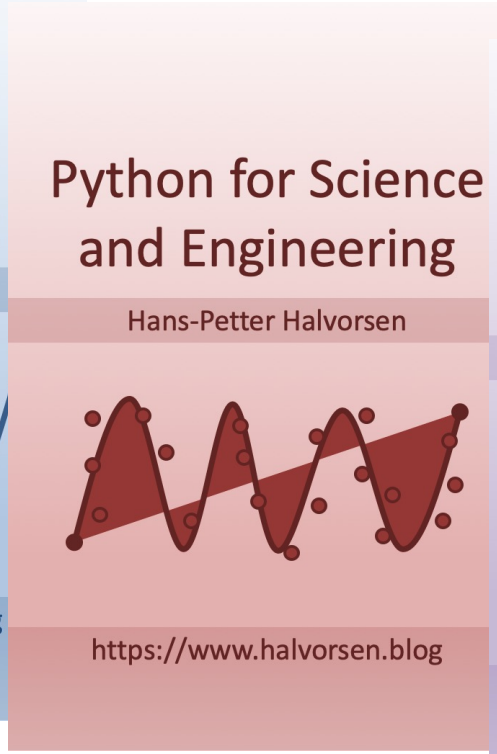


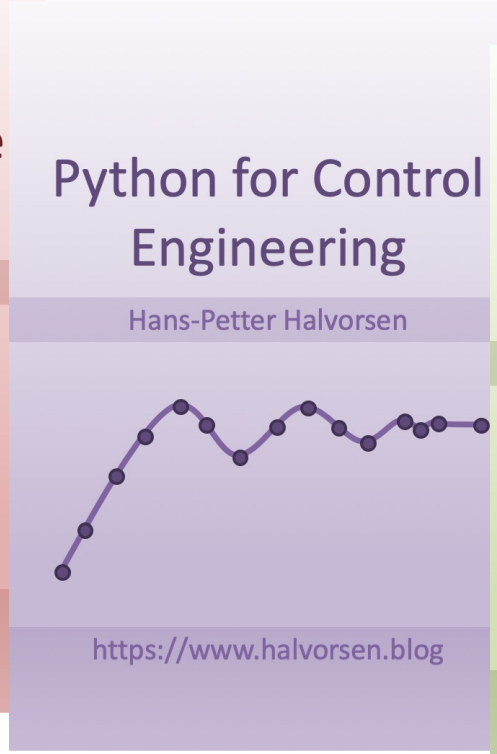
https://www.halvorsen.blog

https://www.halvorsen.blog/documents/programming/python/

Additional Python Resources







https://www.halvorsen.blog/documents/programming/python/

Control System with Python – Part 1

In Part 1 we did the following:

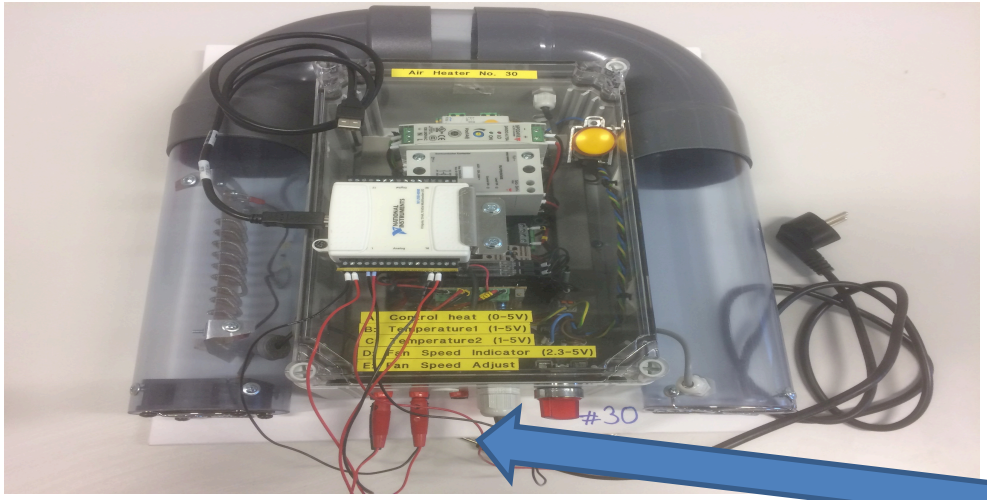
- We use a Mathematical Model of a Small-scale Industrial Process (called “Air Heater“)
- We made a Process Simulator using a Discrete version of the Mathematical Model
- We made a Control System where we used the Simulator (Model of the System)

YouTube Video Part 1:

XXXXXX (Coming soon…)

What's Next?

Control the Real System



Small-scale Industrial Process

I/O Module

This will be demonstrated and explained in this Tutorial

Contents

- How can we use NI Hardware with Python?
  - NI USN-6008 will be used in this Tutorial
- What is DAQ and I/O Modules?
- NI-DAQmx
- “nidaqmx“ Python API/Library
- Control System controlling the real Small-scale Industrial Process (called “Air Heater“)

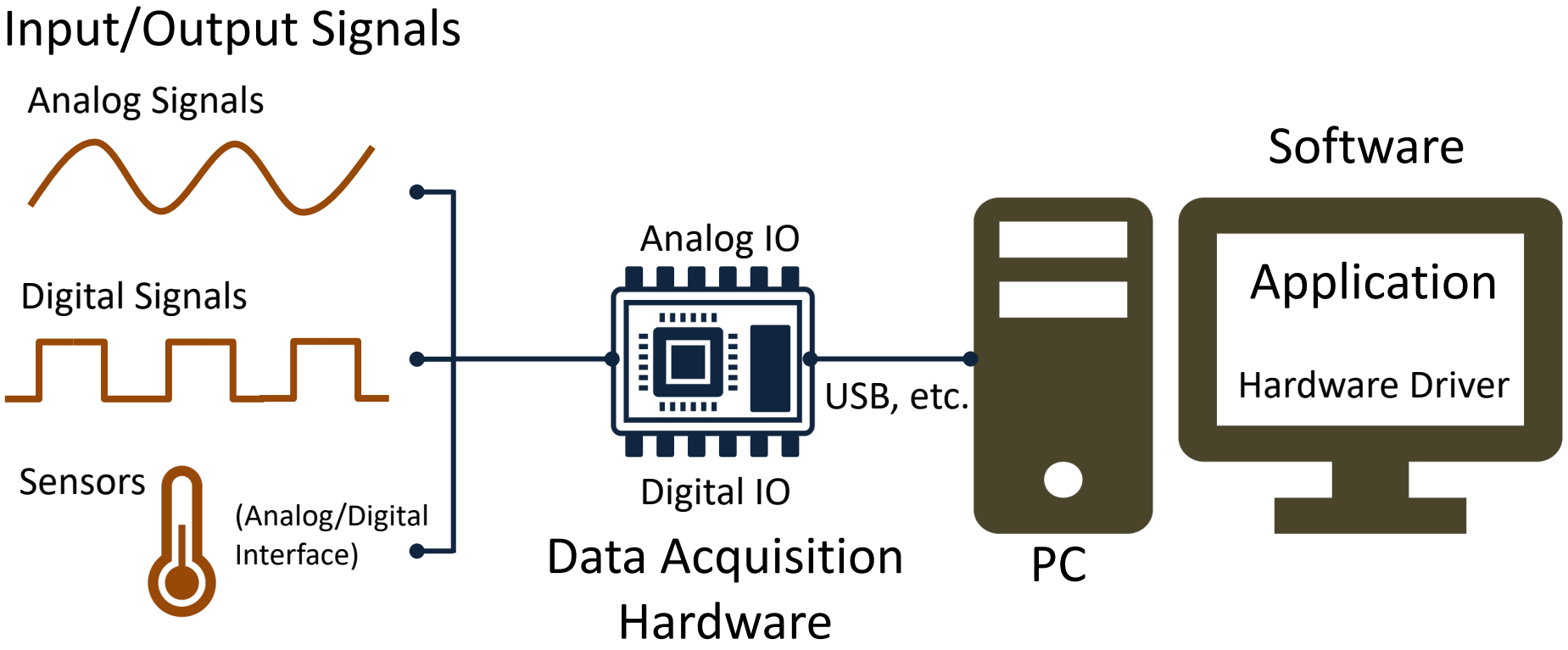
DAQ

Hans-Petter Halvorsen

Data Acquisition (DAQ)

- To read sensor data you typically need a DAQ (Data Acquisition) device connected to you PC
- You can also use devices like Arduino , Raspberry Pi, etc.
- In all cases you will typically need to install a driver from the vendor of the DAQ device or the sensor you are using

DAQ System


Input/Output Signals
Analog Signals
Digital Signals
Sensors
(Analog/Digital Interface)
Analog IO
Digital IO
Data Acquisition Hardware
USB, etc.
PC
Software
Application
Hardware Driver

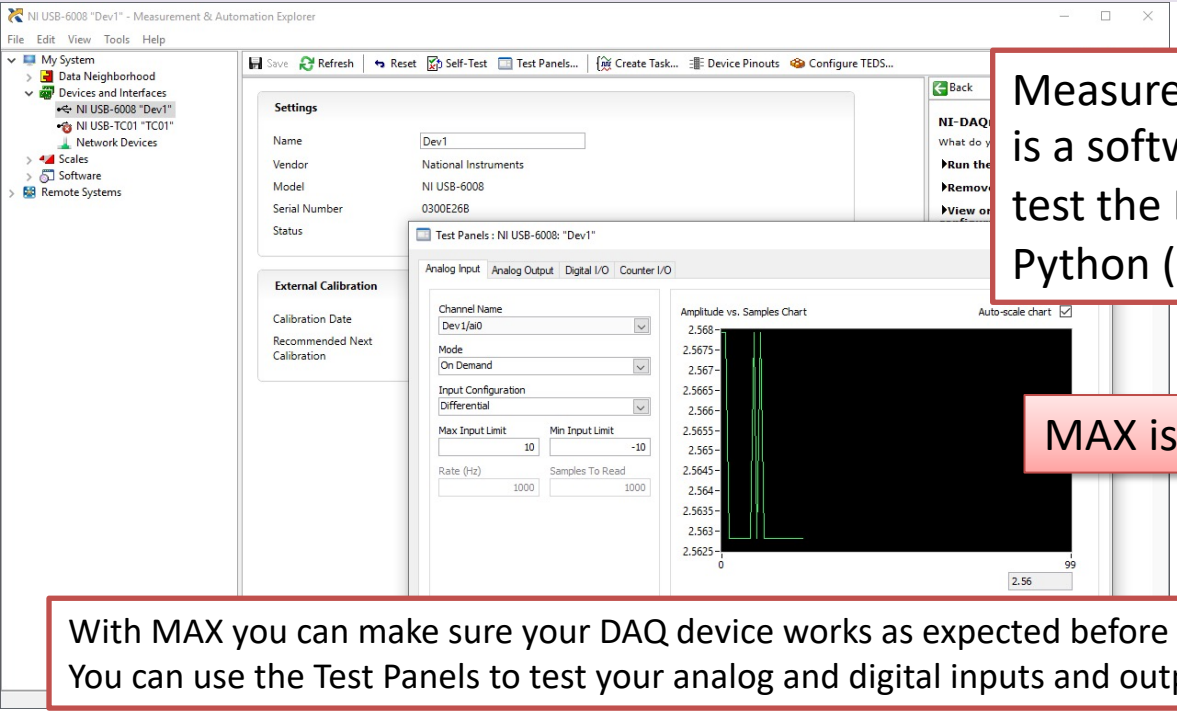
NI-DAQmx

- NI-DAQmx is the software you use to communicate with and control your NI data acquisition (DAQ) device.
- NI-DAQmx supports only the Windows operating system.
- Typically you use LabVIEW in combination with NI DAQ Hardware, but the NI-DAQmx can also be used from C, C#, Python, etc.
- The NI-DAQmx Driver is Free!
- Visit the ni.com/downloads to download the latest version of NI-DAQmx

NI DAQ Device with Python

How to use a NI DAQ Device with Python

Measurement & Automation Explorer (MAX)



Measurement & Automation Explorer (MAX) is a software you can use to configure and test the DAQ device before you use it in Python (or other programming languages).

MAX is included with NI-DAQmx software

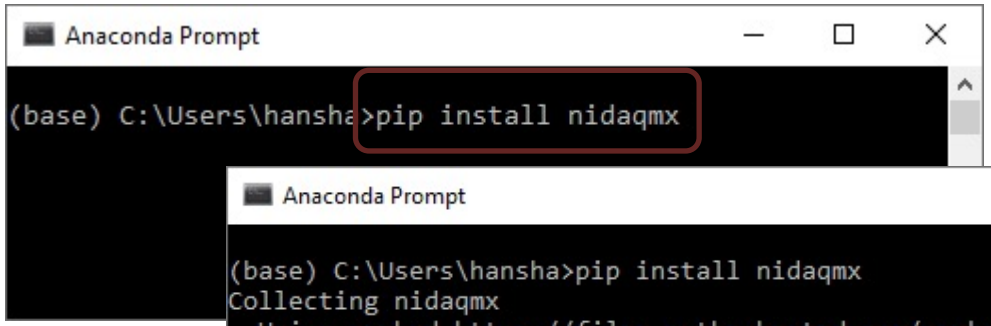
With MAX you can make sure your DAQ device works as expected before you start using it in your Python program. You can use the Test Panels to test your analog and digital inputs and outputs channels.

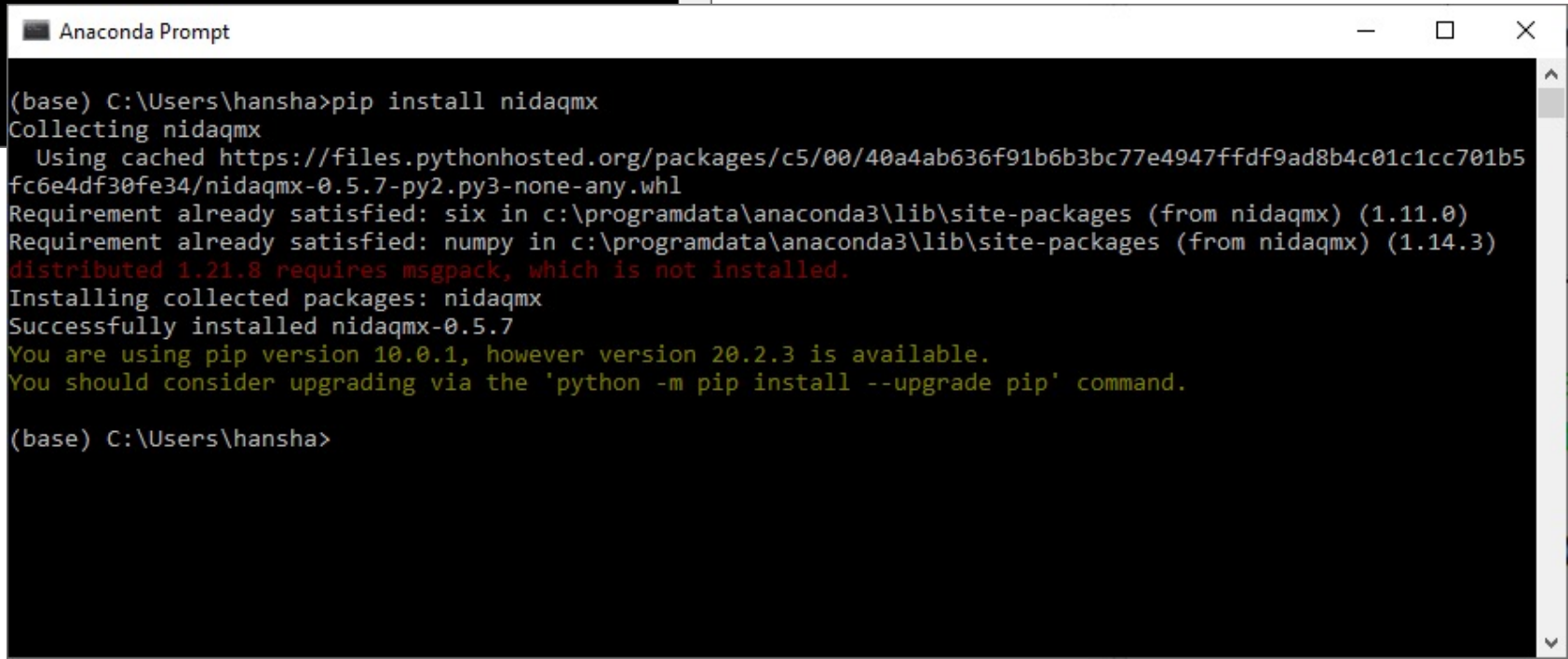
nidaqmx Python API

- Python Library/API for Communication with NI DAQmx Driver
- Running **nidaqmx** requires NI-DAQmx or NI-DAQmx Runtime
- Visit the ni.com/downloads to download the latest version of NI-DAQmx
- nidaqmx can be installed with **pip**:
  `pip install nidaqmx`
- https://github.com/ni/nidaqmx-python

nidaqmx Python Package

Installation





DAQ Tutorials

Lots of DAQ Tutorials have already been made.
Some Examples:

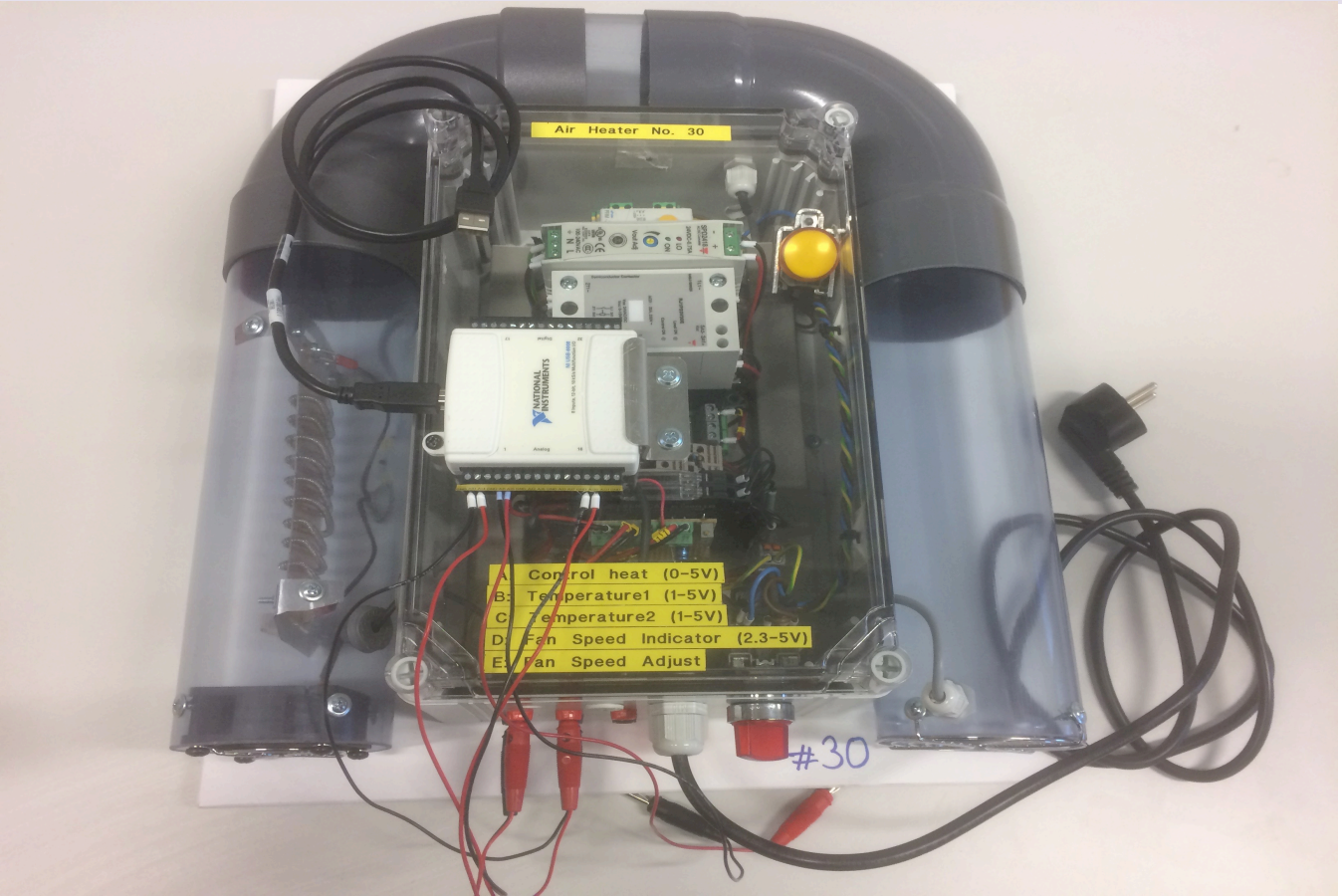
- DAQ with Python
  https://youtu.be/HkybKgnF0sQ
- DAQ with I/O Modules in Python
  https://youtu.be/umXMrr6Z0Og
- Sensors and Actuators with Python
  https://youtu.be/8lfZkDIQ6NY

https://www.halvorsen.blog/documents/programming/python/

Hardware

Hans-Petter Halvorsen

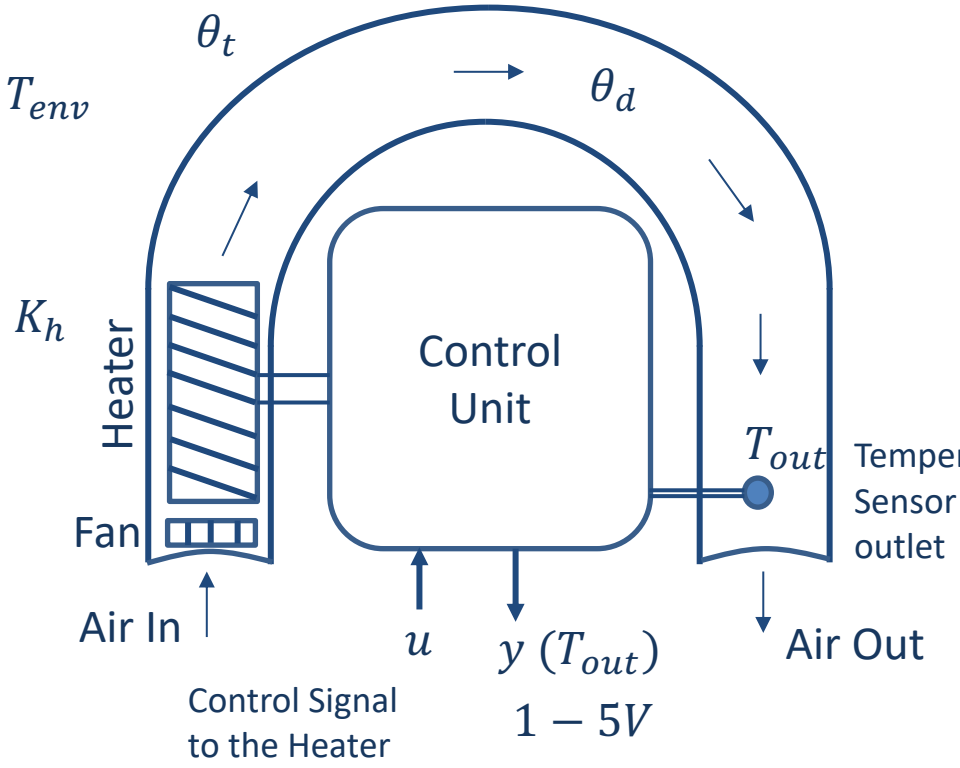
Small-scale Industrial Process



The Air Heater is a small-scale laboratory process suitable for learning about control systems

We want to implement and control the Air Heater system in Python. We start by implementing a control system using a mathematical model of the system

Air Heater System

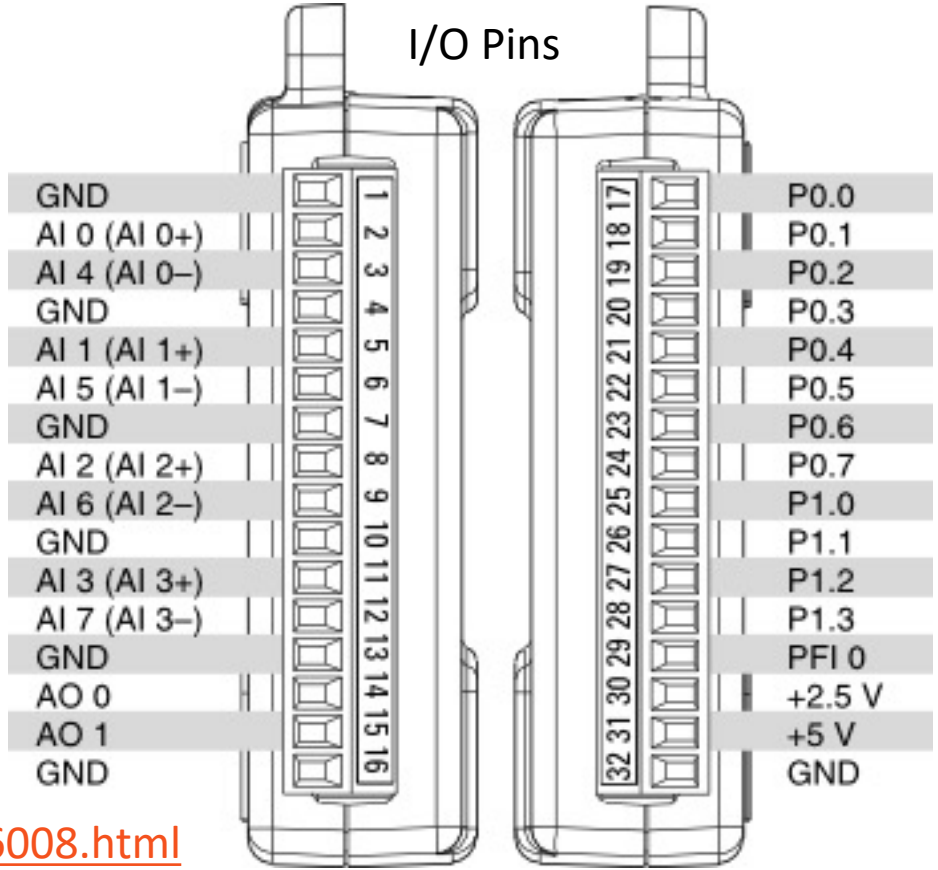


Aim:

Control the Temperature on the outlet (T_{out})

NI USB-6008

We will use NI USB-6008 in our examples

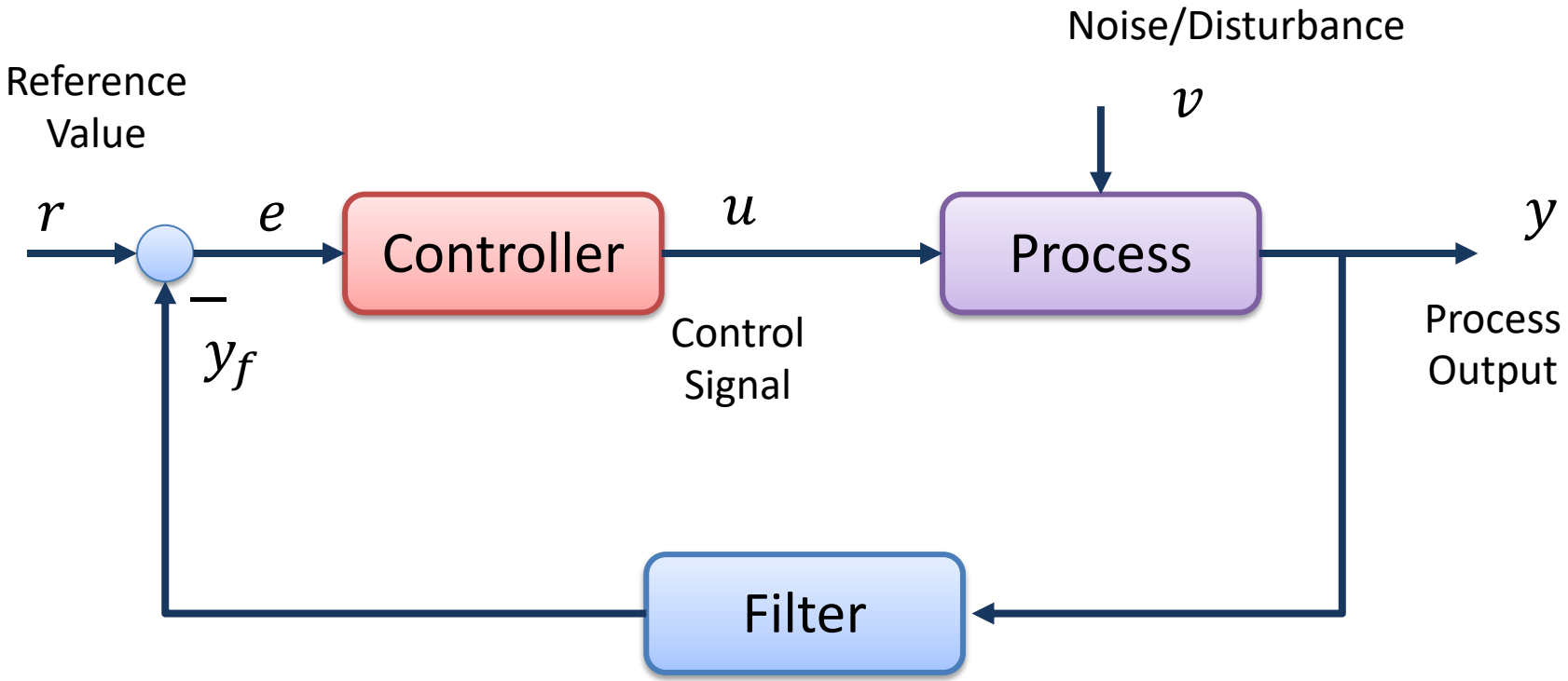


http://www.ni.com/en-no/support/model.usb-6008.html

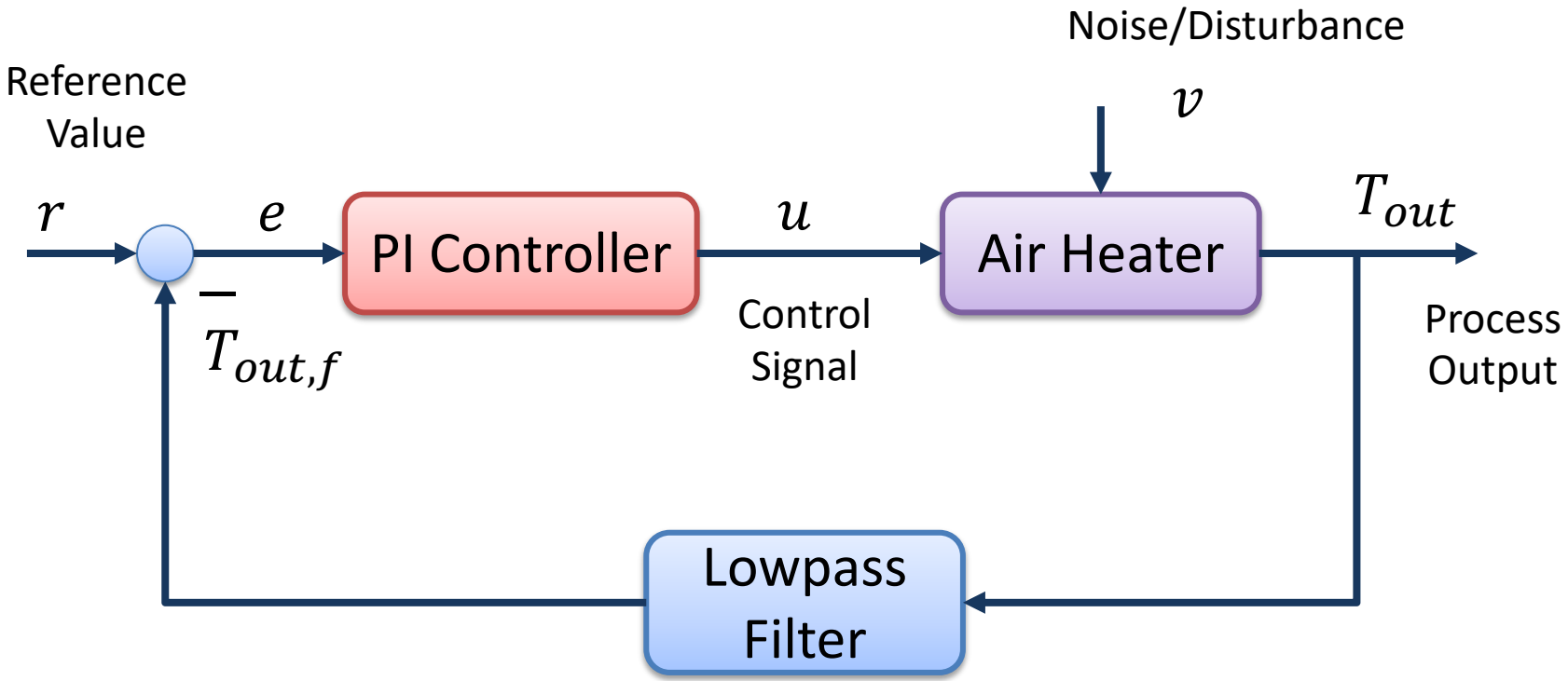
Control System

Hans-Petter Halvorsen

General Control System



Air Heater Control System



PID Controller

Hans-Petter Halvorsen

The PID Algorithm

$$u(t) = K_p e + \frac{K_p}{T_i}\int_0^t e d\tau + K_p T_d \dot{e}$$

Where u is the controller output and e is the control error:

$$e(t) = r(t) - y(t)$$

r is the Reference Signal or Set-point

y is the Process value, i.e., the Measured value

Tuning Parameters:

K_p Proportional Gain

T_i Integral Time [sec.]

T_d Derivative Time [sec.]

The PID Algorithm

$$u(t) = \underbrace{K_p e}_{\text{P}} + \underbrace{\frac{K_p}{T_i}\int_0^t e\,d\tau}_{\text{I}} + \underbrace{K_p T_d \dot{e}}_{\text{D}}$$

	P	I	D
	Proportional Gain	Integral Time	Derivative Time
Tuning Parameters:	K_p	T_i	T_d

Discrete PI Controller

We start with the continuous PI Controller:

$$u(t) = K_p e + \frac{K_p}{T_i}\int_0^t e d\tau$$

We derive both sides in order to remove the Integral:

$$\dot{u} = K_p \dot{e} + \frac{K_p}{T_i} e$$

We can use the Euler Backward Discretization method:

$$\dot{x} \approx \frac{x(k) - x(k-1)}{T_s}$$

Where T_s is the Sampling Time

Then we get:

$$\frac{u_k - u_{k-1}}{T_s} = K_p \frac{e_k - e_{k-1}}{T_s} + \frac{K_p}{T_i} e_k$$

Finally, we get:

$$u_k = u_{k-1} + K_p(e_k - e_{k-1}) + \frac{K_p}{T_i} T_s e_k$$

Where, $e_k = r_k - y_k$

Lowpass Filter

Hans-Petter Halvorsen

Lowpass Filter

The Transfer Function for a Low-pass filter is given by:

$$H(s) = \frac{y(s)}{u(s)} = \frac{1}{T_f s + 1}$$

Where T_f is the Filter Time Constant

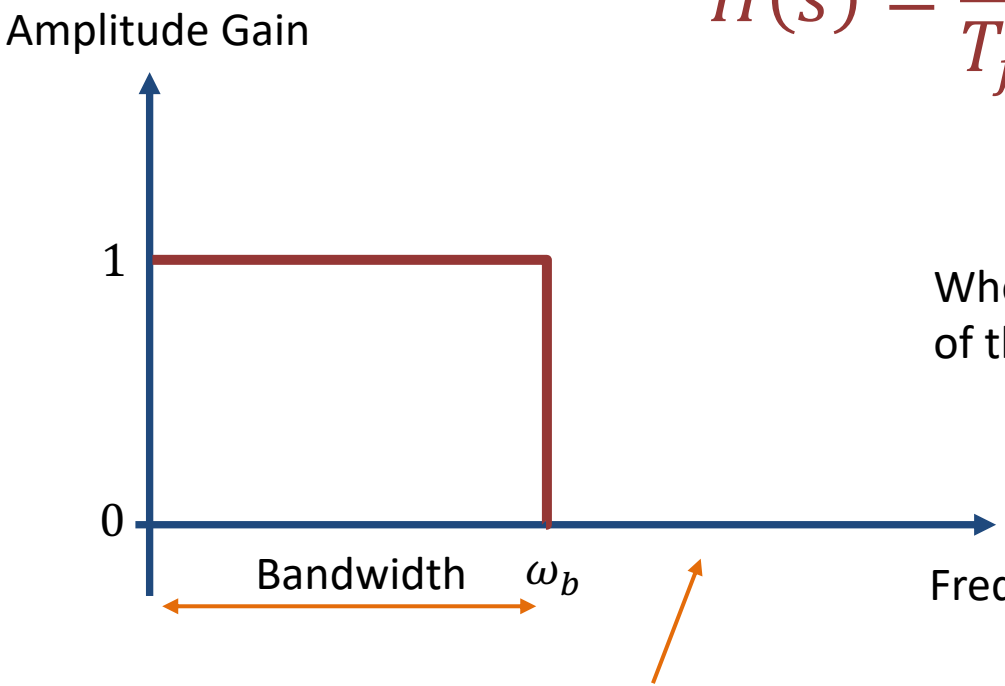
Why Lowpass Filter?

- In Measurement systems and Control Systems we typically need to deal with noise
- Noise is something we typically don't want
- Lowpass Filters are used to remove noise from the measured signals
- Noise is high-frequency signals
- A Lowpass Filter make sure the low frequencies pass and removes the high frequencies (the noise)

Lowpass Filter

Below we see an Ideal Lowpass Filter:

$$H(s) = \frac{1}{T_f s + 1} = \frac{1}{\frac{1}{\omega_b} s + 1}$$



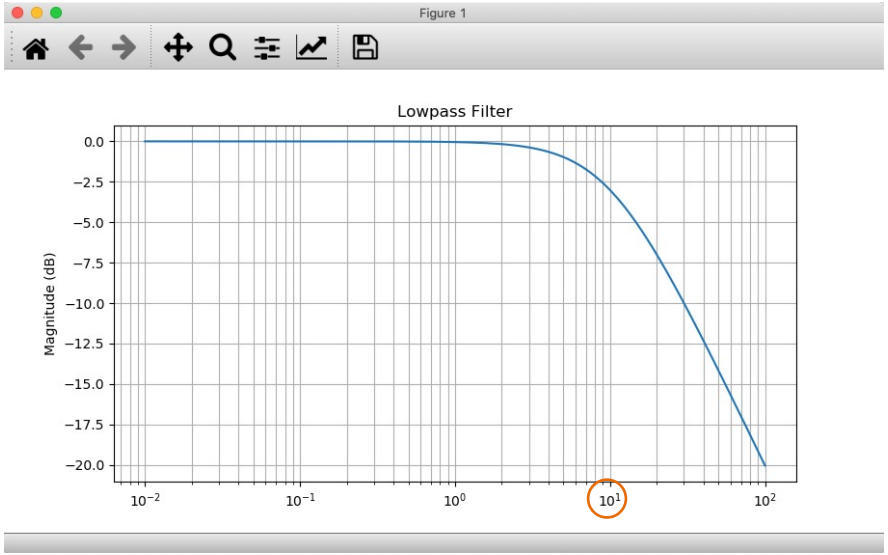
Where ω_b is the Bandwidth of the Lowpass Filter

High frequencies (above ω_b) are removed (or reduced)

Python

$$H(s) = \frac{1}{T_f s + 1} = \frac{1}{\frac{1}{\omega_b} s + 1}$$

We set $\omega_b = 10$ in this example




```
import numpy as np
import scipy.signal as signal
import matplotlib.pyplot as plt

# Define Transfer Function
wb = 10 #rad/s
Tf = 1/wb
num = np.array([1])
den = np.array([Tf , 1])

H = signal.TransferFunction(num, den)
print ('H(s) =', H)

# Frequencies
w_start = 0.01
w_stop = 100
step = 0.01
N = int ((w_stop-w_start )/step) + 1
w = np.linspace (w_start , w_stop , N)

# Bode Plot
w, mag, phase = signal.bode(H, w)

plt.figure()
plt.semilogx(w, mag)
plt.title("Lowpass Filter")
plt.grid(b=None, which='major', axis='both')
plt.grid(b=None, which='minor', axis='both')
plt.ylabel("Magnitude (dB)")
```

Discrete Lowpass Filter

The transfer function for the Low-pass filter is given by:

$$H(s) = \frac{y(s)}{u(s)} = \frac{1}{T_f s + 1}$$

Then we get:

$$(T_f s + 1) y(s) = u(s)$$

Inverse Laplace gives ($\dot{y} \Leftrightarrow sy$):

$$T_f s y + y = u \rightarrow T_f \dot{y} + y = u$$

We use se the **Euler Backward** method:

$$\dot{x} = \frac{x_k - x_{k-1}}{T_s}$$

This gives:

$$T_f \frac{y_k - y_{k-1}}{T_s} + y_k = u_k$$

Then we get:

$$T_f(y_k - y_{k-1}) + y_k T_s = u_k T_s$$
$$T_f y_k - T_f y_{k-1} + y_k T_s = u_k T_s$$
$$y_k(T_f + T_s) = T_f y_{k-1} + u_k T_s$$

This gives:

$$y_k = \frac{T_f}{T_f + T_s} y_{k-1} + \frac{T_s}{T_f + T_s} u_k$$

We set:

$$a \equiv \frac{T_s}{T_f + T_s}$$

This gives the following discrete Low-pass filter:

$$y_k = (1 - a) y_{k-1} + a u_k$$

Note! $T_s \ll T_f$

Golden Rule: $T_s \leq \frac{T_f}{5}$

Discrete Lowpass Filter

$$a = \frac{T_s}{T_f + T_s}$$

$$y_k = (1 - a)y_{k-1} + au_k$$

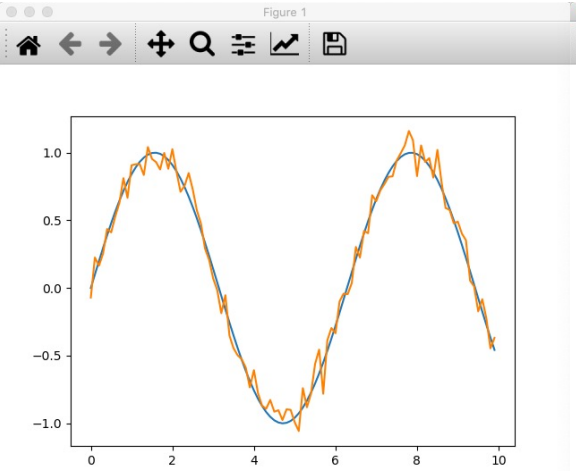
In SciPy.signal there are also lots of premade Filters that you can use that is better than this simple Lowpass Filter

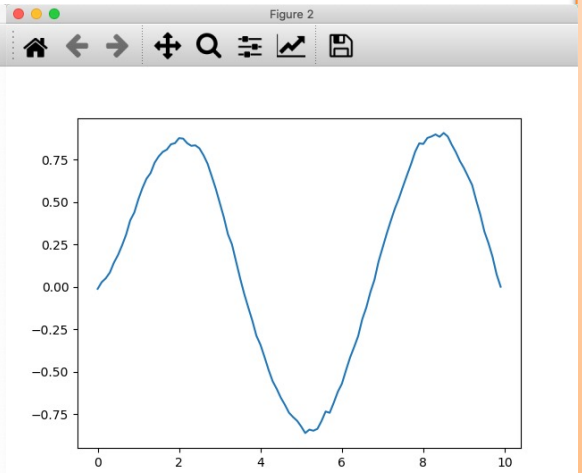
```
def lowpass(y, y_prev, Tf, Ts):
    a = Ts/(Tf+Ts)
    yf = (1-a)*y_prev + a*u
    return yf

..

Ts = 0.1
Tf = 0.5

..

y[k+1] = task.read()

..

y[k+1] = lowpass(y[k+1], y[k], Tf, Ts)

..
```

Test Filter with Simulated Noise

$$a = \frac{T_s}{T_f + T_s}$$

$$y_k = (1 - a)y_{k-1} + au_k$$



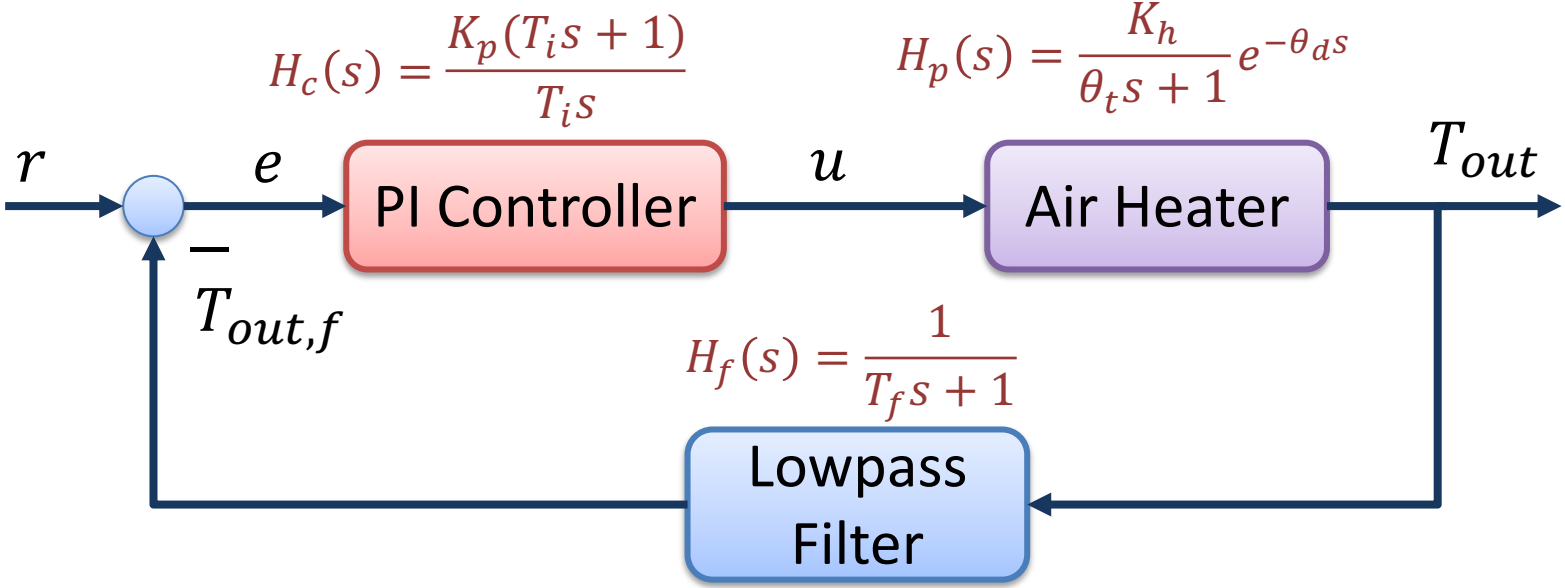



```
import numpy as np
import matplotlib.pyplot as plt

def lowpass(y, y_prev, Tf, Ts):
    a = Ts/(Tf+Ts)
    yf = (1-a)*y_prev + a*y
    return yf


Ts = 0.1
Tf = 0.5

start = 0; stop = 10; step =0.1
x = np.arange(start, stop, step)
N = len(x)

y_noise = np.zeros(N)
y_filter = np.zeros(N)

y = np.sin(x)

plt.plot(x, y)

for k in range(N):
    #print(k)
    mean = 0; std = 0.1; n = 1
    noise = np.random.normal(mean, std, n)

    y_noise[k] = y[k] + noise

    y_filter[k] = lowpass(y_noise[k],y_filter[k-1],Tf,Ts)

    print(y_filter[k],y_filter[k-1])


plt.plot(x, y_noise)
plt.show()

plt.figure(2)
plt.plot(x, y_filter)
plt.show()
```

Stability Analysis

Hans-Petter Halvorsen

Stability Analysis



We use the following Transfer Functions in the Stability Analysis of the Control System:

Loop Transfer Function:

$$L(s) = H_c(s)H_p(s)H_f(s)$$

Tracking Transfer Function:

$$T(s) = \frac{y(s)}{r(s)} = \frac{L(s)}{1 + L(s)}$$

```
import numpy as np
import matplotlib.pyplot as plt
import control

# Process Parameters
Kh = 3.5
theta_t = 22
theta_d = 2

# Transfer Function Process
num_p = np. array ([Kh])
den_p = np. array ([theta_t , 1])
Hp1 = control.tf(num_p , den_p)
#print ('Hp1(s) =', Hp1)

N = 5 # Time Delay - Order of the Approximation
[num_pade,den_pade] = control.pade(theta_d, N)
Hp_pade = control.tf(num_pade,den_pade);
#print ('Hp_pade(s) =', Hp_pade)

Hp = control.series(Hp1, Hp_pade);
#print ('Hp(s) =', Hp)

# Transfer Function PI Controller
Kp = 0.52
Ti = 18
num_c = np.array ([Kp*Ti, Kp])
den_c = np.array ([Ti , 0])

Hc = control.tf(num_c, den_c)
print ('Hc(s) =', Hc)

# Transfer Function Lowpass Filter
Tf = 0.5
num_f = np.array ([1])
den_f = np.array ([Tf , 1])

Hf = control.tf(num_f, den_f)
print('Hf(s) =', Hf)

# The Loop Transfer function
L = control.series(Hc, Hp, Hf)
print('L(s) =', L)

# Tracking Transfer function
T = control.feedback(L,1)
print('T(s) =', T)
```

```
# Step Response Feedback System (Tracking System)
t, y = control.step_response(T)
plt.figure(1)
plt.plot(t,y)
plt.title("Step Response Feedback System T(s)")
plt.grid()


# Bode Diagram with Stability Margins
plt.figure(2)
control.bode(L, dB=True, deg=True, margins=True)

# Poles and Zeros
control.pzmap(T)

p = control.pole(T)
z = control.zero(T)
print("poles = ", p)

# Calculating stability margins and crossover frequencies
gm , pm , w180 , wc = control.margin(L)

# Convert gm to Decibel
gmdb = 20 * np.log10(gm)

print("wc =", f'{wc:.2f}', "rad/s")
print("w180 =", f'{w180:.2f}', "rad/s")

print("GM =", f'{gm:.2f}')
print("GM =", f'{gmdb:.2f}', "dB")
print("PM =", f'{pm:.2f}', "deg")


# Find when System is Marginally Stable (Kritical Gain -  Kc)
Kc = Kp*gm
print("Kc=", f'{Kc:.2f}')
```

Results

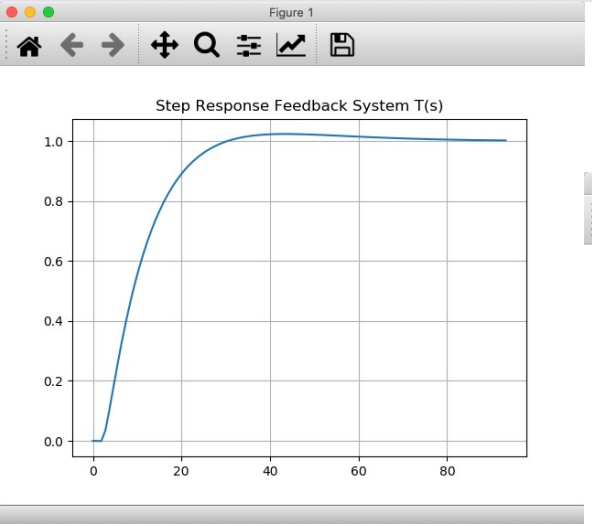


Figure 1
Step Response Feedback System T(s)
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80

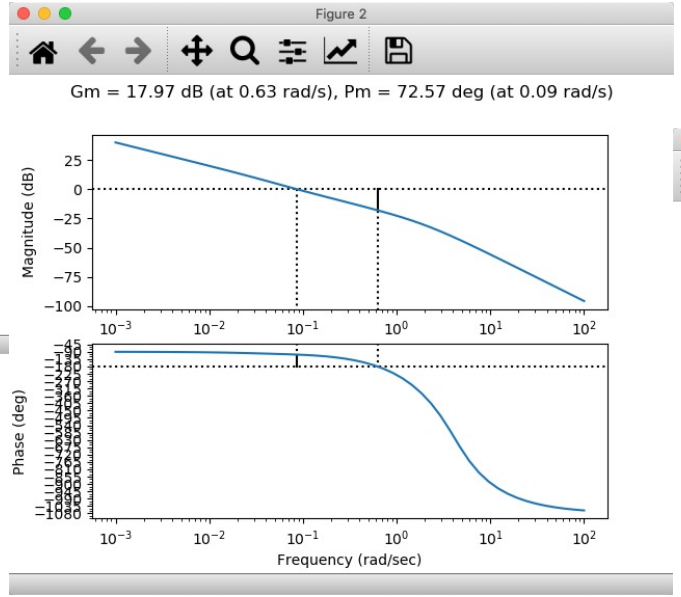


Figure 2
Gm = 17.97 dB (at 0.63 rad/s), Pm = 72.57 deg (at 0.09 rad/s)
Magnitude (dB)
25
0
−25
−50
−75
−100
Phase (deg)
Frequency (rad/sec)

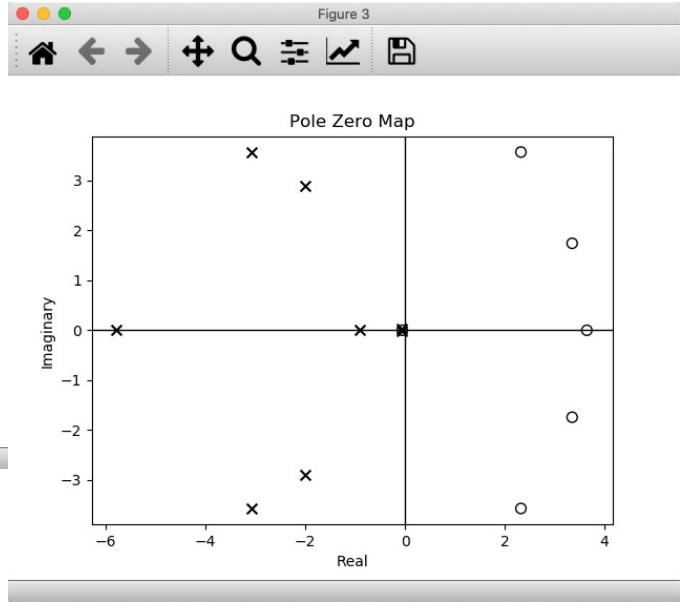


Figure 3
Pole Zero Map
Imaginary
3
2
1
0
−1
−2
−3
−6
−4
−2
0
2
4
Real

Control System Simulations

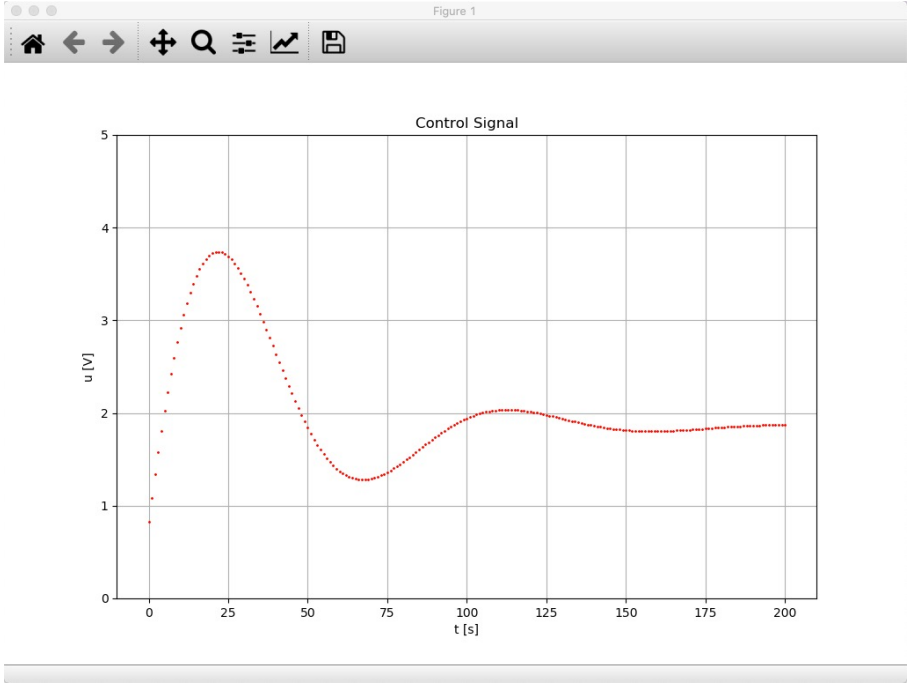
Hans-Petter Halvorsen

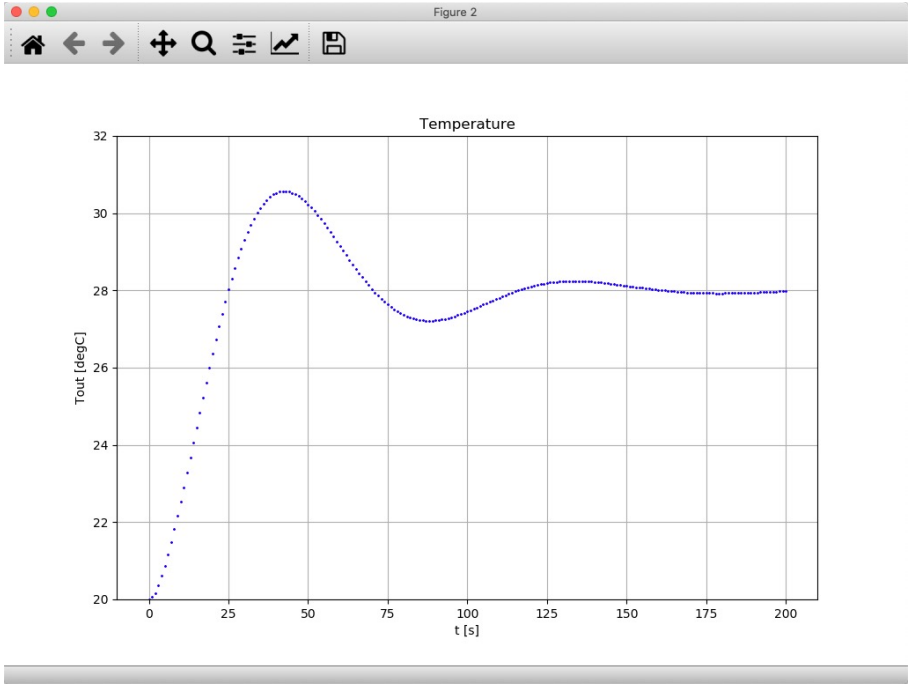
Control System Simulations

```
# Air Heater System
import numpy as np
import time
import matplotlib.pyplot as plt

# Model Parameters
Kh = 3.5
theta_t = 22
theta_d = 2
Tenv = 21.5
# Simulation Parameters
Ts = 0.1 # Sampling Time
Tstop = 200 # End of Simulation Time
N = int(Tstop/Ts) # Simulation length
Tout = np.zeros(N+2) # Initialization the Tout vector
Tout[0] = 20 # Initial Vaue

# PI Controller Settings
Kp = 0.1
Ti = 30

r = 28 # Reference value [degC]
e = np.zeros(N+2) # Initialization
u = np.zeros(N+2) # Initialization

t = np.arange(0,Tstop+2*Ts,Ts) #Create the Time Series

# Formatting the appearance of the Plot
plt.figure(1)
plt.title('Control Signal')
plt.xlabel('t [s]')
plt.ylabel('u [V]')
plt.grid()

plt.figure(2)
plt.title('Temperature')
plt.xlabel('t [s]')
plt.ylabel('Tout [degC]')
plt.grid()
```

```
# Simulation
for k in range(N+1):
    # Controller
    e[k] = r - Tout[k]

    u[k] = u[k-1] + Kp*(e[k] - e[k-1]) + (Kp/Ti)*e[k] #PI Controller

    if u[k]>5:
        u[k] = 5

    # Process Model
    Tout[k+1] = Tout[k] + (Ts/theta_t) * (-Tout[k] + Kh*u[int(k-theta_d/Ts)] + Tenv)


    print("t = %2.1f, u = %3.2f, Tout = %3.1f" %(t[k], u[k], Tout[k+1]))

    if k%10 == 0: #Update Plot only every second
        # Plot Control Signal
        plt.figure(1)
        plt.plot(t[k],u[k], '-o', markersize=1, color='red')
        plt.ylim(0, 5)
        plt.show()
        plt.pause(Ts)

        # Plot Temperature
        plt.figure(2)
        plt.plot(t[k],Tout[k+1], '-o', markersize=1, color='blue')
        plt.ylim(20, 32)
        plt.show()
        plt.pause(Ts)

    time.sleep(Ts)
```

Simulation Results





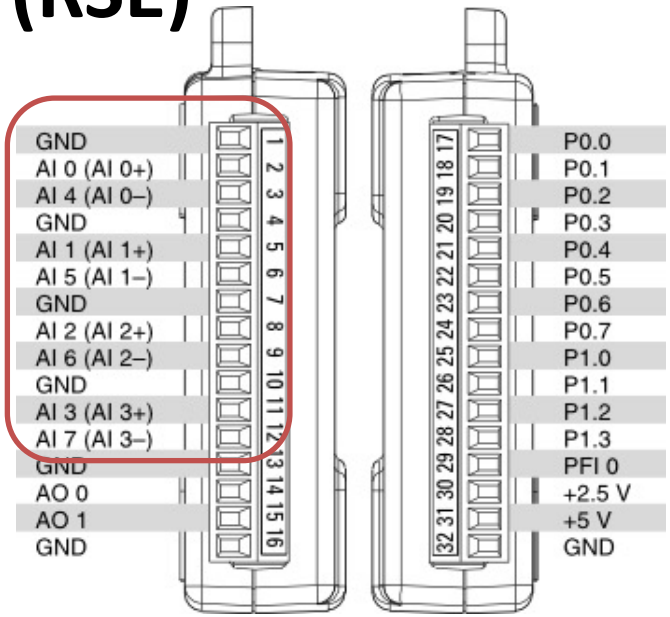
USB-6008

Hans-Petter Halvorsen

Analog In (Read)

USB-6008 has

- 8 AI **Referenced Single Ended (RSE)** Analog Inputs Channels
- or 4 AI **Differential** Analog Inputs Channels Default



The Voltage Range is $-10V - 20V$

$0V - 5V$ is default

Analog In (Read)

```
import nidaqmx

task = nidaqmx.Task()
task.ai_channels.add_ai_voltage_chan("Dev1/ai0")
task.start()

value = task.read()
print(value)

task.stop
task.close()
```

Differential is Default

Analog In with RSE

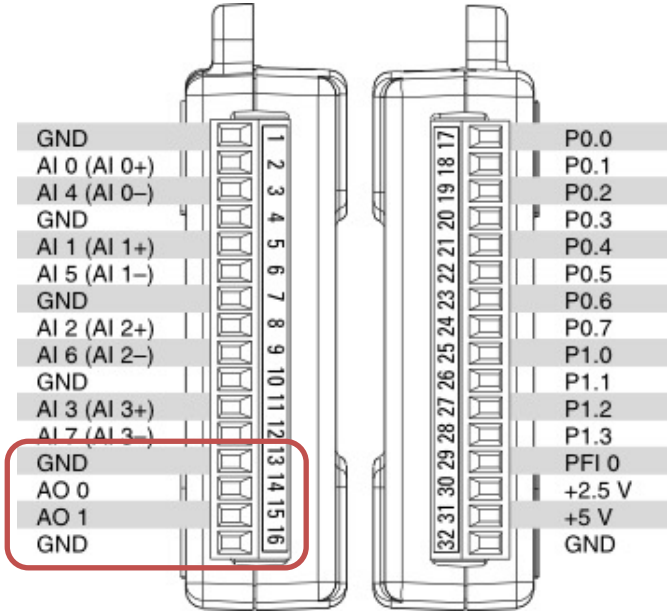
```
import nidaqmx

from nidaqmx.constants import (
    TerminalConfiguration)

task = nidaqmx.Task()

task.ai_channels.add_ai_voltage_chan("Dev1/ai0",
                                     terminal_config=TerminalConfiguration.RSE)

task.start()

value = task.read()
print(value)
task.stop()
task.close()
```

Analog In with Differential

```
import nidaqmx

from nidaqmx.constants import (
    TerminalConfiguration)

task = nidaqmx.Task()

task.ai_channels.add_ai_voltage_chan("Dev1/ai0",
                                     terminal_config=TerminalConfiguration.DIFFERENTIAL)
task.start()

value = task.read()
print(value)

task.stop()
task.close()
```

Analog Out (Write)

- Note! The USB-6008 can only output a voltage signal between 0 and 5V
- The USB-6008 has 2 Analog Out Channels:
  - **AO0**
  - **AO1**



Analog Out (Write)

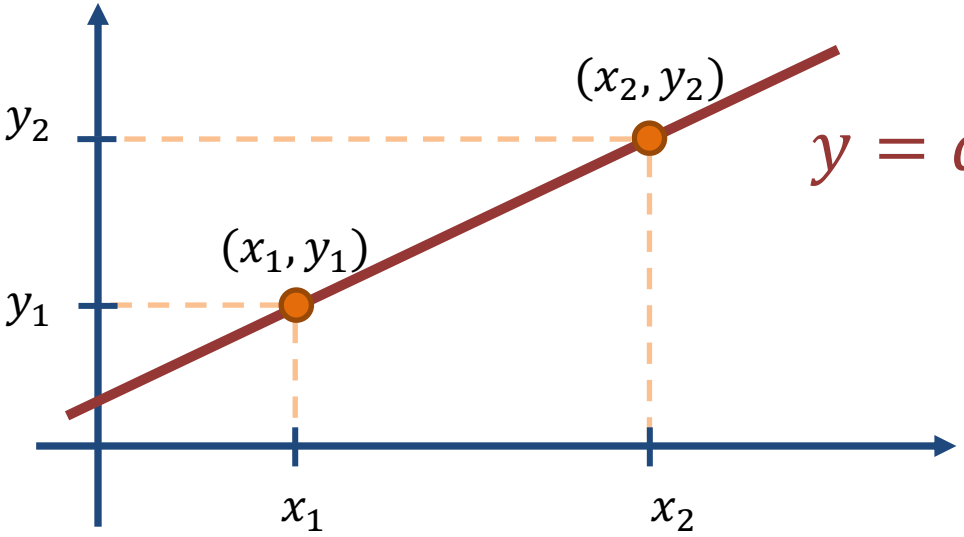
```
import nidaqmx

task = nidaqmx.Task()
task.ao_channels.add_ao_voltage_chan('Dev1/ao0','mychannel',0,5)
task.start()

value = 2
task.write(value)

task.stop()
task.close()
```

You can, e.g., use a **Multimeter** in order to check if the the program outputs the correct value

Scaling

Hans-Petter Halvorsen

Linear Scaling



We find the Linear scaling ($y = ax + b$) using the following formula:

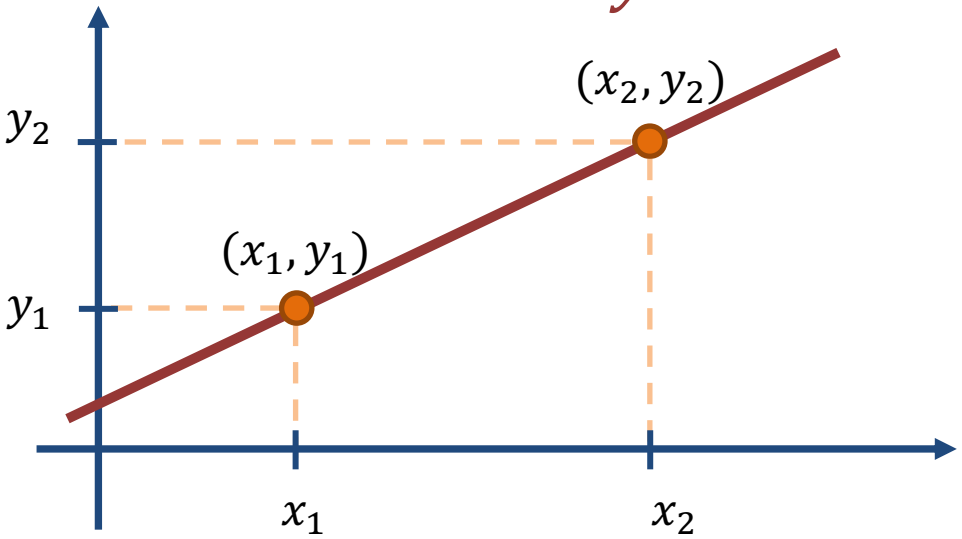
$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

Scaling

From the Air Heater we get a voltage signal $(1 - 5V)$ which we need to scale to a Temperature value in degrees Celsius $(0 - 50°C)$. It is a Linear relation between the voltage signal and the value in degrees Celsius.

$1 - 5V \rightarrow 0 - 50°C$

$$y = ax + b$$



We have the following:

$(x_1, y_1) = (1, 0)$
$(x_2, y_2) = (5, 50)$

We use:

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

Then we get:

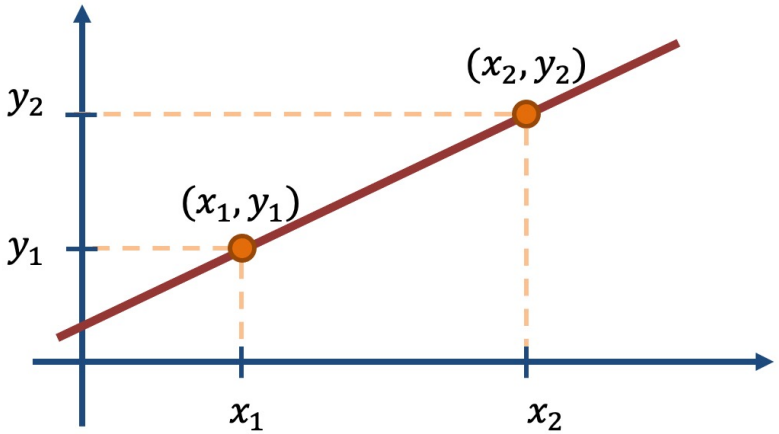
$$y - 0 = \frac{50 - 0}{5 - 1}(x - 1)$$

Finally:

$$y = 12.5x - 12.5$$

Scaling as a Python Function

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$



$$y = ax + b$$

```
def scaling(x, x1, x2, y1, y2):
    y = y1 + (x-x1)*(y2-y1)/(x2-x1)
    return y

x = 1

y = scaling(x, 1, 5, 0, 50)

print(y)
```

Test of the scaling() Function:

$x = 1$ should give $y = 0$

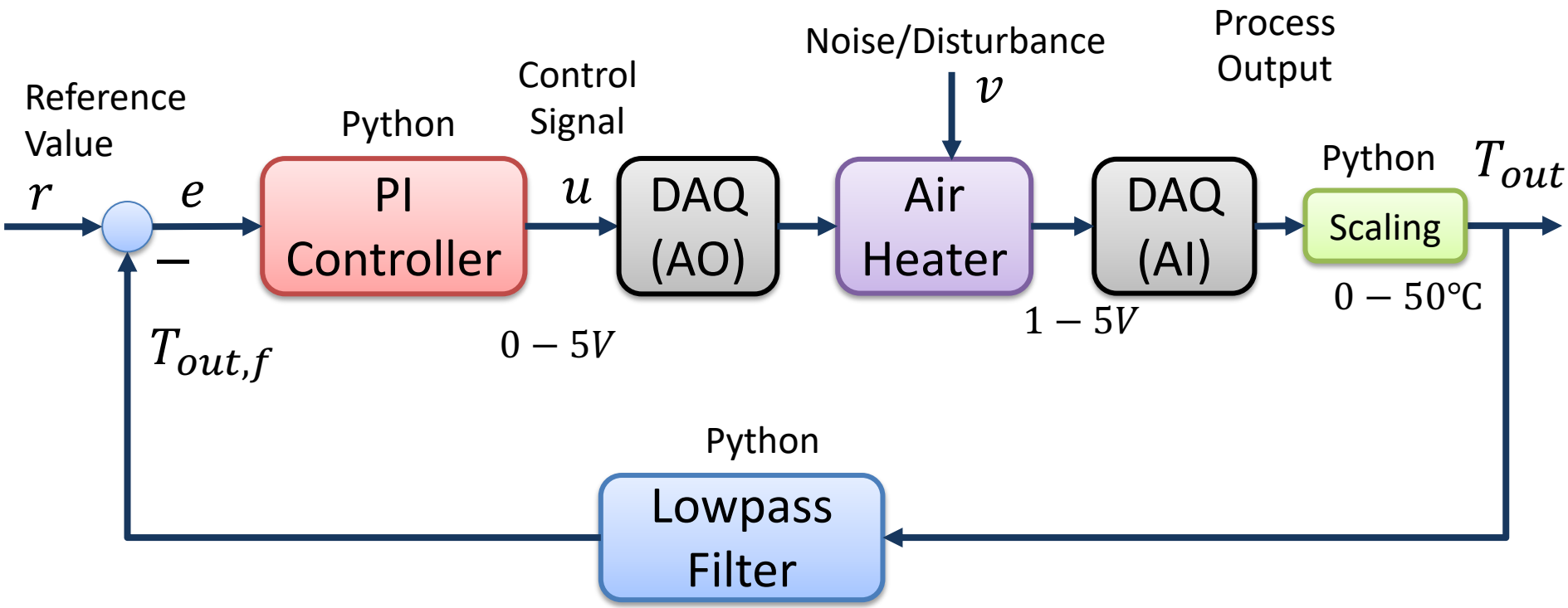
$x = 3$ should give $y = 25$

$x = 5$ should give $y = 50$

Control System Implementation

Hans-Petter Halvorsen

Air Heater Control System



```
# Air Heater System
import numpy as np
import time
import matplotlib.pyplot as plt
import nidaqmx

from nidaqmx.constants import (
    TerminalConfiguration)

task_ai = nidaqmx.Task()
task_ai.ai_channels.add_ai_voltage_chan("DAQAir/ai0",
                                        terminal_config=TerminalConfiguration.RSE)
task_ai.start()

task_ao = nidaqmx.Task()
task_ao.ao_channels.add_ao_voltage_chan('DAQAir/ao0','mychannel',0,5)
task_ao.start()

# Control System Parameters
Ts = 0.1 # Sampling Time
Tstop = 100
N = int(Tstop/Ts)
Tout = np.zeros(N+2) # Initialization the Tout vector
Tout[0] = 20 # Initial Value
Tf = 0.5 #Lowpass Filter

# PI Controller Settings
Kp = 0.1
Ti = 30
r = 28 # Reference value [degC]
e = np.zeros(N+2) # Initialization
u = np.zeros(N+2) # Initialization
t = np.arange(0,Tstop+2*Ts,Ts) #Create the Time Series

# Formatting the appearance of the Plot
plt.figure(1)
plt.title('Control Signal')
plt.xlabel('t [s]')
plt.ylabel('u [V]')
plt.grid()

plt.figure(2)
plt.title('Temperature')
plt.xlabel('t [s]')
plt.ylabel('Tout [degC]')
plt.grid()

def scaling(x, x1, x2, y1, y2):
    y = y1 + (x-x1)*(y2-y1)/(x2-x1)
    return y
def lowpass(u, y_prev, Tf, Ts):
    a = Ts/(Tf+Ts)
    y = (1-a)*y_prev + a*u
    y_prev = y
    return y
```

```
# Control System Loop
for k in range(N+1):
    # Controller
    e[k] = r - Tout[k]
    u[k] = u[k-1] + Kp*(e[k] - e[k-1]) + (Kp/Ti)*e[k] #PI Controller

    if u[k]<0:
        u[k] = 0
    if u[k]>5:
        u[k] = 5

    task_ao.write(u[k])

    # Process Model
    ToutVolt = task_ai.read()
    if ToutVolt<1:
        ToutVolt = 1
    if ToutVolt>5:
        ToutVolt = 5

    Tout[k+1] = scaling(ToutVolt, 1, 5, 0, 50)

    Tout[k+1] = lowpass(Tout[k+1], Tout[k], Tf, Ts)

    print("t = %2.1f, u = %3.2f, Tout = %3.1f" %(t[k], u[k], Tout[k+1]))

    if k%10 == 0: #Update Plot every second
        # Plot Control Signal
        plt.figure(1)
        plt.plot(t[k],u[k], '-o', markersize=5, color='red')
        plt.ylim(0, 5)
        plt.show()
        plt.pause(Ts)

        # Plot Temperature
        plt.figure(2)
        plt.plot(t[k],Tout[k+1], '-o', markersize=5, color='blue')
        plt.ylim(20, 32)
        plt.show()
        plt.pause(Ts)

    time.sleep(Ts)

plt.figure(3)
plt.plot(t, Tout)
plt.title('Temperature')
plt.xlabel('t [s]')
plt.ylabel('Tout [degC]')
plt.grid()

task_ai.stop()
task_ai.close()
task_ao.write(0)
task_ao.stop()
task_ao.close()
```

Results

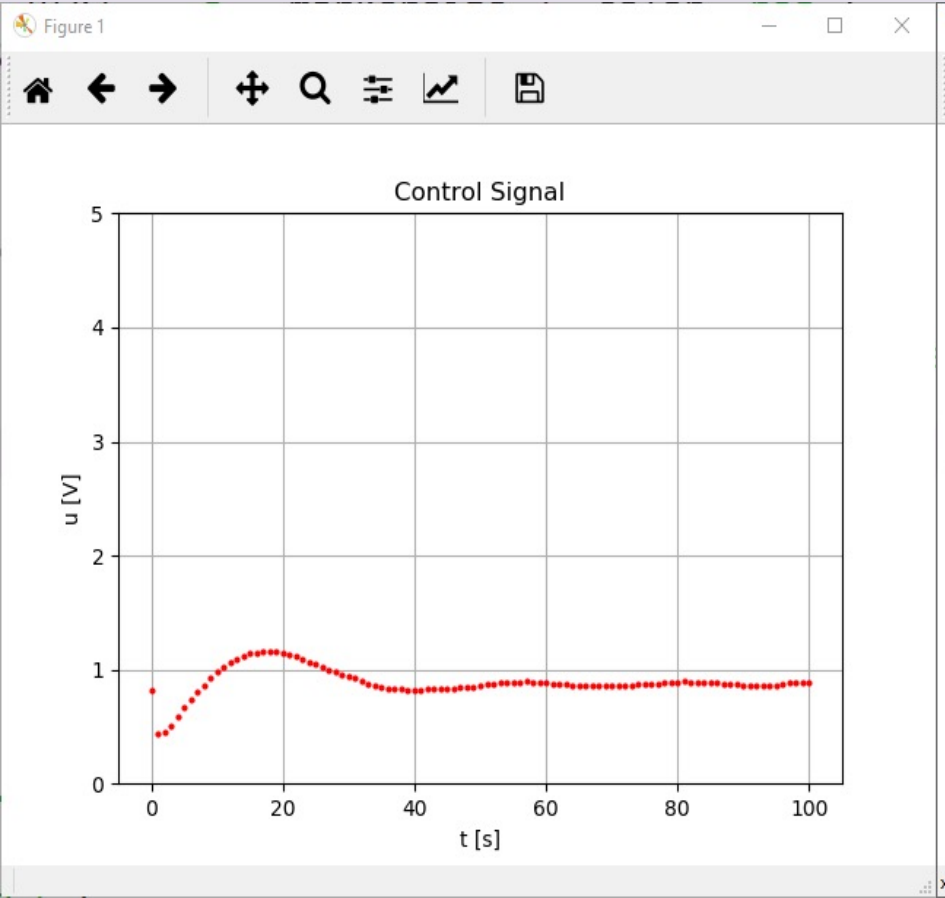


Figure 1
Control Signal
u [V]
t [s]
0
1
2
3
4
5
0
20
40
60
80
100

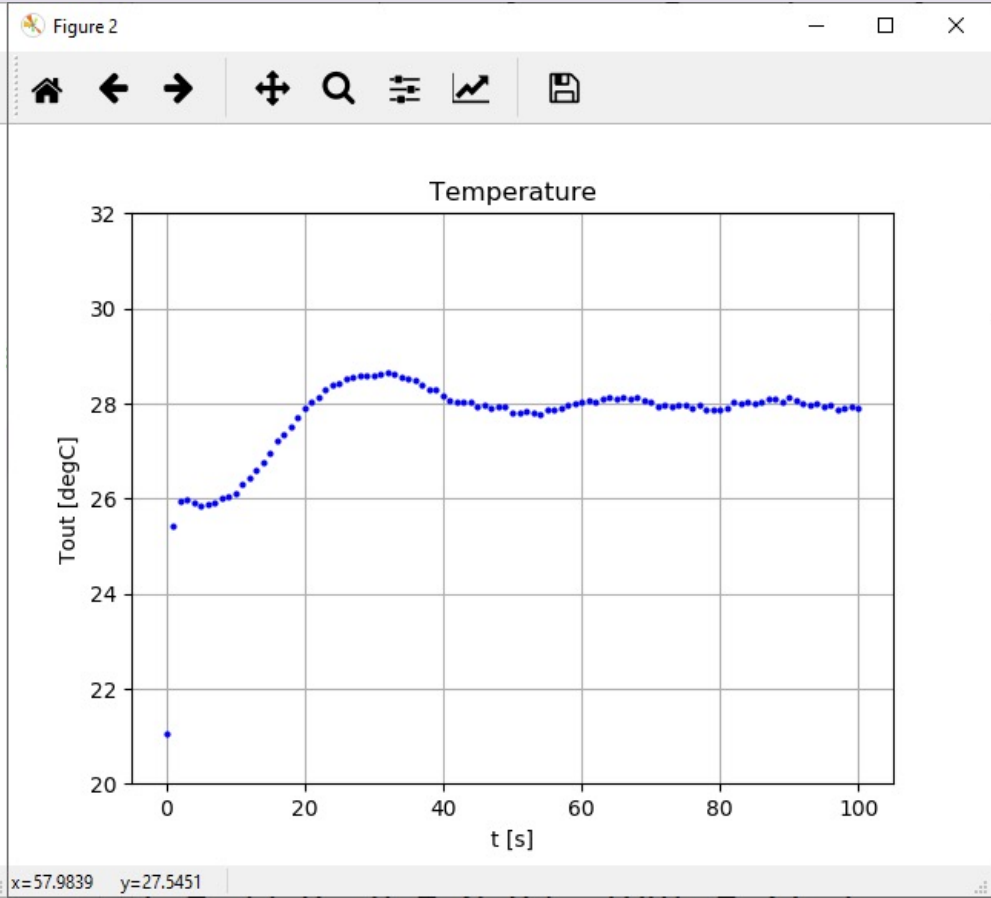
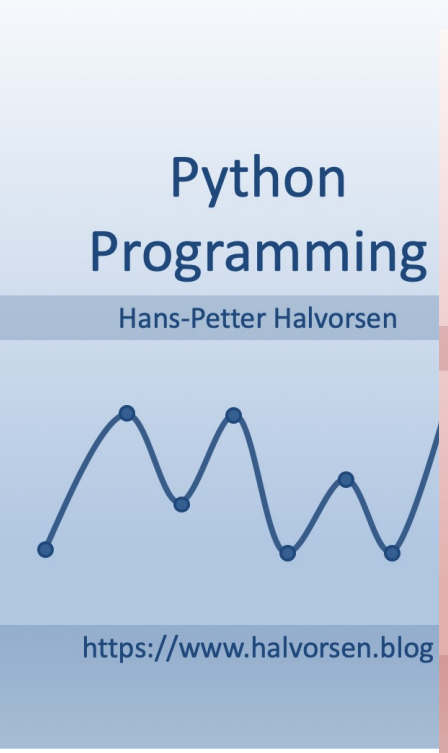
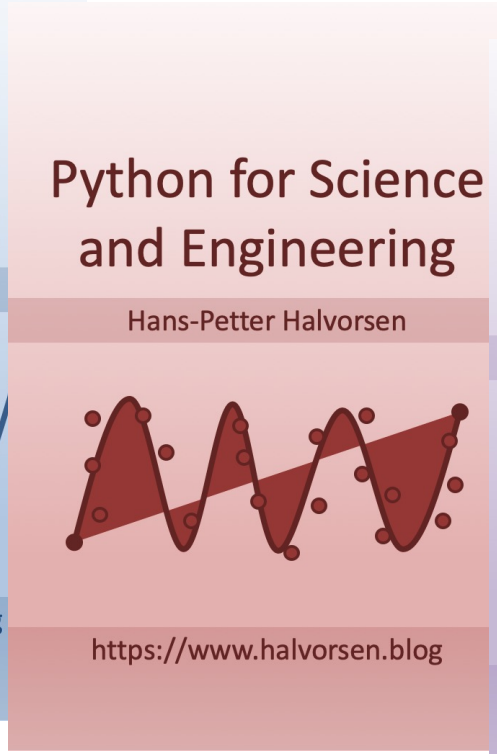
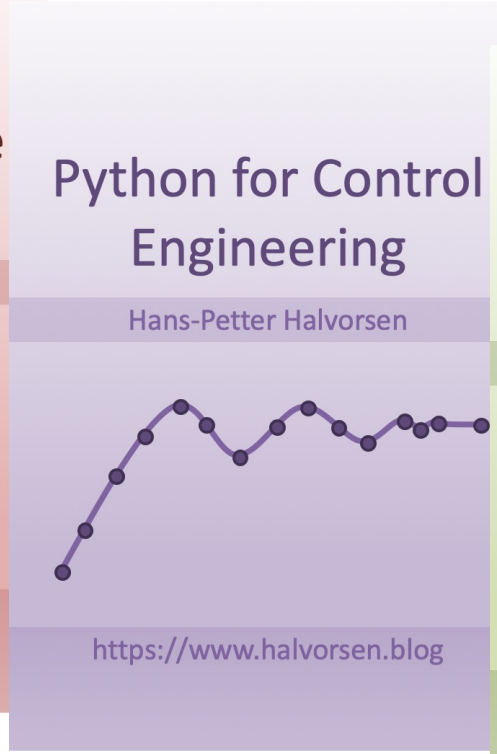


Figure 2
Temperature
Tout [degC]
t [s]
20
22
24
26
28
30
32
0
20
40
60
80
100
x=57.9839 y=27.5451

Additional Python Resources







https://www.halvorsen.blog/documents/programming/python/

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

E-mail: hans.p.halvorsen@usn.no

Web: https://www.halvorsen.blog