https://www.halvorsen.blog

Control Systems with Python

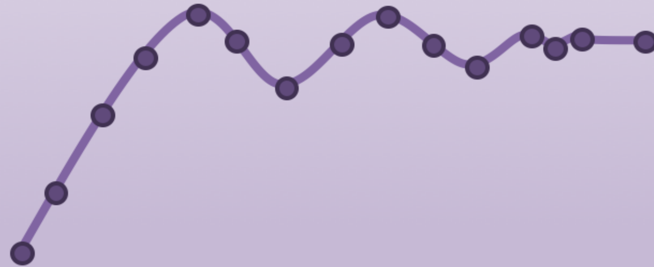
Exemplified by using a Small-scale Industrial Process called “Air Heater System”

Hans-Petter Halvorsen

Free Textbook with lots of Practical Examples

Python for Control Engineering

Hans-Petter Halvorsen



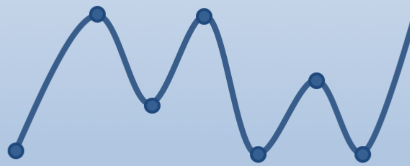
https://www.halvorsen.blog

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

Additional Python Resources

Python Programming

Hans-Petter Halvorsen



https://www.halvorsen.blog

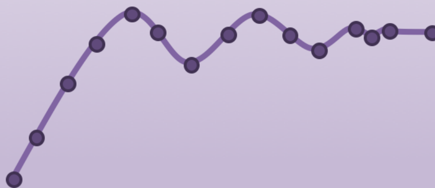
Python for Science and Engineering

Hans-Petter Halvorsen

https://www.halvorsen.blog

Python for Control Engineering

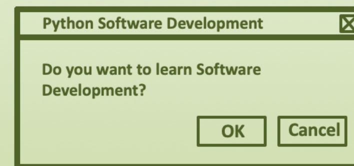
Hans-Petter Halvorsen



https://www.halvorsen.blog

Python for Software Development

Hans-Petter Halvorsen



https://www.halvorsen.blog

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

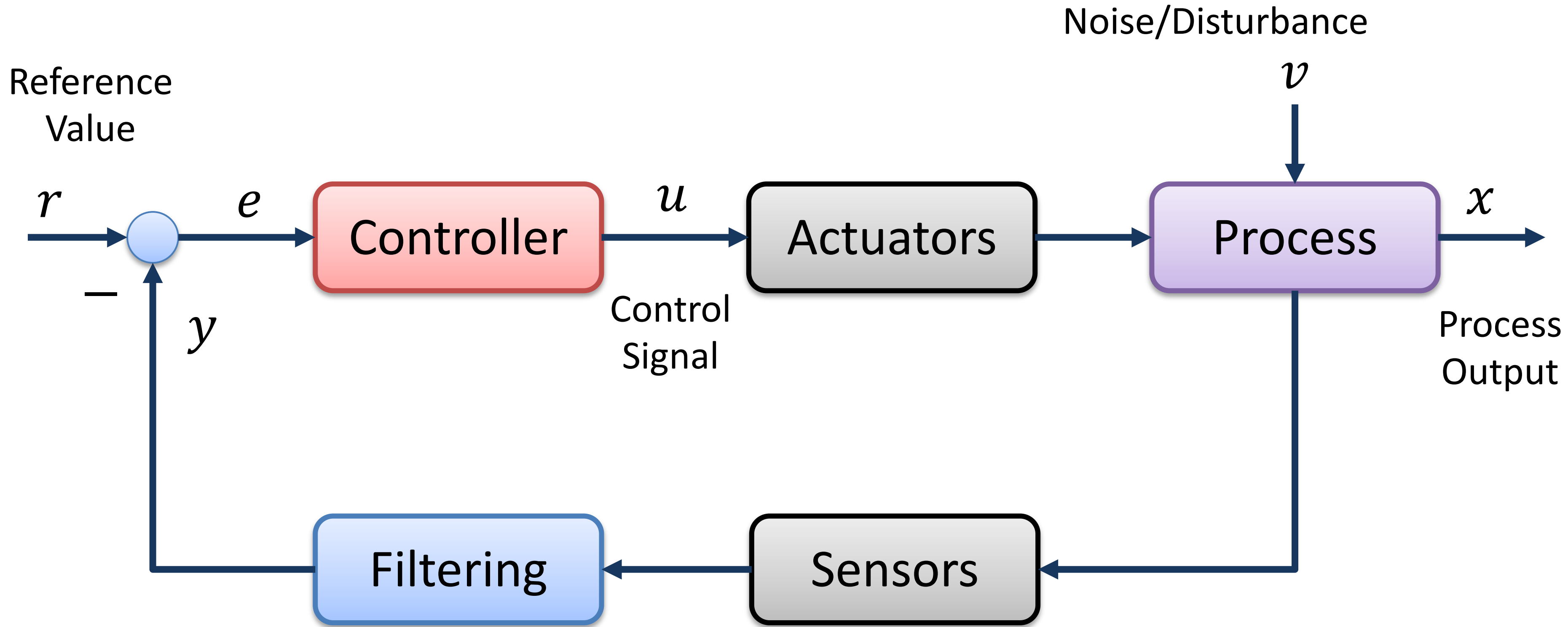
Contents

- Introduction to Control Engineering/Process Control
- Small-scale Industrial Process
- Process Simulator
- Control System

Control Engineering

Hans-Petter Halvorsen

Control System



https://www.halvorsen.blog

Python Libraries for Control Engineering

Hans-Petter Halvorsen

Python Libraries for Control Engineering

- SciPy.signal
  - The SciPy Python Library is included with the Python Anaconda Distribution
- Python Control Systems Library
  - This Python Library requires a separate installation (but this is very easy)

SciPy.signal

- The SciPy.signal contains Signal Processing functions
- SciPy is also included with the Anaconda distribution
- If you have installed Python using the Anaconda distribution, you don't need to install anything
- https://docs.scipy.org/doc/scipy/reference/signal.html

Continuous-time linear systems

lti(*system)	Continuous-time linear time invariant system base class.
StateSpace(*system, **kwargs)	Linear Time Invariant system in state-space form.
TransferFunction(*system, **kwargs)	Linear Time Invariant system class in transfer function form.
ZerosPolesGain(*system, **kwargs)	Linear Time Invariant system class in zeros, poles, gain form.
lsim(system, U, T[, X0, interp])	Simulate output of a continuous-time linear system.
lsim2(system[, U, T, X0])	Simulate output of a continuous-time linear system, by using the ODE solver scipy.integrate.odeint.
impulse(system[, X0, T, N])	Impulse response of continuous-time system.
impulse2(system[, X0, T, N])	Impulse response of a single-input, continuous-time linear system.
step(system[, X0, T, N])	Step response of continuous-time system.
step2(system[, X0, T, N])	Step response of continuous-time system.
freqresp(system[, w, n])	Calculate the frequency response of a continuous-time system.
bode(system[, w, n])	Calculate Bode magnitude and phase data of a continuous-time system.

Python Control Systems Library

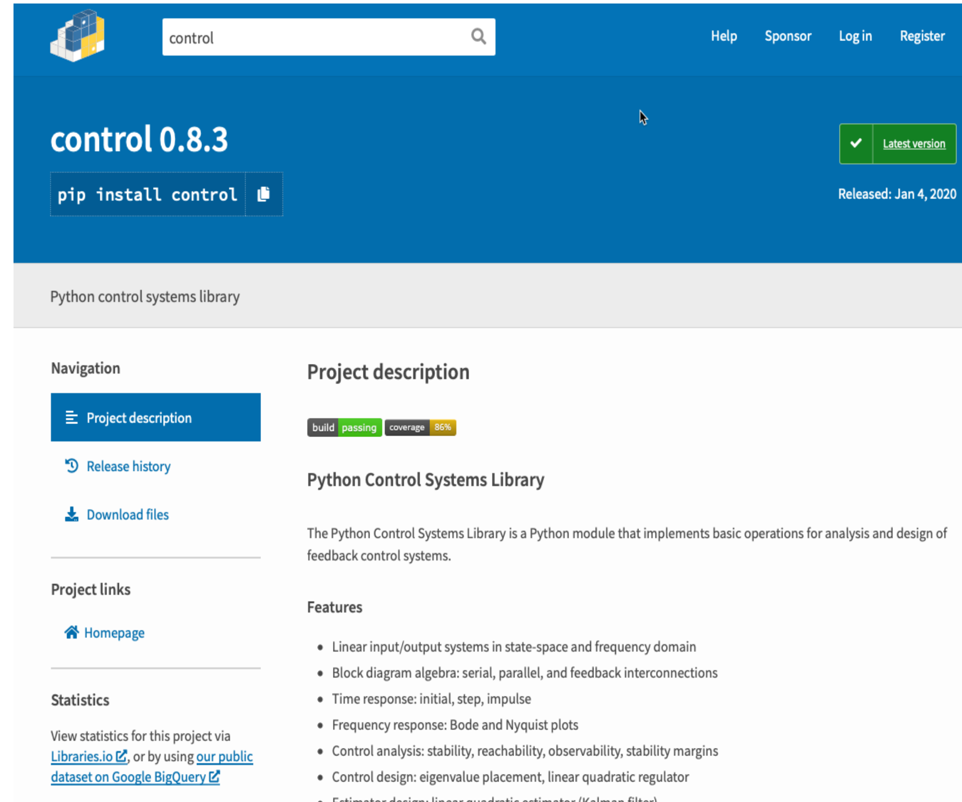
- The Python Control Systems Library (control) is a Python package that implements basic operations for analysis and design of feedback control systems.
- Existing MATLAB user? The functions and the features are very similar to the MATLAB Control Systems Toolbox.
- Python Control Systems Library Homepage: https://pypi.org/project/control
- Python Control Systems Library Documentation: https://python-control.readthedocs.io

Installation

The Python Control Systems Library package may be installed using pip:

```
pip install control
```

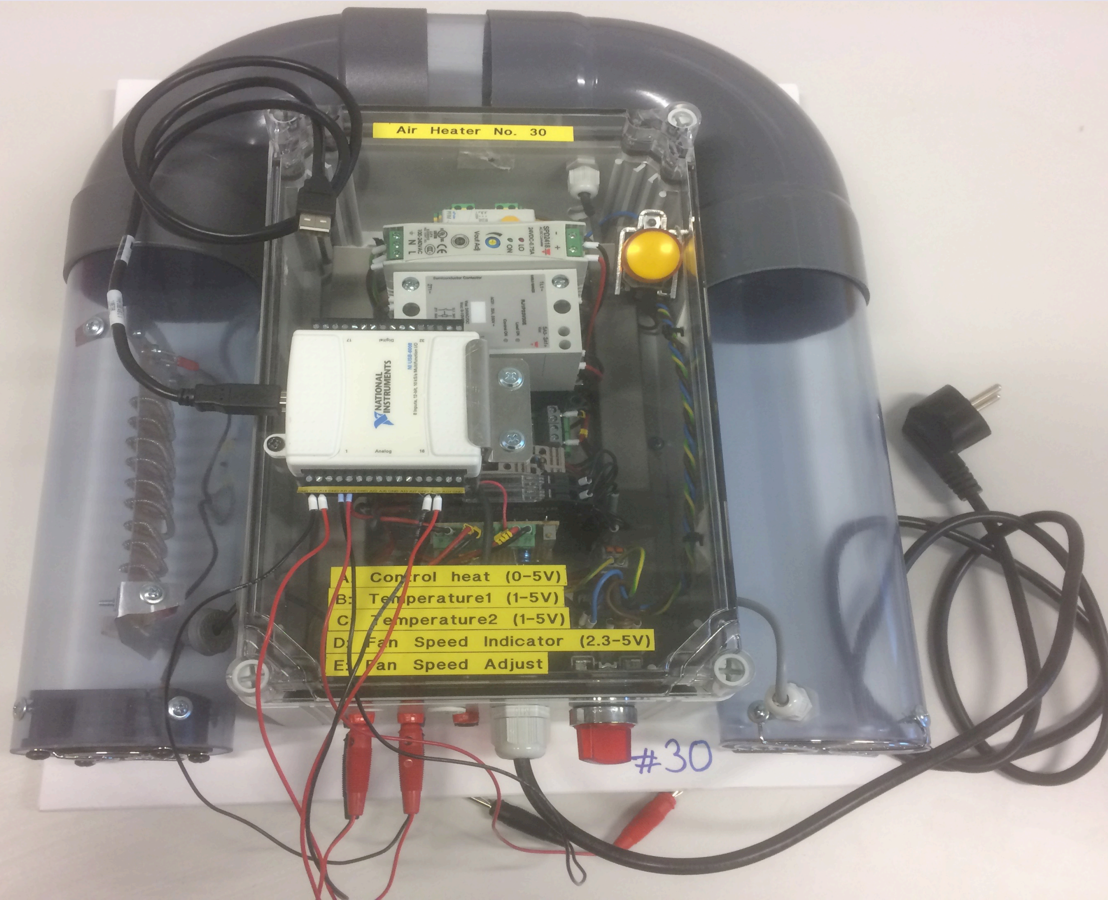
- PIP is a **Package Manager** for Python packages/modules.
- You find more information here: https://pypi.org
- Search for "control".
- **The Python Package Index (PyPI)** is a repository of Python packages where you use PIP in order to install them



Small-scale Industrial Process

Hans-Petter Halvorsen

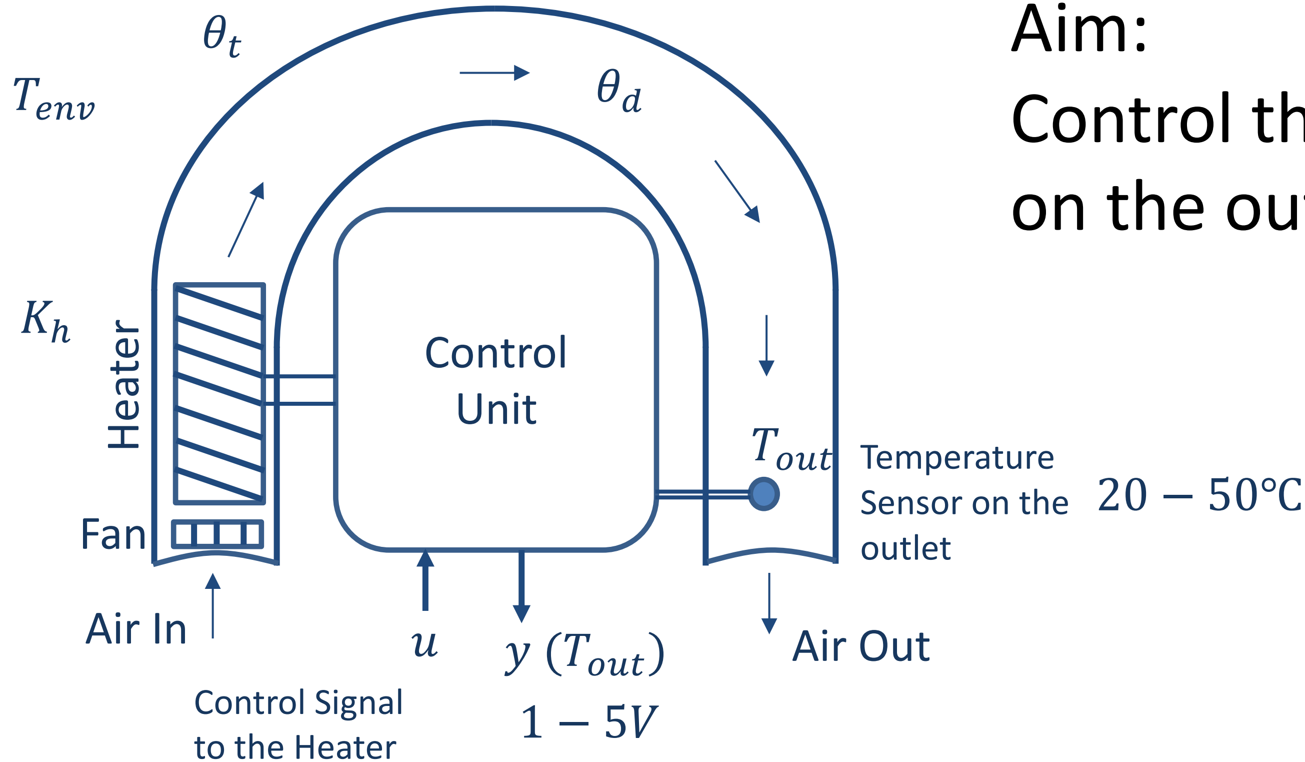
Small-scale Industrial Process



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



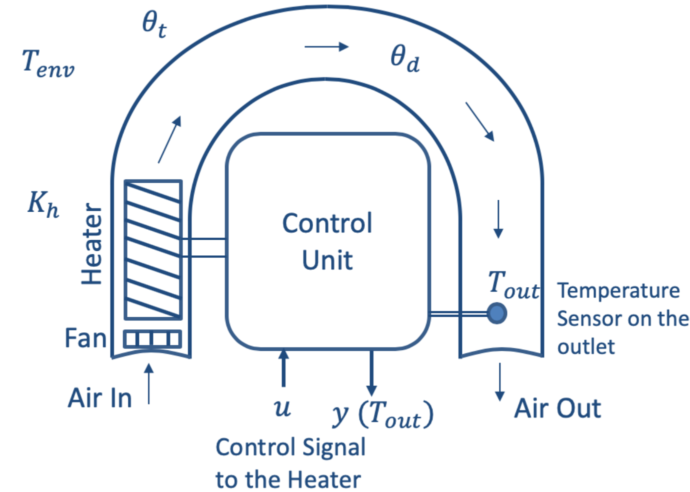
Aim:

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

Air Heater Model

The system can be modelled as a 1. order system with time-delay

$$\dot{T}_{out} = \frac{1}{\theta_t}\{-T_{out} + [K_h u(t - \theta_d) + T_{env}]\}$$



- T_{out} is the air temperature at the tube outlet
- u $[V]$ is the control signal to the heater
- θ_t $[s]$ is the time-constant
- K_h $[deg\ C\ /\ V]$ is the heater gain
- θ_d $[s]$ is the time-delay representing air transportation and sluggishness in the heater
- T_{env} is the environmental (room) temperature. It is the temperature in the outlet air of the air tube when the control signal to the heater has been set to zero for relatively long time (some minutes)

Model Values

You can assume the following values in your simulations:

$$\theta_t = 22\ s$$

$$\theta_d = 2\ s$$

$$K_h = 3.5\ \frac{°\mathrm{C}}{V}$$

$$T_{env} = 21.5\ °\mathrm{C}$$

The range for T_{out} could, e.g., be $20°\mathrm{C} \leq T_{out} \leq 50°\mathrm{C}$

Process Simulator

Hans-Petter Halvorsen

Discrete Air Heater

Continuous Model:

$$\dot{T}_{out} = \frac{1}{\theta_t}\{-T_{out} + [K_h u(t - \theta_d) + T_{env}]\}$$

We can use e.g., the Euler Approximation in order to find the discrete Model:

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

T_s - Sampling Time

$x(k)$ - Present value

$x(k+1)$ - Next (future) value

The discrete Model will then be on the form:

$$x(k+1) = x(k) + \ \ldots$$

We can then implement the discrete model in any programming language

Discrete Air Heater

We make a discrete version:

$$\dot{T}_{out} = \frac{1}{\theta_t}\{-T_{out} + [K_h u(t - \theta_d) + T_{env}]\}$$

$$\frac{T_{out}(k+1) - T_{out}(k)}{T_s} = \frac{1}{\theta_t}\{-T_{out}(k) + [K_h u(k - \theta_d) + T_{env}]\}$$

This gives the following discrete system:

$$T_{out}(k+1) = T_{out}(k) + \frac{T_s}{\theta_t}\{-T_{out}(k) + [K_h u(k - \theta_d) + T_{env}]\}$$

The Time delay θ_d makes it a little complicated. We can simplify by setting $\theta_d = 0$

$$T_{out}(k+1) = T_{out}(k) + \frac{T_s}{\theta_t}\{-T_{out}(k) + [K_h u(k) + T_{env}]\}$$

Discrete Air Heater (Simplified)

Discrete version with Time delay $\theta_d = 0$

$$T_{out}(k+1) = T_{out}(k) + \frac{T_s}{\theta_t}\{-T_{out}(k) + [K_h u(k) + T_{env}]\}$$

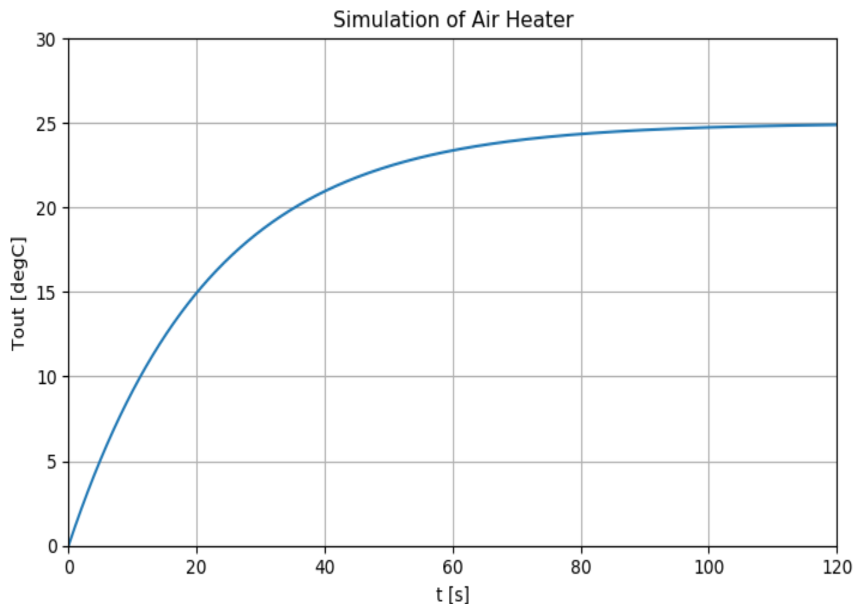
We can use the following values in the simulation:

$\theta_t = 22$
$K_h = 3.5$
$T_{env} = 21.5$

We can set the Sampling Time $T_s = 0.1s$

Python

Simulation of Discrete Air Heater with Time delay $\theta_d = 0$




```python
import numpy as np
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 = 120 # End of Simulation Time
uk = 1 # Step Response
N = int(Tstop/Ts) # Simulation length
Tout = np.zeros(N+2) # Initialization the Tout vector
Tout[0] = 0 # Initial Vaue

# Simulation
for k in range(N+1):
    Tout[k+1] = Tout[k] + (Ts/theta_t) * (-Tout[k] + Kh*uk + Tenv)

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

plt.plot(t,Tout)

# Formatting the appearance of the Plot
plt.title('Simulation of Air Heater')
plt.xlabel('t [s]')
plt.ylabel('Tout [degC]')
plt.grid()
xmin = 0
xmax = Tstop
ymin = 0
ymax = 30
plt.axis([xmin, xmax, ymin, ymax])
plt.show()
```

Discrete Air Heater with Time Delay

We have the following discrete system:

$$T_{out}(k+1) = T_{out}(k) + \frac{T_s}{\theta_t}\{-T_{out}(k) + [K_h u(k-\theta_d) + T_{env}]\}$$

The Time delay θ_d makes it more complicated to implement

θ_d is in seconds and we need to convert it to discrete intervals in forms of k

The discrete version of θ_d is: $\frac{\theta_d}{T_s}$

Then we get:

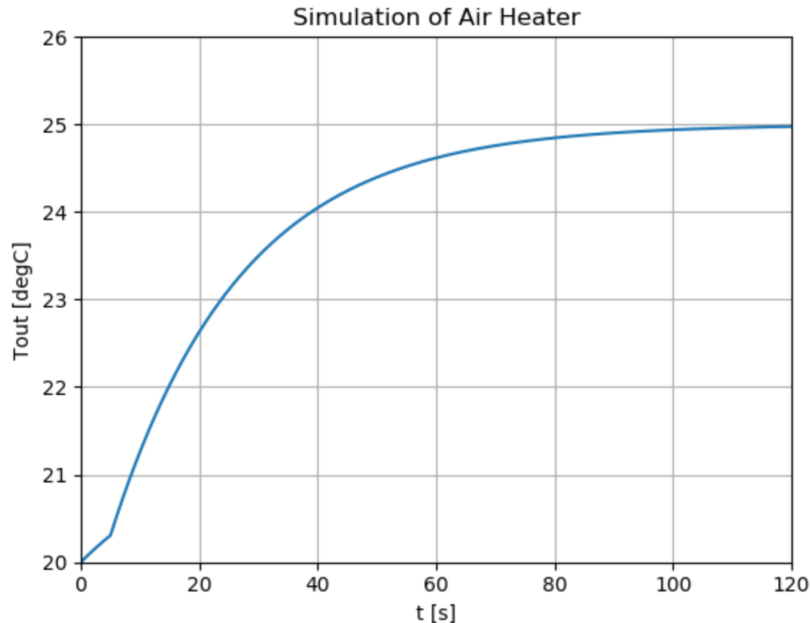
$$T_{out}(k+1) = T_{out}(k) + \frac{T_s}{\theta_t}\left\{-T_{out}(k) + \left[K_h u\left(k - \frac{\theta_d}{T_s}\right) + T_{env}\right]\right\}$$

Assuming $\theta_d = 2s$ and $T_s = 0.1s$ we get $u(k-20)$

This means that we must remember the 20 previous samples of $u(k)$

Python

Simulation of Discrete Air Heater with Time delay




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

# Model Parameters
Kh = 3.5
theta_t = 22
theta_d = 5
Tenv = 21.5

# Simulation Parameters
Ts = 0.1 # Sampling Time
Tstop = 120 # End of Simulation Time
N = int(Tstop/Ts) # Simulation length
Tout = np.zeros(N+2) # Initialization the Tout vector
Tout[0] = 20 # Initial Vaue

# Control Signal with Time Delay
N1 = int(theta_d/Ts)
u1 = np.zeros(N1)
u2 = np.ones(N)
uk = np.append(u1,u2)


# Simulation
for k in range(N+1):
    Tout[k+1] = Tout[k] + (Ts/theta_t) * (-Tout[k] + Kh*uk[k] + Tenv)

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

plt.plot(t,Tout)

# Formatting the appearance of the Plot
plt.title('Simulation of Air Heater')
plt.xlabel('t [s]')
plt.ylabel('Tout [degC]')
plt.grid()
xmin = 0; xmax = Tstop; ymin = 20; ymax = 26
plt.axis([xmin, xmax, ymin, ymax])
plt.show()
```

Air Heater Transfer System

The differential Equation for the Air Heater System:

$$\dot{T}_{out} = \frac{1}{\theta_t}\{-T_{out} + [K_h u(t - \theta_d) + T_{env}]\}$$

We need to find the following Transfer Function of the Air Heater: $H(s) = \frac{T_{out}(s)}{u(s)}$

By using Laplace we get the following Transfer Function for the Air Heater:

$$H(s) = \frac{T_{out}(s)}{u(s)} = \frac{K_h}{\theta_t s + 1} e^{-\theta_d s}$$

This is actually a standard 1.order Transfer Function with Time Delay

It is recommended that you know about Transfer Functions. If not, take a closer look at my Tutorial “Transfer Functions with Python”

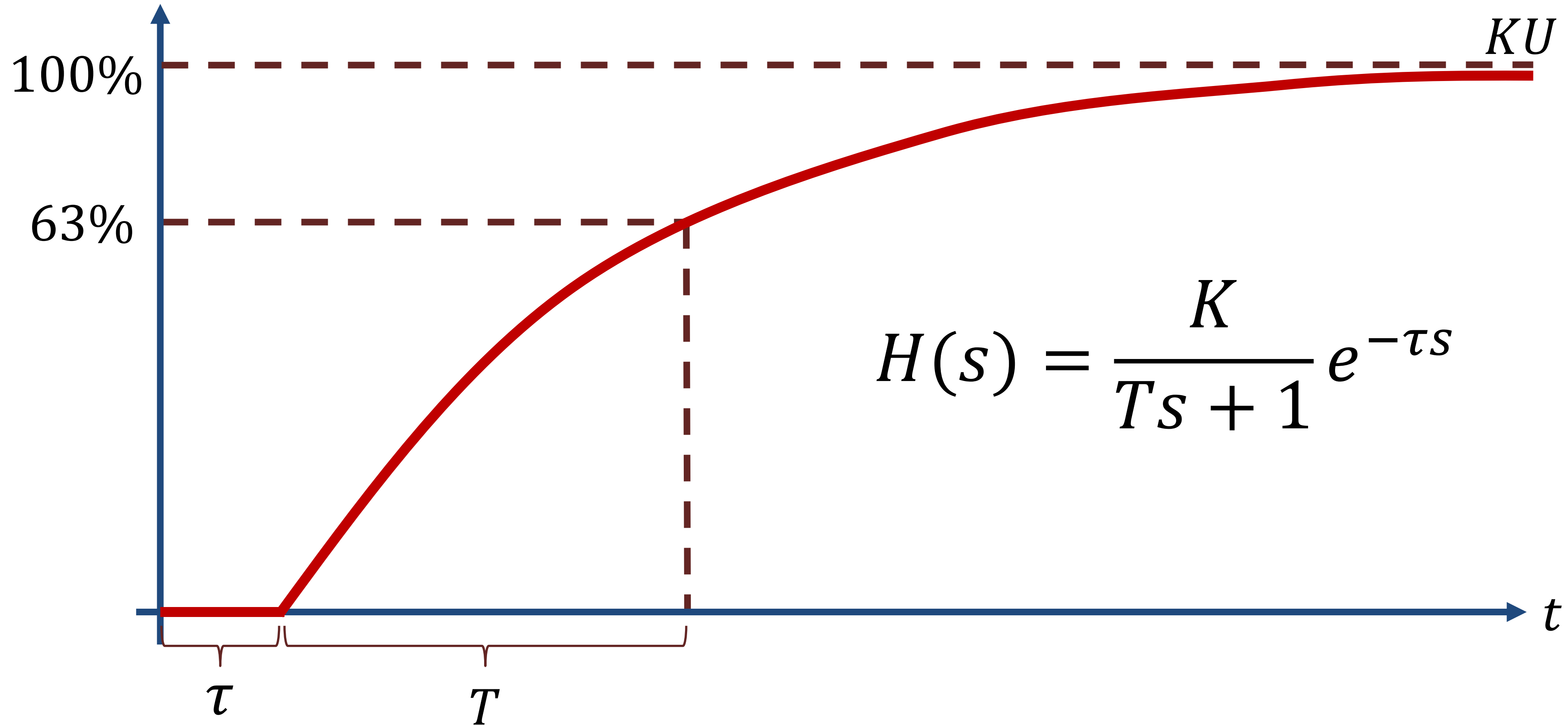
1.order Transfer Function with Time Delay

A general 1. order Transfer Function with Time Delay can be written as:

$$H(s) = \frac{K}{Ts + 1} e^{-\tau s}$$

Where K is the Gain, T is the Time constant and τ is the Time Delay

Step Response

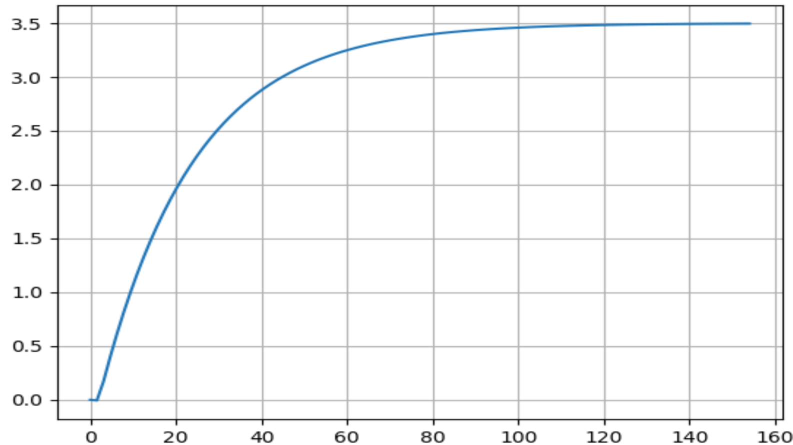


Python

Simulation of Air Heater Transfer Function:

$$H(s) = \frac{T_{out}(s)}{u(s)} = \frac{K_h}{\theta_t s + 1} e^{-\theta_d s}$$

We are using a Padé Approximation when implementing the Transfer Function:




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

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

# Transfer Function
num = np.array ([Kh])
den = np.array ([theta_t , 1])
H1 = control.tf(num , den)
print ('H1(s) =', H1)

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

H = control.series(H1, Hpade);
print ('H(s) =', H)

# Step Response
t, y = control.step_response(H)
plt.plot(t,y)
plt.grid()
plt.show()
```

PID Controller

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:

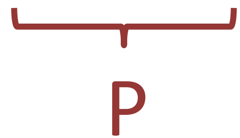
K_p Proportional Gain

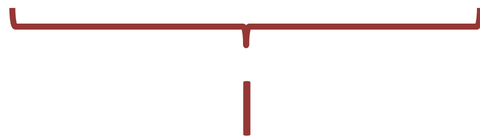
T_i Integral Time [sec.]

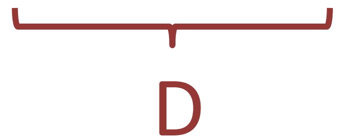
T_d Derivative Time [sec.]

The PID Algorithm

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







	Proportional Gain	Integral Time	Derivative Time
Tuning Parameters:	K_p	T_i	T_d

PID Controller Transfer Function

We have:

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

Laplace gives:

$$H_{PID}(s) = \frac{u(s)}{e(s)} = K_p + \frac{K_p}{T_i s} + K_p T_d s$$

or:

$$H_{PID}(s) = \frac{u(s)}{e(s)} = \frac{K_p(T_d T_i s^2 + T_i s + 1)}{T_i s}$$

PI Controller Transfer Function

We have:

Very often we just need a PI controller

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

Laplace gives:

$$H_{PI}(s) = \frac{u(s)}{e(s)} = K_p + \frac{K_p}{T_i s} = \frac{K_p T_i s + K_p}{T_i s} = \frac{K_p (T_i s + 1)}{T_i s}$$

Finally:

$$H_{PI}(s) = \frac{u(s)}{e(s)} = \frac{K_p (T_i s + 1)}{T_i s}$$

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:

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

Where $e_k = r_k - y_k$

Control System Design and Analysis

Hans-Petter Halvorsen

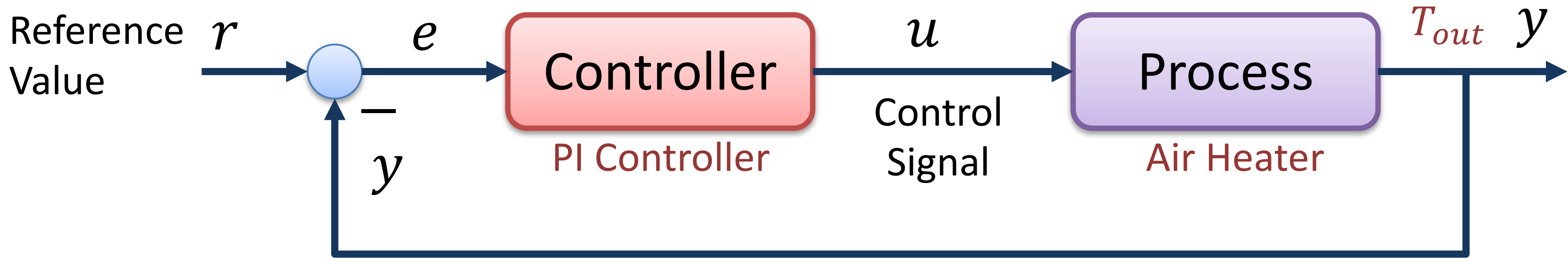
Design and Analysis

- We use the mathematical model to design and test proper PID parameters
- We will do some Stability Analysis
- Finally, we will implement a control system

Control System

$$H_{PI}(s) = \frac{u(s)}{e(s)} = \frac{K_p(T_i s + 1)}{T_i s}$$

$$H_P(s) = \frac{y(s)}{u(s)} = \frac{K_h}{\theta_t s + 1} e^{-\theta_d s}$$



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:

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

PI Controller

- We will use a PI Controller for this System
- There are lots of ways to find the PI parameters (K_p and T_i)
- I have used the “Skogestad Tuning Rules“ and found the following:

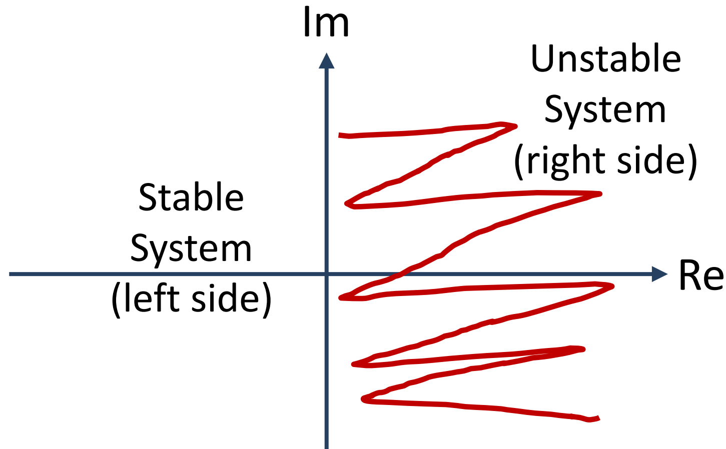
$$K_p = 0.52$$
$$T_i = 18s$$

- These values will used in the Stability Analysis of the system.
- Based on the Stability Analysis we will check the performance of the Control System and adjust/fine-tune the PI Parameters

Skogestad: http://folk.ntnu.no/skoge/publications/2001/tuningpaper_reno/tuningpaper_06nov01.pdf

Poles and Stability of the System

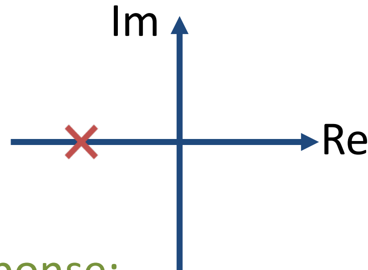
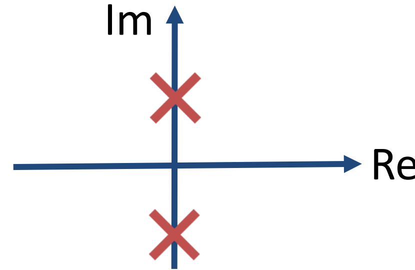
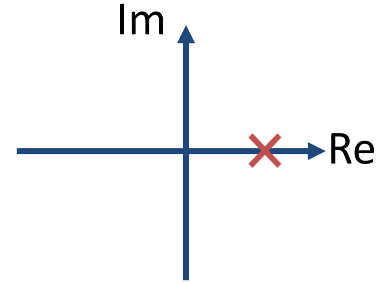
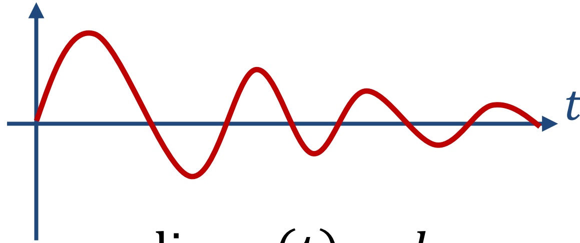
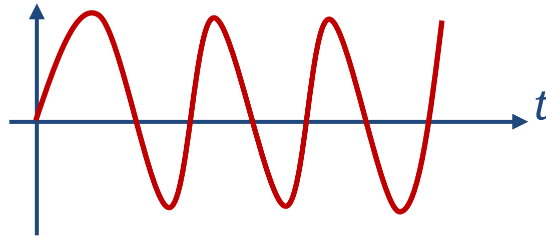
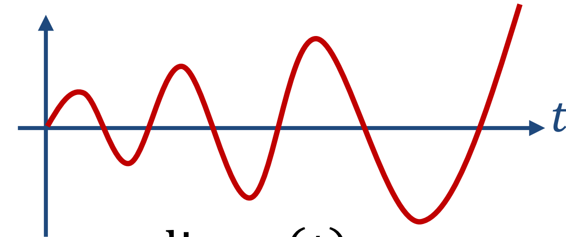
The poles are important when analyzing the stability of a system. The Figure below gives an overview of the poles impact on the stability of a system.



We have 3 different Alternatives:

1. Asymptotically Stable System
2. Marginally Stable System
3. Unstable System

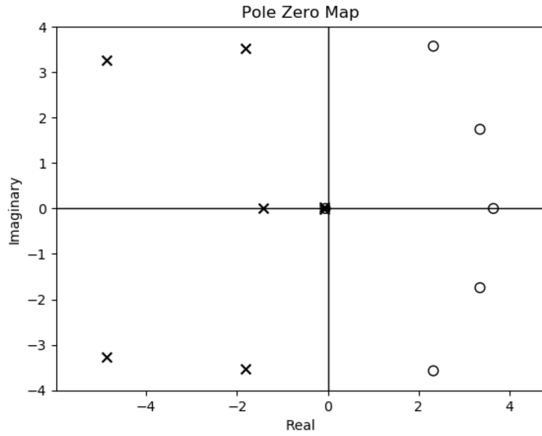
Stability Analysis

	Asymptotically Stable System	Marginally Stable System	Unstable System
Poles:			
Step Response:			
	$\lim_{t\to\infty} y(t) = k$	$0 < \lim_{t\to\infty} y(t) < \infty$	$\lim_{t\to\infty} y(t) = \infty$
Frequency Response:	$\omega_c < \omega_{180}$	$\omega_c = \omega_{180}$	$\omega_c > \omega_{180}$

```python
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)

# The Loop Transfer function
L = control.series(Hc, Hp)
print ('L(s) =', L)
# Tracking transfer function
T = control.feedback(L,1)
print ('T(s) =', T)
# Sensitivity transfer function
S = 1 - T
print ('S(s) =', S)
```

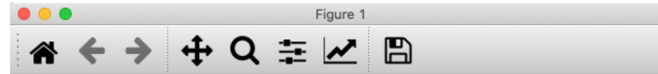
```python
# Step Response Feedback System (Tracking System)
t, y = control.step_response(T)
plt.figure(1)
plt.plot(t,y)
plt.title("Step Response Feedback System T(s)")
plt.grid()


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

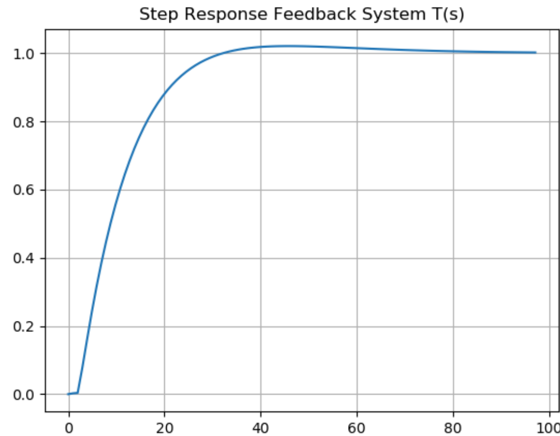
Results



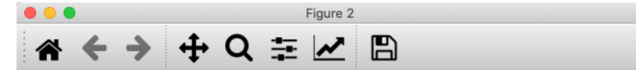
$$K_p = 0.52$$
$$T_i = 18s$$

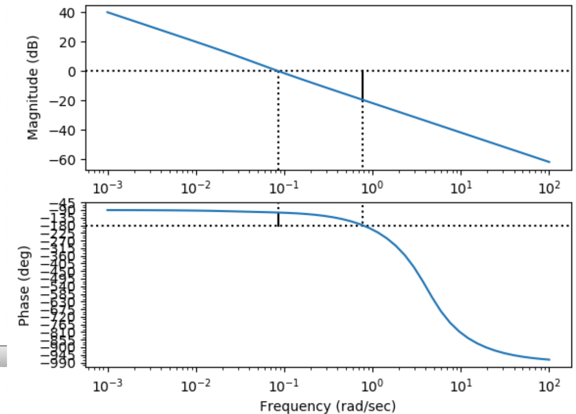
Step Response:





Frequency Response:





Poles:

Analysis

- We see that our margins could be a little smaller, so it is possible to increase K_p for our control system.
- How much can we increase K_p before the system gets unstable?
- The Critical Gain is as follows:

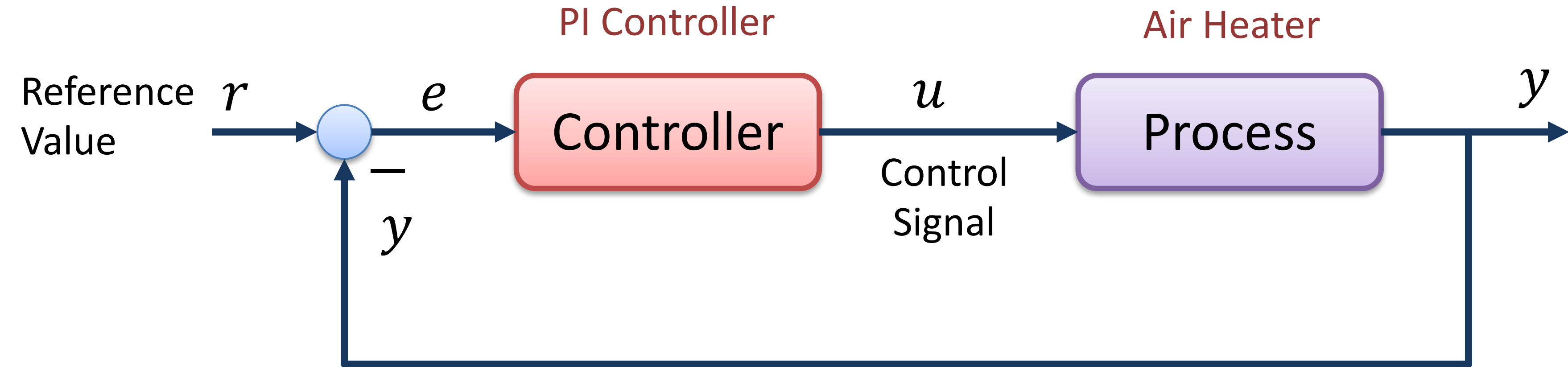
$$K_c = K_p \times \Delta K = 0.52 \times 9.41 \approx 4.89$$

- This means:
  - Asymptotically Stable System: $K_p < K_c$
  - Marginally Stable System: $K_p = K_c$
  - Unstable System: $K_p > K_c$

Control System implementation

Hans-Petter Halvorsen

Control System

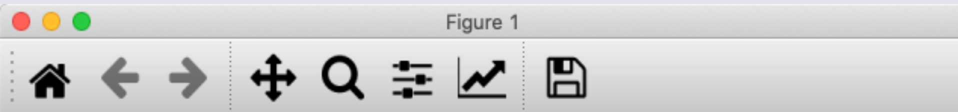


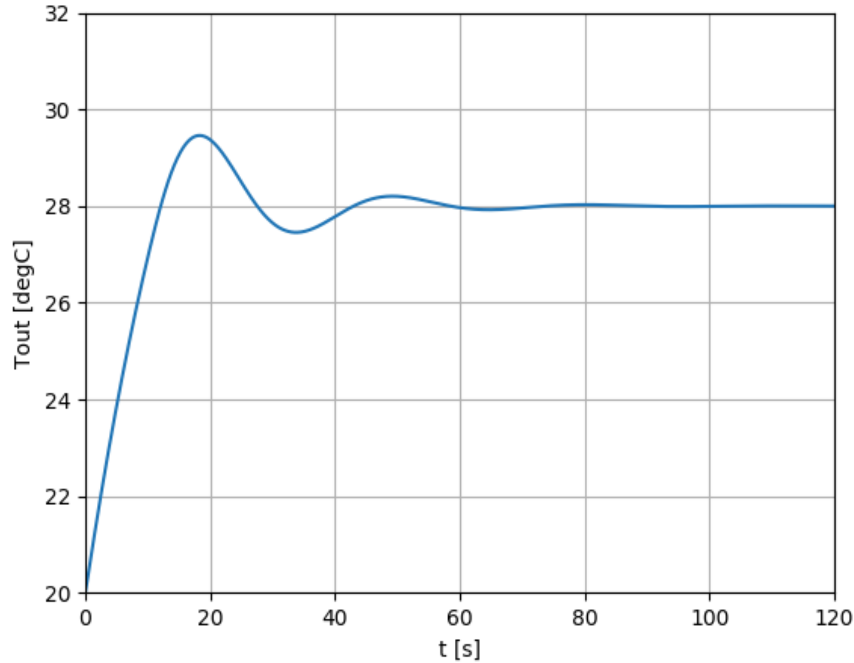
Control System (No Time Delay)

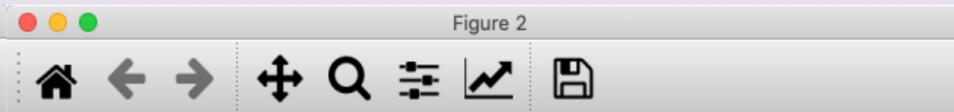
```
# Air Heater System
import numpy as np
import matplotlib.pyplot as plt

# Model Parameters
Kh = 3.5
theta_t = 22
Tenv = 21.5

# Simulation Parameters
Ts = 0.1 # Sampling Time
Tstop = 120 # 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 = 0.5
Ti = 18

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

# Simulation
for k in range(N+1):
    e[k] = r - Tout[k]
    u[k] = u[k-1] + Kp*(e[k] - e[k-1]) + (Kp/Ti)*e[k]
    if u[k]>5:
        u[k] = 5
    Tout[k+1] = Tout[k] + (Ts/theta_t) * (-Tout[k] + Kh*u[k] + Tenv)
```

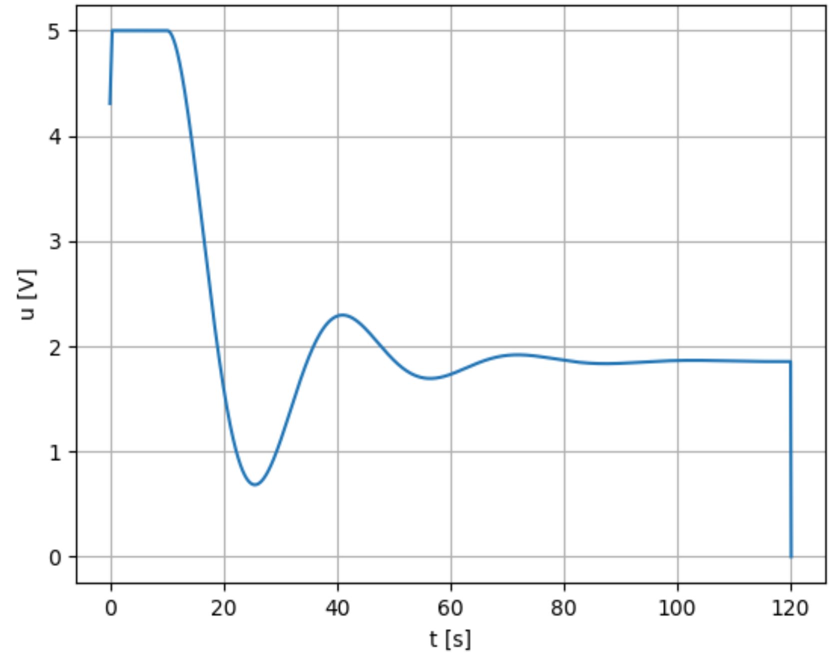
```
# Plot the Simulation Results
t = np.arange(0,Tstop+2*Ts,Ts) #Create the Time Series

# Plot Process Value
plt.figure(1)
plt.plot(t,Tout)

# Formatting the appearance of the Plot
plt.title('Simulation of Air Heater')
plt.xlabel('t [s]')
plt.ylabel('Tout [degC]')
plt.grid()
xmin = 0
xmax = Tstop
ymin = 20
ymax = 32
plt.axis([xmin, xmax, ymin, ymax])
plt.show()

# Plot Control Signal
plt.figure(2)
plt.plot(t,u)

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

Simulation Results







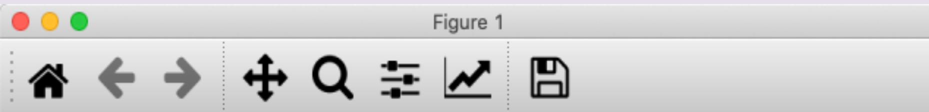


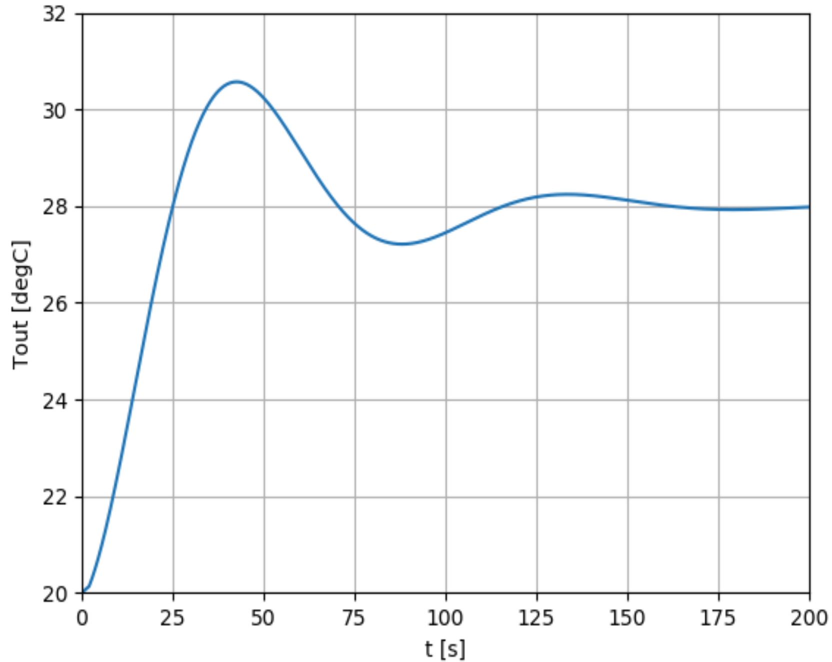
Control System (with Time Delay)

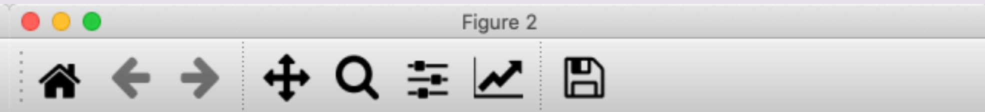
```
# Air Heater System
import numpy as np
import matplotlib.pyplot as plt

# 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

# Simulation
for k in range(N+1):
    e[k] = r - Tout[k]
    u[k] = u[k-1] + Kp*(e[k] - e[k-1]) + (Kp/Ti)*e[k]
    if u[k]>5:
        u[k] = 5
    Tout[k+1] = Tout[k] + (Ts/theta_t) * (-Tout[k] + Kh*u[int(k-theta_d/Ts)] + Tenv)
```

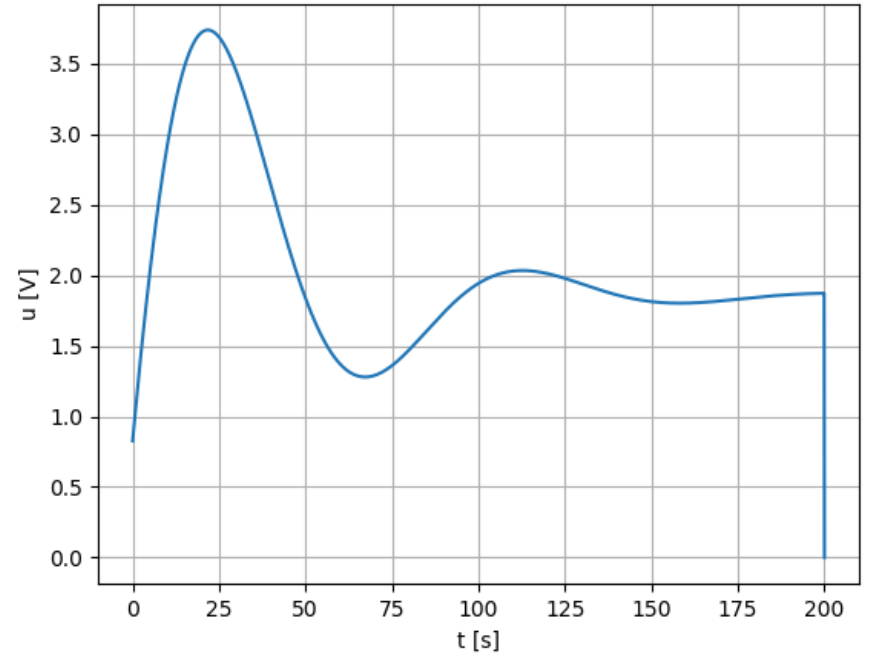
```
# Plot the Simulation Results
t = np.arange(0,Tstop+2*Ts,Ts) #Create the TimeSeries

# Plot Process Value
plt.figure(1)
plt.plot(t,Tout)

# Formatting the appearance of the Plot
plt.title('Simulation of Air Heater')
plt.xlabel('t [s]')
plt.ylabel('Tout [degC]')
plt.grid()
xmin = 0
xmax = Tstop
ymin = 20
ymax = 32
plt.axis([xmin, xmax, ymin, ymax])
plt.show()

# Plot Control Signal
plt.figure(2)
plt.plot(t,u)

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

Simulation Results









Real Time Plotting

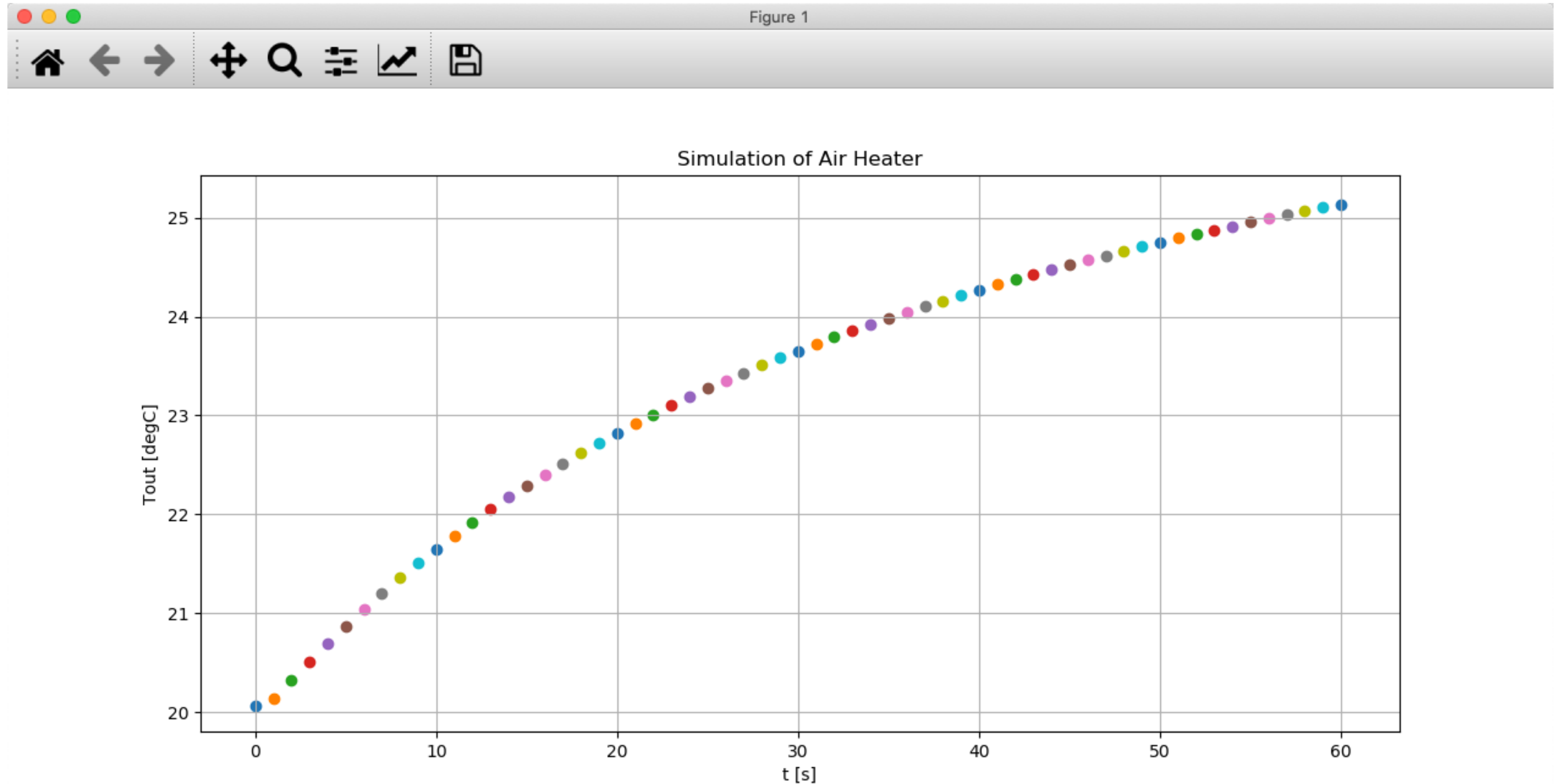
- In the simulations so far, we do the simulations inside a For loop.
- Then when the For loop is finished, we plot the simulation results
- It is better to see the simulation results immediately in each iteration inside the loop
- Here we have many possible solutions to make this happen:
  - We can use the **scatter()** function in the matplotlib library
  - For more advanced features, we can use the **animation module** in the matplotlib library (matplotlib.animation)
  - etc.

Real-Time Control System

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

# Model Parameters
Kh = 3.5
theta_t = 22
theta_d = 2
Tenv = 21.5

# Simulation Parameters
Ts = 1 # Sampling Time
Tstop = 60 # End of Simulation Time
N = int(Tstop/Ts) # Simulation length
Tout = np.zeros(N+2) # Initialization
Tout 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 Time Series
```

```
# Formatting the appearance of the Plot
plt.figure(1)
plt.title('Simulation of Air Heater')
plt.xlabel('t [s]')
plt.ylabel('Tout [degC]')
plt.grid()

# Simulation
for k in range(N+1):
    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
    Tout[k+1] = Tout[k]+(Ts/theta_t)*(-Tout[k] + Kh*u[int(k-theta_d/Ts)]+Tenv)

    plt.scatter(t[k],Tout[k+1])
    plt.pause(Ts)
```

Simulation Results



Adjustment of Plot

```
plt.scatter(t[k],Tout[k+1])
```

Some Parameters:

```
plt.scatter(t[k],Tout[k+1], marker='o', c='b', alpha=0.5, s=20)
```

Symbol
o – circle
* – star
etc.

Color
b = blue
g = green
etc.

Transparency

Size of the Symbol (actually Area of the Symbol)

We don’t need to use the **scatter()** function, we can use the standard **plot()** instead:

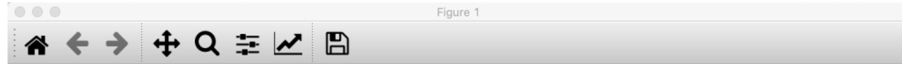
```
plt.plot(t[k],Tout[k+1], '-o', markersize=5, color='blue')
```

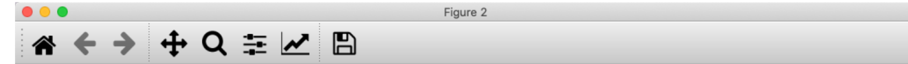
Modified Solution

```
# 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)
```

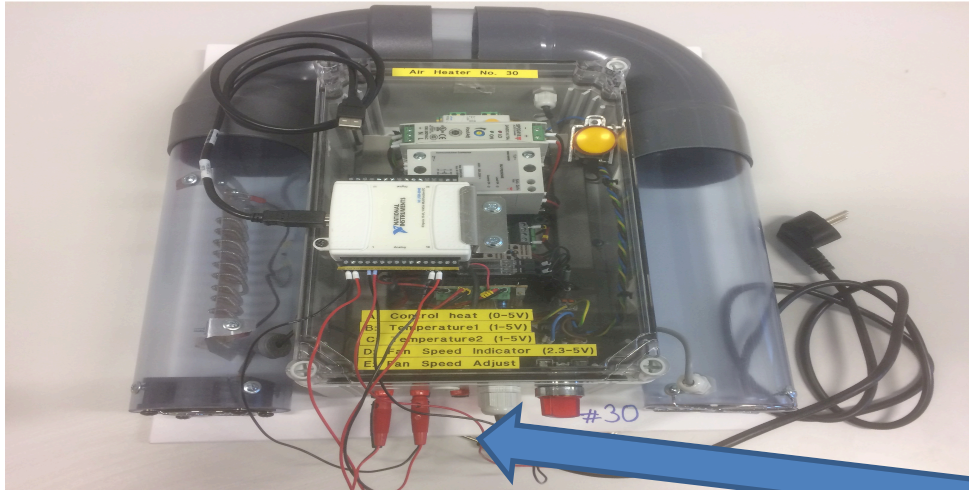
Simulation Results





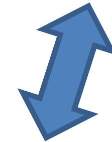
What's Next?

Control the Real System



Small-scale Industrial Process

PC



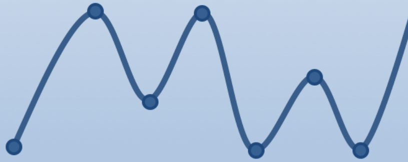
I/O Module

This I will demonstrate and Explain in another Tutorial

Additional Python Resources

Python Programming

Hans-Petter Halvorsen



https://www.halvorsen.blog

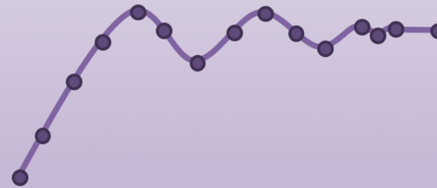
Python for Science and Engineering

Hans-Petter Halvorsen

https://www.halvorsen.blog

Python for Control Engineering

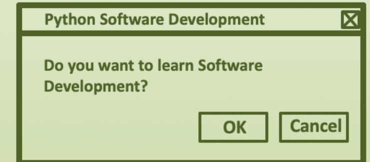
Hans-Petter Halvorsen



https://www.halvorsen.blog

Python for Software Development

Hans-Petter Halvorsen



https://www.halvorsen.blog

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