Universitetet i Sørøst-Norge

Coursework Requirement 2: Heater Process and PID Controller

Industrial Automation (IA3112)

Michael Winge (229507, IA)

USN Porsgrunn

September 2020

Nomenclature

CV	Control Variable
PI	Proportional-Integral
PID	Proportional-Integral-Derivative
PV	Process Variable
RZN	Relaxed Ziegler-Nichols
SP	Setpoint
ZN	Ziegler-Nichols

Table of Contents

1 Introduction

The purpose of this coursework is to use a high-level programming language such as Python to write a script that simulates the temperature regulation of a heating process, as it stabilizes at a set point using a PID Controller, and to test various methods with which the PID Controller can be tuned.

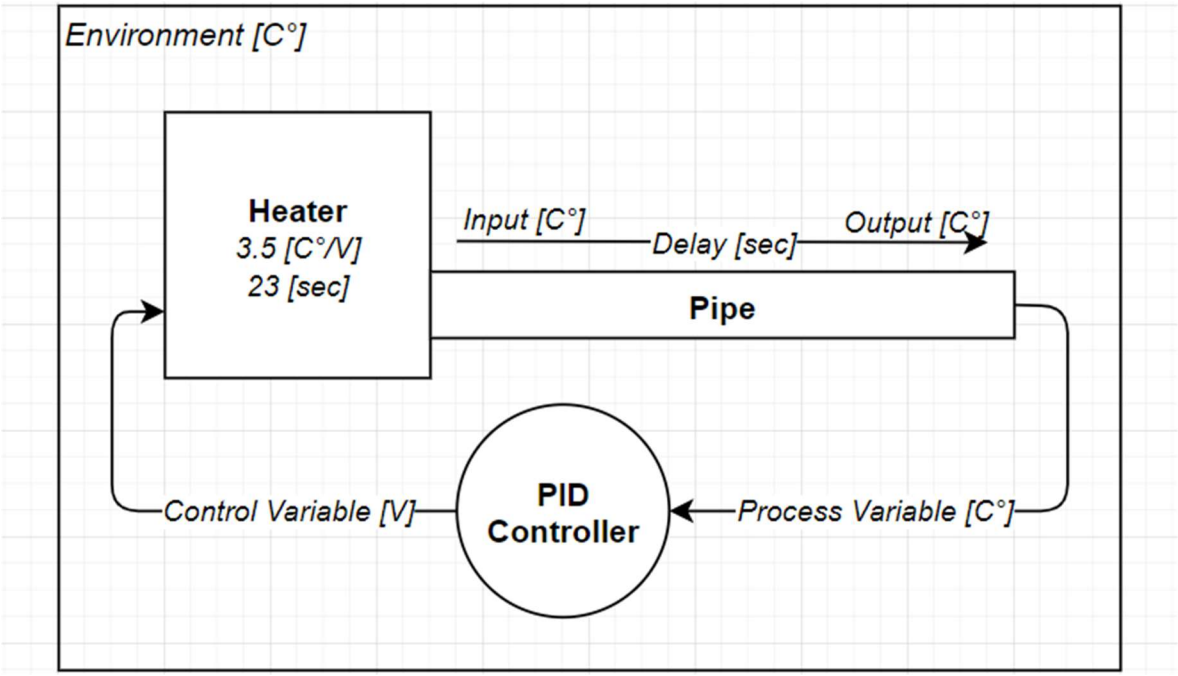

Figure 1-1: A crude diagram of how the Process and PID Controller interact.

The heater should have a minimum and maximum voltage capacity between 0-5 V, and a pipe that causes a 3 second delay. It should be placed in a room where the environment temperature is 20 degrees Celsius, and the script should simulate the temperature reading from the heater's pipe output from which the PID Controller can regulate the heater at a regular interval of 0.05 seconds across 150 seconds.

The differential means of an On-Off Controller and a PID Controller used to regulate the process at a given SP should be reported. The Pk and Ti values of the PID Controller should be set with the ZN PI, RZN PI, and Skogestad PI methods in separate installments, and disturbance tests should be performed on each of them; fluctuation in both set point and environment temperature.

The script must be written in Python so that it may be readable by any potential participants and supervisors of the coursework, and the heater must use a set of constant values determined from its realistic equivalence found at the USN laboratory.

2 Simulator

The Simulator imports the PIDController and Heater class libraries found in the Library folder, and is set to run once every 0.05 second over the course of 150 seconds, resulting in a total of 3000 intervals.

Table 2-1: Constants used to configure the simulator.

Constants	Value
DeltaTime	0.05 [sec]
Start	0 [sec]
End	150 [sec]

The Process and PID Controller are made into class libraries due to the length of the script and need for structure.

2.1 Process

The process is split further into two class libraries, the Heater and Pipe, where the former instantiates the latter. Combined, they make up the entirety of the process and its mathematical model.

2.1.1 Heater

The Heater class object is found in the Heater.py file, and is instantiated with environment temperature, K, T, delay, and DeltaTime as parameters. DeltaTime is added specifically because the Heater is dependent on the Pipe class object which uses DeltaTime to determine the physical length of the delay. A full list of properties used for the heater in this simulator can be found in the table below.

Table 2.1.1-2: The heater properties used in this simulator, where K and T determine temperature over time.

Properties	Value
Environment temperature	20 [C°]
K	3.5 [C°/V]
T	23 [sec]
Delay	3 [sec]

The SetVoltage method in the Heater contains the main part of the mathematical model and takes voltage and DeltaTime as parameters to adjust the temperature of the heater. From there, the temperature is released into the Pipe using its Input method.

```
def SetVoltage(self, voltage, dt):
  temperature += (k * voltage - temperature) * dt / t
  pipe.Input(temperature)
```

Figure 2.1.1-1: A compact snippet of the partial mathematical model found in the Heater.SetVoltage method.

2.1.2 Pipe

The Pipe class object is in the Pipe.py file, and is considered a component of the Heater class object, where it serves to delay the PV. As values are added to the Pipe's intake using the Input method, the values are inserted at the beginning of a list of temperatures, where they accumulate until the length of the list exceeds the delay over DeltaTime. These delayed values are retrieved via the GetOutput method.

```
temperatures.insert(0, temperature)
```

Figure 2.1.2-1: A compact snippet of the contents in the Input method that serves to delay the temperature value.

```
if len(temperatures) >= delayDt:
  temperature = temperatures[-1]
  temperatures.pop()
return temperature
```

Figure 2.1.2-2: A compact snippet of the contents in the GetOutput method that serves to retrieve the delayed value.

2.2 PID Controller

The PIDController class object is used to calculate the CV that regulates the process, and can be found in the PIDController.py file. It requires a valid SP, and the heater's minimum and maximum voltage capacity as parameters upon instantiation. Its SP, Manual, Pk and Ti properties can be set via the object directly, and manipulated further via the TuneZNPI, TuneRZNPI, and TuneSkogestad methods. Using any of these methods will reset the Manual joint of the PID to 0.

The CV is returned via the GetControlVariable method and requires a PV and DeltaTime as parameters. This method calculates the differential error between the SP and given PV, and uses it to determine the PI gains required to reach the SP in an optimal fashion. That is, if Kp and Ti has been tuned properly beforehand.

```
error = SP – pv
p = Kp * error
i += (Kp / Ti) * error * dt
i = Clamp(i)                          # Clamp integral
d = Kp * Td * ((pv - pvLast) / dt)
pvLast = pv
return Clamp(M + p + i + d)           # Clamp CV
```

Figure 2.2-1: A compact snippet of the content in the PIDController.GetControlVariable method.

Anti-windup is important. If the integral joint is not clamped between the min-max voltage capacity of the heater, the integral will accumulate a value disproportionate to its actual size. The entire CV expression must finally be clamped to avoid it from exceeding the voltage capacity of the heater.

2.2.1 Ziegler-Nichols PI Tuning

One way to go about tuning the PID Controller is to use the ZN PI method, where the Pk value is initially set low, and the Ti value is set high, and the manual joint of the PID Controller is raised to a point that allows the CV to reach the PV. Once complete, the Pk value can slowly be increased until each wave on the graph appears to be of near equal height and length, but carefully, so that CV does not encroach on the boundaries of the heater's voltage capacity. This Pk value is then used along with a wavelength in the TuneZNPI or TuneRZNPI methods to adjust the Pk and Ti values, so that the PID Controller may function as intended.

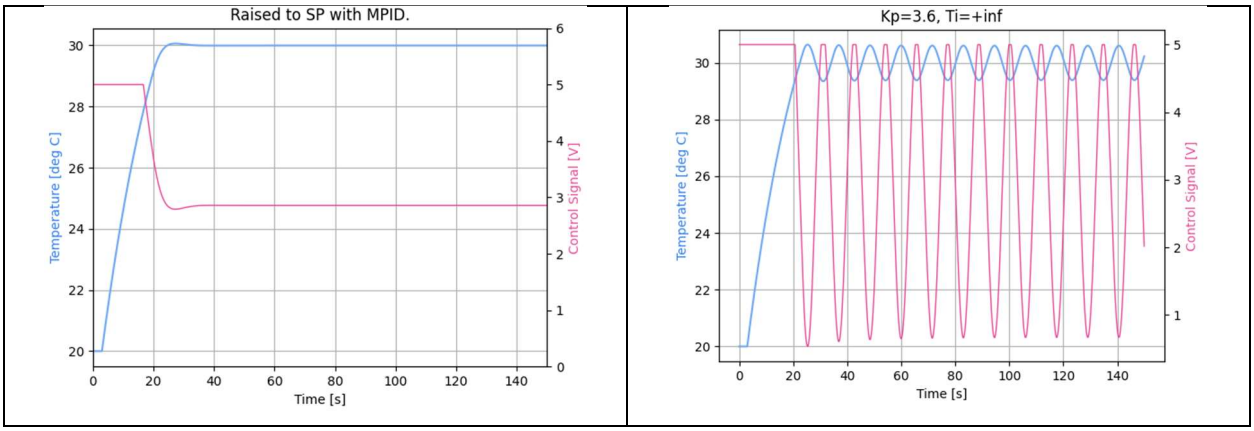

Figure 2.2.1-1: Graphs illustrating the steps taken in tuning with ZN PI and RZN PI. The result is discussed 3.2.

The values found in the graph above is that the ZN PI method requires a kpu of 3.6, and a wavelength pu of 11.55.

2.2.2 Skogestad PI Tuning

A different approach is required when tuning with the Skogestad PI method. Instead of letting the PID Controller reach SP, the SP is instead reached by setting the voltage to a constant in the SetVoltage method. (10 / k) will result in a temperature of 30° C, (12 / k) will result in 32° C, and so on, where k is the constant used in the heater. The steepest ascent of the tangent line, typically found post-delay, is then divided by the voltage increase, and used along with the delay in the TuneSkogestad method. The Tc value is set at τ (tau), then gradually reduced, or increased, until a satisfying result is made.

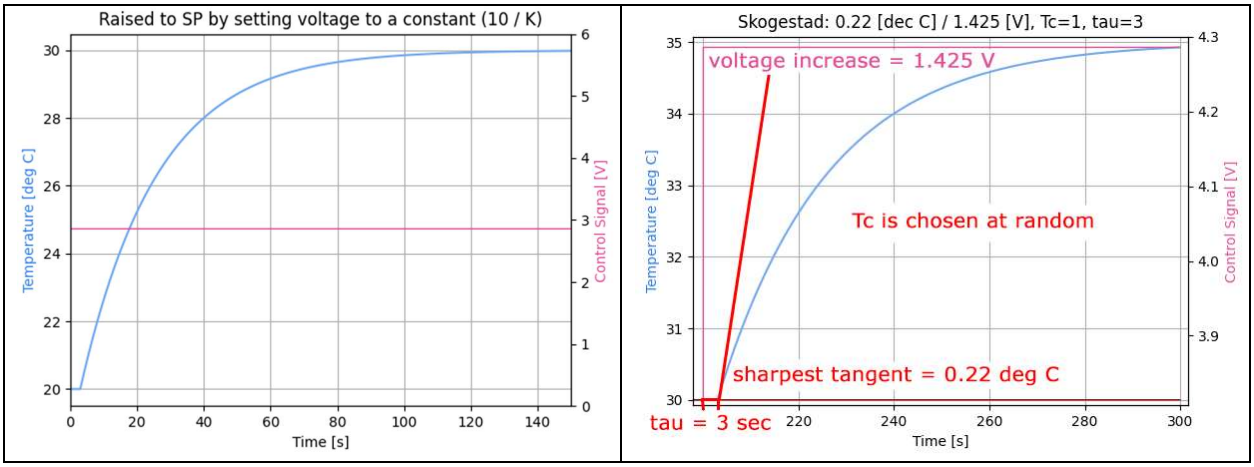

Figure 2.2.1-1: Graphs illustrating the steps taken in tuning with the Skogestad PI method, where the lower half of the voltage increase, starting at roughly 2.86 V and ending at 4.3 V, is not entirely visible on the graph. The result is discussed in 3.3.

The values used to tune with the Skogestad PI method is a tangent of 0.22 [C°] / 1.425 [V], a Tc of 1, and a τ of 3.

3 Results

3.1 Ziegler-Nichols PI

One of the key features of the ZN method, or perhaps the PID controller -- since the result will be the same regardless of tuning method if the Pk and Ti values in the different methods fall close to one another -- is that the result should yield a characteristic quarter wave decay, in which there are both under- and overshoots.

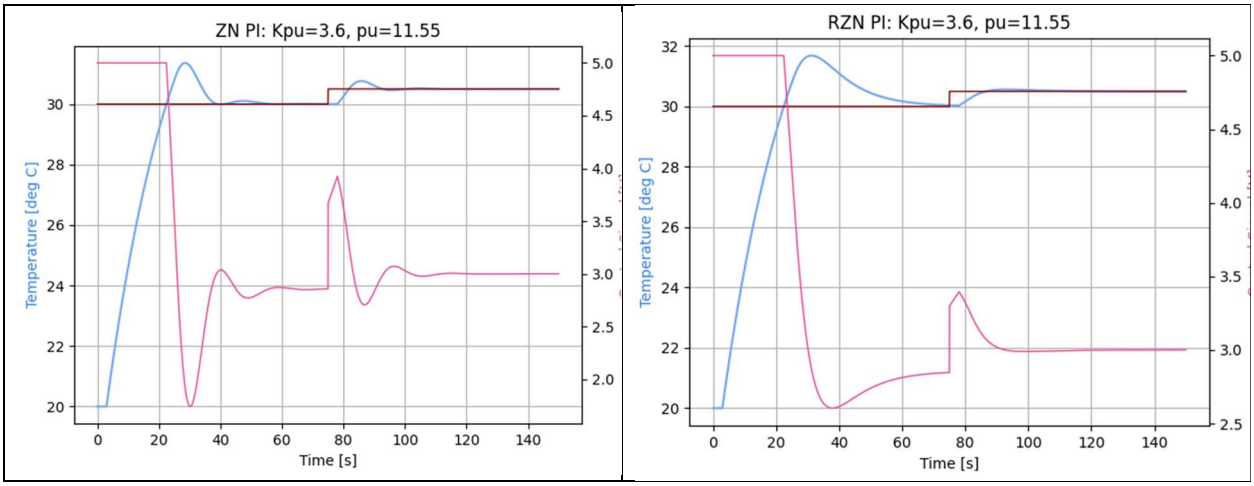

Figure 3.1-1: Graphs illustrating the difference between ZN PI and RZN PI, where both reach SP at roughly 22.5 seconds, but RZN results in less oscillation at the expense of taking longer to calm at SP.

Due to the large amount of windup in the integral joint of the PID at startup, however, where the voltage is set to maximum for 22.5 seconds, the ZN PI method appears to have almost no undershoot at all. By setting the SP value from 30 to 29 after the temperature has had time to settle, which is a value that does not encroach on the voltage boundary too much, the quarter wave decay becomes clearer. Example given in figure 3.1-2.

A common pattern in these results is that the wavelength is pretty much persistent around 11.5. This wavelength after considerable tests, speaks almost directly to the size of the T or τ in the process. If it takes longer to warm up, the wavelengths will reduce, and if it takes almost no time at all to warm up, the wavelengths will increase. If the K value, however, is increased or reduced, the wavelength remains at about the same length as before, at around 11.5.

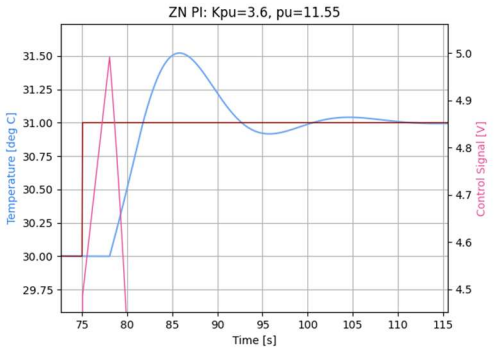

Figure 3.1-2: Graph illustrating oscillation with less windup.

Once the wavelength is discovered, the Pk can be further adjusted to find certain sweet spots. For instance, if there is a desire to reach SP as quickly as possible, but to not hit the voltage boundaries, the Pk can be reduced. Whereas if the desire is to reach the setpoint as quickly as possible, regardless of oscillation, the Pk can be increased. If there is a need for less oscillation, the Pk can

either be reduced, or the relaxed variant of the ZN PI method can be used. Ultimately, tuning depends on the Use Case.

3.2 Skogestad PI

With the Skogestad PI method the primary interest is to see how the controller reacts when the SP and environment temperature is disturbed, as well as to see what happens when the delay of the heater (τ) is for whatever reason distorted without also resetting the τ in the already tuned controller.

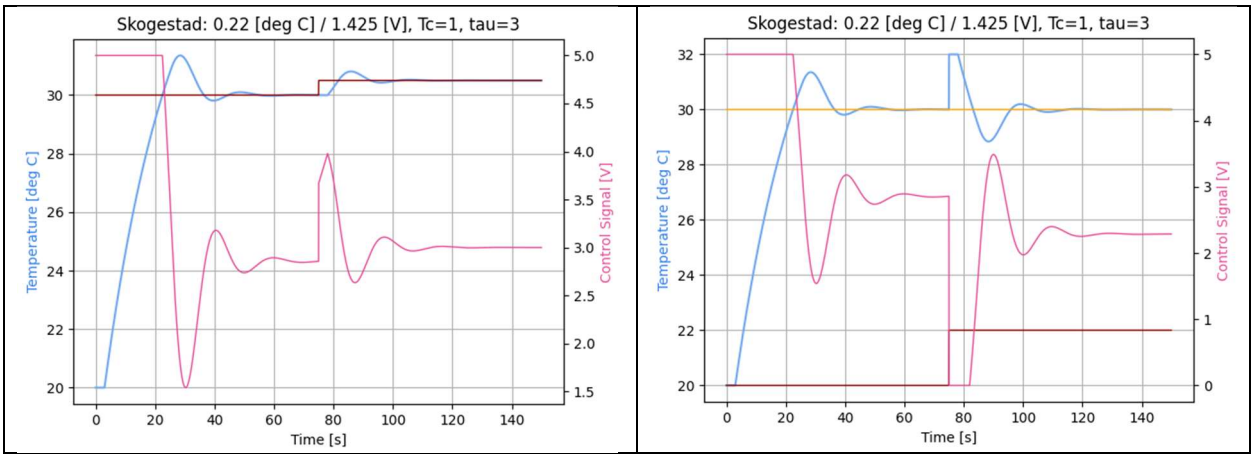

Figure 3.2-1: Graphs illustrating the PID controller tuned with the Skogestad PI method, where the result on the left received a change in SP, and the result on the right received a change in environment temperature.

Tuning the PID Controller with the Skogestad PI method makes the controller behave similarly as when it is tuned with the ZN PI method seen in the left graph of Figure 3.1-1. This is because the Tc value has been set to 1. If Tc is set back to 3 it will look more like RZN PI in the right graph of Figure 3.1-1. This will later be discussed in 3.3.

There is really no difference in response in the various tuning methods when the environment temperature is disturbed, neither.

If the process τ is increased above 4.75 using the settings seen in the title of the graph to the right, then the equation will become imbalanced, and control will be lost. When Tc is increased to 3, this delay can be increased as far as 7.75 before control is lost.

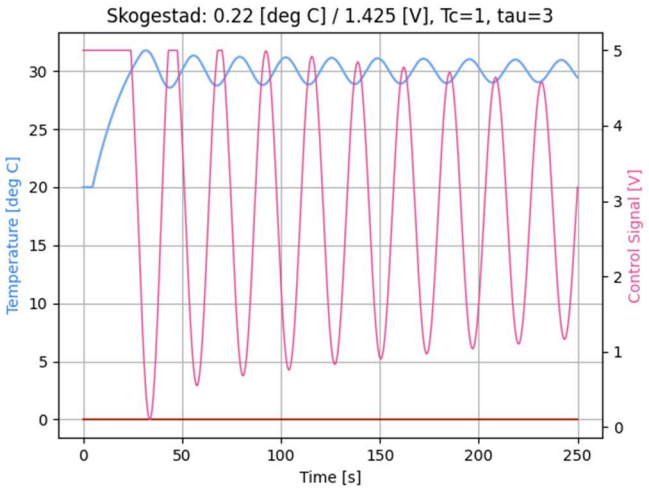

Figure 3.2-2: Graph illustrating what happens when the process delay is increased from 3 [sec] to 4.75 [sec], whilst using the same configuration as before.

3.3 Differential Average and Mean

The differential average and mean are calculated for both the On-Off Control system that was given with the coursework and the ZN tuned PID Controller. The area of interest is between 50 and 150 seconds, found by slicing the array containing the temperature output.

Table 3.1-1: The differential mean and average results of using various controllers and tuning methods.

	Kpu	pu	Tangent	Tc	τ	Mean	Average
On-Off						*-0.153*	*-0.185*
PID ZN PI	3.6	11.55				0.041	0.003
PID RZN PI	3.6	11.55				0.224	0.046
PID Skogestad PI			0.22 / 1.425	1	3	0.033	0.001
PID Skogestad PI			0.22 / 1.425	3	3	0.077	0.012

While it should already be obvious that the On-Off Controller will result in a repeated case of under- and overshoots, thus resulting in a larger mean and average, what this table suggests in regards to the ZN PI methods is that the relaxed variant takes longer to settle at SP. Graphs are however required to illustrate oscillation.

In 3.2 it was claimed, not proven, that the PID Skogestad PI and RZN PI *looked* similarly, and although graphs have not been added to the report to support this claim, the result can be visualized in the table above. That is, it takes considerably longer, especially on average, for the Skogestad PI method with a Tc of 3 to settle down, than a Tc of 1, much like the RZN PI method. This is something that one might not pick up on just by looking at the graph if precision is required.