

Control theory and applications: laboratory assignment

PID control with feedforward
using the Temperature Control Lab kit

3beau3C

Academic year 2022-2023

Franky De Bruyne
F.DeBruyne@ecam.be

February 7, 2023

1 Problem statement

This control laboratory is about **applying** the **control theory** of the theoretical course “Control theory and applications” in practice on the Temperature Control Lab kit also known as TCLab. Refer to <http://apmonitor.com/pdc/index.php/Main/ArduinoTemperatureControl>.

TCLab is a pocket-sized lab with software in Python, MATLAB, and Simulink for the purpose of reinforcing control theory for students. Refer to Figure 1.

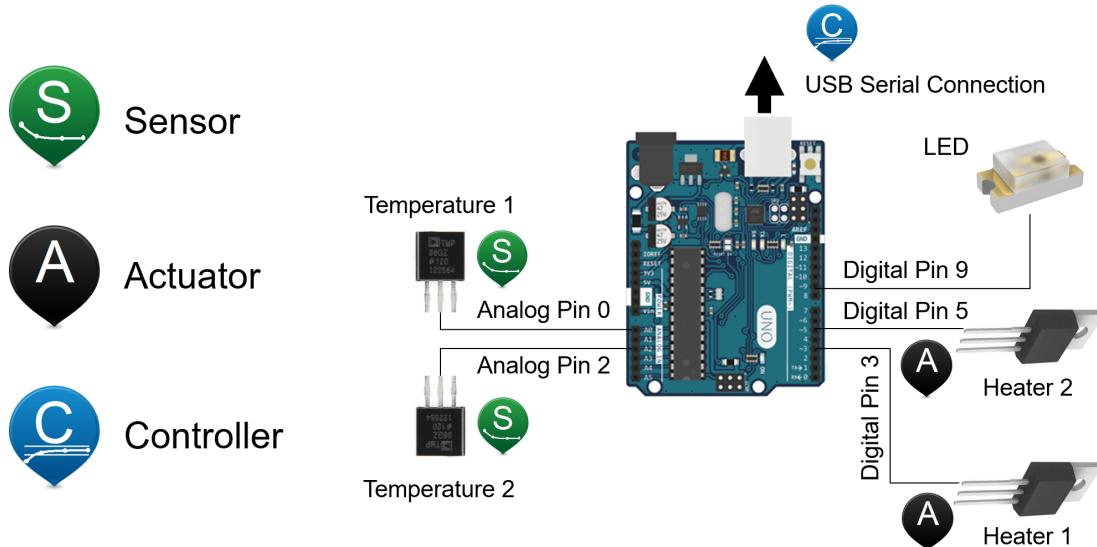


Figure 1: Temperature control lab or TCLab for short

TCLab is a simple **multivariable system** comprised of **2 heaters** and **2 temperature sensors**.

In this control laboratory, we will consider the **single variable system** in between the **manipulated variable MV**, the heating power of heater 1 [%], and the **process variable PV**, the temperature of sensor 1 [°C]. We will use *DV*, the heating power of heater 2 [%] as a **disturbance variable**. Indeed, through a coupling effect, heater 2 has an effect on the temperature sensor 1 through conduction¹, convection and radiation:

1. Make sure that you have the **Anaconda open-source individual edition** running on your PC before the first laboratory session. Refer to
<https://www.anaconda.com/products/individual>.
for the installation details.
2. **Install the package tclab** before the first laboratory session. Go to the anaconda prompt as shown in Figure 2 and type: “`pip install tclab`”

¹This requires a small change on the set-up, i.e. a metal connection between the two heat sinks.

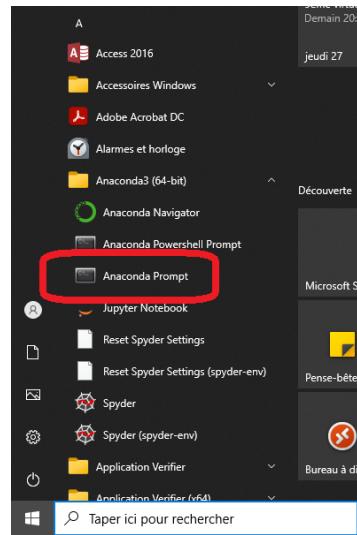


Figure 2: Anaconda prompt

Also refer to

[https://docs.anaconda.com/anaconda/navigator/tutorials/manage-packages/.](https://docs.anaconda.com/anaconda/navigator/tutorials/manage-packages/)

3. Test the Python package `package_DB.R.py` and the JupyterLab file `package_DB.R.ipynb` which contains example code **before** the first laboratory session. These files are contained in the zip file `Control_theory_software.zip` that is readily available on the CLACO platform. Refer to <https://claco.ecam.be/#/desktop/workspaces/open/3bect30/resources/laboratory-2>.

- Make sure that you can run the code in the JupyterLab environment. You might have to **install several packages** at the Anaconda prompt. Also refer to <https://docs.anaconda.com/anaconda/navigator/tutorials/manage-packages/>.
- **Optional but strongly recommended:** analyze and understand the Python package code in `package_DB.R.py` and learn to use it in the JupyterLab environment using the JupyterLab example code in the file `package_DB.R.ipynb`.

4. Install the **Arduino software** **before** the first laboratory session. Refer to

<https://www.arduino.cc/en/Main/Software>.

5. Make sure that you know how to **handle TCLab** **before** the first laboratory session. Refer to Figures 3 to 5 and refer to

<https://apmonitor.com/pdc/index.php>Main/ArduinoSetup>.

A few **security warnings**:

- **Never touch the heat sinks !** The temperatures can rise up to 100°C.
- **Never touch the electronics when powered !**
- Beware **not to bend the heat sinks and destroy the thermal paste connection** with the temperature sensors when **boxing TCLab**.



Figure 3: How to connect TCLab

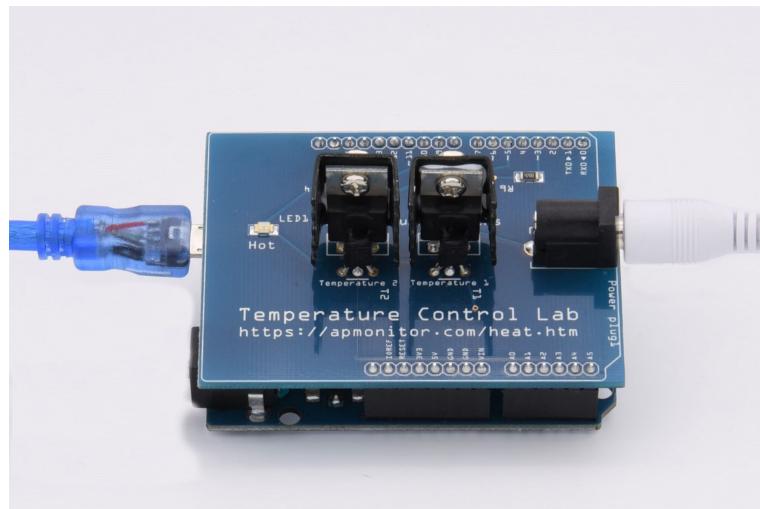


Figure 4: Correct power connection

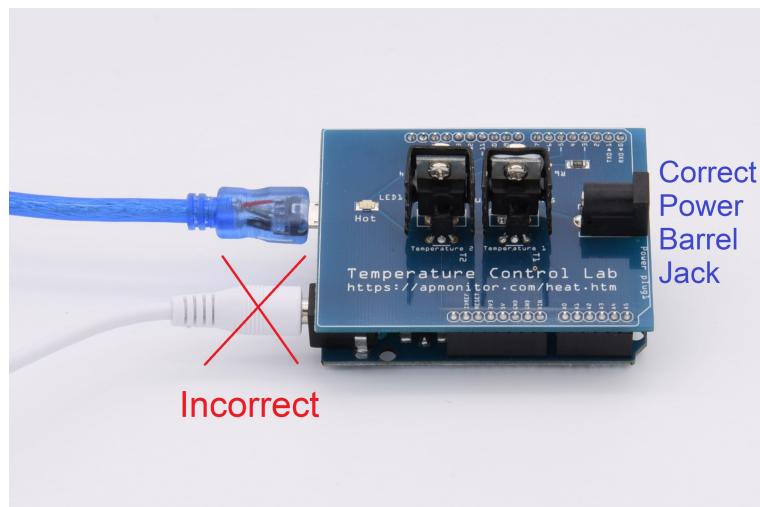


Figure 5: Incorrect power connection

6. Open the **Arduino sketch** `tclab_v2_DBR.ino` that is readily available on the CLACO platform (Unzip the file `Control_theory_software.zip`), i.e. refer to
<https://claco.ecam.be/#/desktop/workspaces/open/3bect30/resources/laboratory-2>.
Compile the code and **upload** it to the TCLab Arduino Leonardo board.
7. Run the **test program** using the JupyterLab file `TCLab_Test.ipynb` that is readily available on the CLACO platform (Unzip the file `Control_theory_software.zip`), i.e. refer to
<https://claco.ecam.be/#/desktop/workspaces/open/3bect30/resources/laboratory-2>.
This presupposes that you will have **installed the package `tclab`** ! Refer to item 2.
8. **Identify** the
 - **process dynamics:** $MV = \text{heating power } HP_1 [\%]$ to temperature $PV = T_1 [^\circ\text{C}]$,
 - **disturbance dynamics:** $DV = \text{heating power } HP_2 [\%]$ to temperature $PV = T_1 [^\circ\text{C}]$.
Use the Python package `package_DB.R.py` and the JupyterLab files
 - `TCLab_OLP.ipynb`
 - `Identification_FOPDT.ipynb`
 - `Identification_SOPDT.ipynb`
that are readily available on the CLACO platform (Unzip the file `Control_theory_software.zip`), i.e. refer to
<https://claco.ecam.be/#/desktop/workspaces/open/3bect30/resources/laboratory-2>.
Proceed as follows:
 - a. **Experimentation** on MV :
 - Run the JupyterLab file `TCLab_OLP.ipynb` with `ExpVariable = "MV"`.
 - This will produce a comma separated text file with header in the folder `Data` with the experimental data. This data set will be used for identification of the input-output dynamics in the next step.
 - Understand the meaning of the **experimentation parameters**.
 - b. **Identification of the input-output dynamics:** use the JupyterLab files
 - `Identification_FOPDT.ipynb` and
 - `Identification_SOPDT.ipynb`
to obtain First Order Plus Dead-Time (FOPDT = Broida) and Second Order Plus Dead-Time (SOPDT = van der Grinten) models.
 - c. **Graphical methods:** obtain **Broida**, **van der Grinten** and **Strejc** models via the **graphical methods** outlined below. Use a print out with experimental step response that was obtained above.
 - d. **Model validation:**
 - Use JupyterLab to compare the **actual step** response with the **simulated step responses** obtained from the models identified above.
 - Use JupyterLab to compare the models identified above in the **frequency domain**.
Implement your code in a JupyterLab file named `Simulation_OLP.ipynb`.
 - f. Redo the **same operations** as above with MV replaced by DV **without** going through steps c. and d.
 9. Create your **own Python package** `package_LAB.py` to implement a **Lead-lag function**. Use a JupyterLab file `package_LAB.ipynb` to **test your code**.

10. Implement a **PID controller function** in your **Python package** `package_LAB.py` created above. Use a JupyterLab file `package_LAB.ipynb` to test your **PID code** in **open-loop**.
11. Implement **your own discrete-time PID controller with feedforward structure**. Go for **intermediate milestones**, i.e. start by implementing the PID controller and add the feedforward structure afterwards.
 - a. **Simulate** the discrete-time PID controller with feedforward structure in closed-loop on a SOPDT model with the parameters identified above. Implement your code in a JupyterLab file named `Simulation_CLP_PID_FF.ipynb`.
 - b. **Run** the discrete-time PID controller with feedforward structure on **actual TCLab system**. Implement your code in a JupyterLab file named `TCLab_CLP_PID_FF.ipynb`. Note that all the experimental data should be written in a comma separated text file with header in the folder `Data`.

Note that it **does not make sense** to run your software on TCLab if it does not run in simulation !

12. **Optimise** your PID controller using the **IMC method** and compute the **stability margins**.
13. Bring an **element of originality** into your report distinguishing it from other reports. Your creativity is only limited by your imagination ! **Work as independently as possible from the supervisor**.
14. Incorporate and apply as many **theoretical concepts** discussed during the course as possible in your report !

2 Instructions and deliverables

Submit your work in the form of:

1. A **pdf report** which is meant as a means of **discussing** the **design choices** and **results** of the project and transferring the know-how that has been acquired to a colleague that has not been involved in the work. It is assumed that this colleague has some knowledge of control theory. **Theoretical reminders are therefore not necessary.**

The clarity, conciseness, structure and discussions of this document are **part of the evaluation**.

Your report should contain **at least** 7 experimental responses that are discussed in the text:

1. Step response from MV to PV
2. Step response from DV to PV
3. Closed-loop response to a setpoint change SP
4. Response to DV : No FF and controller in manual mode
5. Response to DV : FF and controller in manual mode
6. Response to DV : No FF and controller in automatic mode
7. Response to DV : FF and controller in automatic mode

Mathematical developments (e.g. discretisation of the lead-lag system) are **hand-written** and need to be added in **addendum** of the report as a **scanned document**.

2. A **Python package** named `package_LAB.py` and **JupyterLab files** named `Simulation_OLP.ipynb`, `package_LAB.ipynb`, `Simulation_CLP_PID_FF.ipynb` and `TCLab_CLP_PID_FF.ipynb`.
 - All functions should be included in the Python package `package_LAB.py` and then **imported** in the relevant JupyterLab files.

- All functions that are created in the Python package `package_LAB.py` should come with a help function !
- All functions that are created in the Python package `package_LAB.py` should be tested in the **JupyterLab** file `package_LAB.ipynb`.

The observance of these rules is **part of the evaluation**.

3. All **experimental** (open-loop and closed-loop) **data** should be stored in **comma separated text files with headers**. Rename the files in order to clarify the experimental conditions, i.e. OLP (open-loop), CLP (closed-loop), PID, FF, no FF, etc. For **closed-loop data**, at the very least, the following data should be stored:

`header='t,MV,MVP,MVI,MVD,SP,PV,DV,Man'.`

An experimental response in the report that comes without associated csv file will be discarded.

4. The report and models are to be submitted per email with attachments before 4pm, Friday March 31, 2023. **Delays** will incur a **penalty**, i.e.

- less than one day delay incurs a penalty of -2/20,
- in between one and two days delay incurs a penalty of -5/20,
- more than two days delay will result in a mark of 0/20.

Check that your software is **running** and that your “package” is **complete** before making your submission. **Resubmitting** your work incurs a **penalty** of -2/20.

5. The complete kit shown in Figure 6 must be returned intact at the end of the project. Damaged items will not be exchanged ! Respect and care for the equipment are part of the evaluation criteria.



Figure 6: Complete TCLab kit

6. The **use of a personal laptop** with a proper Anaconda environment installation is **mandatory** during the laboratory sessions. Refer to
<https://www.anaconda.com/products/individual>.
7. The attendance to the control laboratory is **compulsory**. The final score for the laboratory will be obtained by multiplying the score given to the final report by the ratio of the actual attendance time² over the intended attendance time. A **minimum penalty** of -0.5/20 is applied per delay.

²Being late for a laboratory session results in a smaller actual attendance time !

8. An unjustified absence will result in a 0/20 mark for the corresponding laboratory session. In case of a justified absence (medical certificate), contact the supervisor ASAP and a **substitution assignment** will be imposed if it is impossible to reschedule the laboratory session.
9. Assignment by teams of **two** students. **Submit your own work:** plagiarism³ will result in a mark of 0/20.

3 Classical model approximation methods

Different methodologies to determine the process parameters given a model structure exist. Refer to Figure 7 which represents the step response of the process. It can be used to “identify” the process dynamics. Once this response has been obtained, the parameters K_P , T_u , T_g , t_1 , t_2 and a can be obtained graphically as shown in Figure 7.

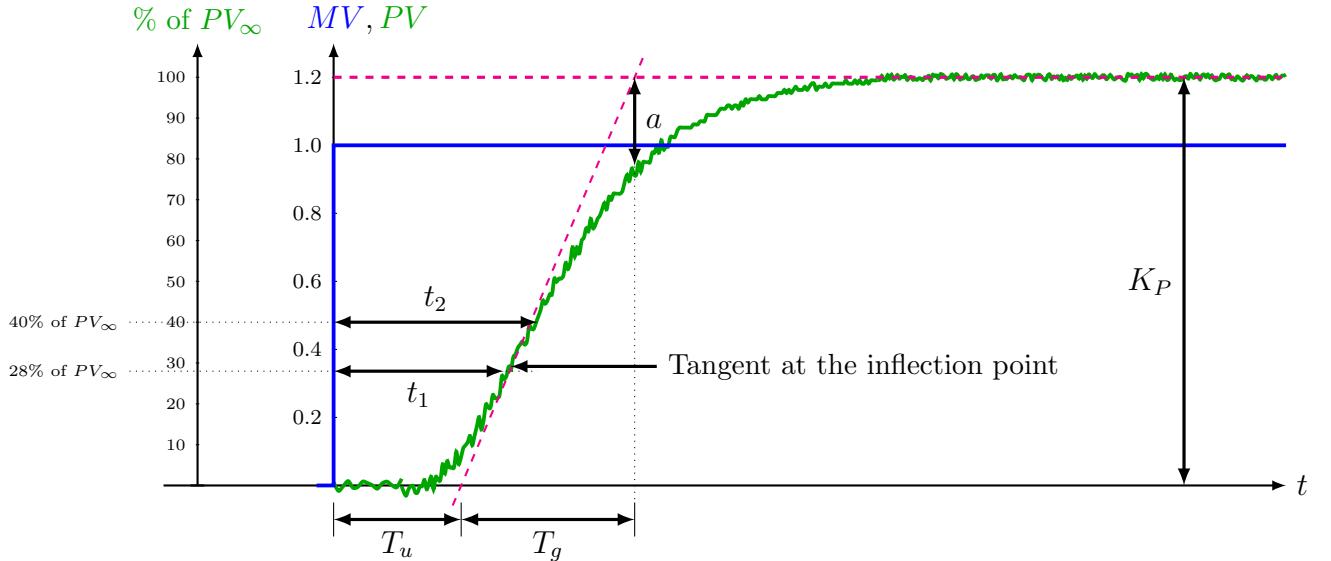


Figure 7: Experimental step response

3.1 Broida model

We can **approximate** the process using a **First Order model Plus Dead-Time** (FOPDT), also called **Broida model**, with transfer function

$$P_B(s) = \frac{K_P e^{-\theta s}}{(Ts + 1)}.$$

- A **simple method** for obtaining the Broida model process parameters is to simply set

$$T = T_g \text{ and } \theta = T_u$$

as outlined in the theoretical course.

- A slightly **more complicated method** is shown below. Use

$$T = 5.5(t_2 - t_1) \text{ and } \theta = 2.8t_1 - 1.8t_2$$

to obtain the time constant T and the process delay θ .

³Dishonest use of work done by others, including work in the public domain or work from fellow students. Source: <https://pdfs.semanticscholar.org/b70b/a0de83d78d17171cb5b427b3f377756a1f8e.pdf>

3.2 van der Grinten model

We can **approximate** the process using a **Second Order model Plus Dead-Time** (SOPDT), also called **van der Grinten model**, with transfer function

$$P_{vdG}(s) = \frac{K_P}{(T_1 s + 1)(T_2 s + 1)} e^{-\theta s}.$$

The model parameters are obtained using the formulae

$$\begin{aligned} T_1 &= T_g \frac{3ae - 1}{1 + ae} \\ T_2 &= T_g \frac{1 - ae}{1 + ae} \\ \theta &= T_u - \frac{T_1 T_2}{T_1 + 3T_2} \end{aligned}$$

where e is the base of the natural logarithm, i.e. the number whose natural logarithm is equal to one. It is approximately equal to 2.71828.

Note this method is well-suited for processes which have an “S-shape” step response.

3.3 Strejc model

We can **approximate** the process using a **n -th order model with identical poles**, also called **Strejc model**, with transfer function

$$P_S(s) = \frac{K_P e^{-\theta s}}{(Ts + 1)^n}.$$

Order n	$T_{u_{th}}/T_g$	T_g/T
	a_n	b_n
1	0.00	1.00
2	0.10	2.72
3	0.22	3.69
4	0.32	4.46
5	0.41	5.12
6	0.49	5.70
7	0.57	6.23

Table 1: Computation of Strejc model parameters

Refer to Table 1. The method works as follows:

1. Determine T_u and T_g
2. The ratio T_u/T_g determines the order of the system, i.e. n . Choose the value of n such that

$$a_n \leq \frac{T_u}{T_g} < a_{n+1}.$$

3. With n fixed, one has

$$b_n = \frac{T_g}{T} \implies T = \frac{T_g}{b_n}.$$

4. With n fixed, one has

$$a_n = \frac{T_{u_{th}}}{T_g} \implies T_{u_{th}} = a_n T_g.$$

Note that $T_{u_{th}}$ is the **apparent dead-time** that results from an n -th order system with time constant T . We see that the **actual observed dead-time** T_u is often larger. The difference must come from a **delay** θ in the system, i.e.

$$\theta = T_u - T_{u_{th}} = T_u - a_n T_g.$$