TeCoLab: Temperature Control Laboratory
User Manual

Contents

Chapter 1

Getting Started

1.1 What is TeCoLab?

TeCoLab, short for Temperature Control Laboratory, is an open-source tool comprising hardware, firmware and software. It is designed for experimenting with control algorithms in the context of temperature control. You can access all the project files on our GitHub repository. Before delving into the details, let's embark on a step-by-step tutorial to implement a discrete-time PI (Proportional-Integral) controller with period of 1 second. Our objective is to regulate the temperature of Heater 1 to 15 °C above ambient temperature.

1.2 What do I need to run TeCoLab?

Hardware:

- **TeCoLab hardware:** You can assemble your own TeCoLab using the files and information available on our GitHub repository;
- **Arduino Nano:** The microcontroller used in this project;
- **USB-B cable:** The standard communication cable for the Arduino;
- **12 V DC power supply:** A 12 V DC power supply with at least 2 A current capacity. Connect the power supply to jack J1 (with positive in the middle) or the bornier terminal J2.

Software:

- **TeCoLab files:** Download from our GitHub repository;
- **Python 3:** Used to execute the TeCoLab program;
- **Python 3 dependencies:** Refer to the list in Chapter 3;
- **Arduino software:** Required for loading the firmware to your Arduino (you will only need to perform this once);
- **Spreadsheet editor:** A spreadsheet editor, such as LibreOffice Calc, will be useful.

1.3 Downloading TeCoLab

To obtain TeCoLab, follow these steps:

1. Visit the TeCoLab GitHub repository, click on "Code" and select "Download ZIP";
2. Unzip the downloaded content;
3. Voilà! TeCoLab is now on your computer.

1.4 Programming Your Arduino

In the folder `TeCoLab/Firmware` you'll find the firmware project named *Firmware.ino*. Follow these steps to program your Arduino:

1. Open the *Firmware.ino* project using the Arduino software;
2. Load the firmware into your Arduino;

No changes are needed in the firmware. This procedure is a one-time procedure.

1.5 Important Files of TeCoLab

Three files are crucial in TeCoLab:

- **Experiment File:** A `.csv` file located in `TeCoLab/Software/Experiments`. In this file, you will define your experiment, including setpoints of temperature, desired disturbances, etc.
- **Control File:** A `.py` file located in `TeCoLab/Software/Controllers`. In this file, you will define the control algorithm you want to test during the experiment.
- **Log File:** A `.csv` file. After defining the experiment and control files, you can execute an experiment. TeCoLab will record the data, such as the temperature evolution, in a log file saved at `TeCoLab/Software/Logs`.

For more information about these files, refer to Chapter 4.

1.6 Creating Your Experiment

Let's create our first experiment, aiming to regulate the temperature of Heater 1 to 15 °C above the ambient temperature, as mentioned in Section 1.1.

1. Navigate to `TeCoLab/Software/Experiments`.
2. Copy the file *Template.csv* and rename it to *MyFirstExperiment.csv*.
3. Open the file. Although the header (first row) contains various fields, we will focus on only two: `TIME` (first column) and `SP1_REL` (fourth column).
4. Edit your file to resemble the example in Table 1.1. It will end with only three rows (besides the header). Do not fill in any other fields besides those specified in the example.

TIME	SP1_ABS	SP2_ABS	SP1_REL	SP2_REL	...
0			0		...
5000			15		...
900000			0		...

Table 1.1: Setting up your experiment file.

In the experiment file, the `TIME` column specifies the time (in ms) when the signals of the other columns will be applied to the experiment. The `SP1_REL` column specifies the relative temperature setpoint, *i.e.*, the temperature difference with respect to the ambient temperature (given in °C).

This proposed experiment file describes the following experiment: at time 0 (TIME = 0 ms), the relative setpoint is 0 °C (SP1_REL = 0 °C) for Heater 1, indicating no desired temperature change. At 5 seconds (TIME = 5000 ms), the relative setpoint for Heater 1 is set to 15 °C (SP1_REL = 15 °C). This value will be accessed by your control algorithm code to compute a control action for changing the temperature of Heater 1. The experiment concludes at 15 minutes (TIME = 900000 ms). Remember: for now, no changes are needed in any other columns not mentioned in this example.

For more information about the experiment file, refer to Section 4.1.

1.7 Creating Your Controller

Let's create our first controller to implement a discrete-time transfer function with a period of 1 second.

1. Navigate to `TeCoLab/Software/Controllers`, copy the file *Template.py*, and rename it to *MyFirstController.py*.

2. Edit this file as shown in the following example (be careful when copying and pasting the code from this manual to your text/code editor, as using spaces instead of tabs can lead to program errors in Python):

```
from Modules.Utils.discrete_time_LTI import Controller
import control

class Controller(Controller):
  def __init__(self):
    super().__init__()

  def control_setup(self):
    self.set_signal_period(5)
    tf = control.tf([1, -0.95], [1, -1], True)
    self.index = self.set_LTI(tf)

  def control_action(self):
    temp = self.temperature_heater_1 - self.temperature_ambient
    error = self.setpoint_rel_1 - temp
    self.actuator_heater_1 = self.LTI_compute(self.index, error)
```

The first line imports tools from the TeCoLab discrete-time LTI (Linear Time Invariant) module, providing easier transfer function usage and access to experiment variables such as

temperature sensor measurements and applied heater power. The second line imports the standard Python control library, which will be used to create our controller from the desired discrete-time transfer function. Moreover, in the initialization of our `Controller` class, we initialize the parent class, which sets the parents attributes to their default values.

The `control_setup()` method is created in this file and will be called once by the TeCoLab software at the beginning of your experiment to configure your controller. It sets the attribute `signal_period` to 5 (this attribute is defined by a TeCoLab module, see Chapter 5 for more details). This means that your control algorithm will compute a new control action every five iterations of the TeCoLab experiment. Since, by default, each experiment iteration occurs every 200 ms, the controller period is 5 x 200 ms = 1 second (see Chapter 6 for more details). We also create the variable `tf`, defined as the discrete-time transfer function

$$C(z) = \frac{z - 0.95}{z - 1},$$

by specifying the numerator and denominator of this transfer function.

The `controlAction()` method is created to implement the control action computation. We created a `temp` variable, representing the difference between Heater 1's temperature and the ambient temperature. The second line computes the error. Then we compute the control action using the created transfer function and the error as its input, applying it to Heater 1's related output power variable. TeCoLab will have access to attributes like `actuator_heater_1` and will apply them to the physical board. Note that the actuator values (that is, heater 1, heater 2 and fan output power) can range from 0 to 100, representing percentages of the maximum power (see Chapter 2 for power specifications). However, you need not worry about this in your controller algorithm, as the TeCoLab software handles the adjustment of these values before sending them to the physical board.

For more information about the control file, refer to Section 4.2.

1.8 Executing the Experiment

It's time to execute your experiment. Navigate to `TeCoLab/Software`. Open the terminal and type the following command:

```
>> python3 tecolab.py MyFirstExperiment MyFirstController
```

Now, you will have to wait a little bit since your experiment runs for 15 minutes. The blinking green LED will become fully on, indicating communication between your computer and TeCoLab. At some point, the red LED can start to blink, indicating that the heater elements are becoming warm. This is expected behavior. You will know that your experiment has finished when the green LED starts to blink again. If you experience any problem, check the FAQ in Chapter 7.

1.9 Checking the Log File

When your experiment is complete, TeCoLab will save a log file in `TeCoLab/Software/Logs`. This file has the extension `.csv` and its name will be the date and time of the experiment execution.

The log file includes all the columns from the experiment file, along with additional columns providing information about the experiment's execution. You can find the meanings of all columns in Section 4.3. Currently, three columns are crucial: `TIME`, `H1_TEMP` and `H1_C_PWM`. These represent the time elapsed in the experiment (in ms), the temperature of Heater 1 (in

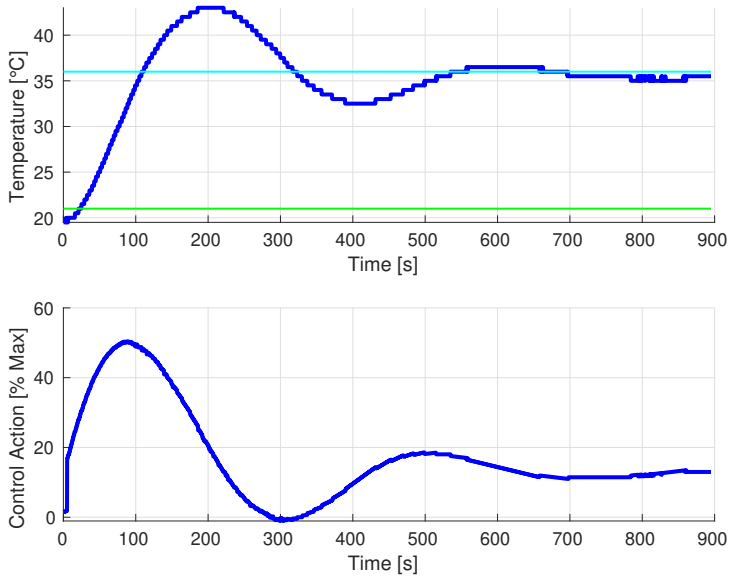

Figure 1.1: Data from the log file.

°C) and the computed power for Heater 1 (in % of $P_{\max}$), respectively. These three columns are graphically summarized in Figure 1.1. In the upper subfigure, the green line represents the ambient temperature, the cyan line is the setpoint (15 °C above ambient temperature) and the blue line is the temperature of Heater 1. In the lower subfigure, the blue line indicates the computed power for Heater 1.

1.10 What's next?

How can we enhance the results depicted in Figure 1.1? We could think about reducing the settling time (which was approximately 11 minutes in this experiment), eliminating overshoot (which was approximately 50% in this experiment) or minimizing the energy expended by the controller to reach the reference temperature. Additionally, we could explore the behavior of our controller in the presence of disturbances in the computed control action, or subject to a more restricted saturation range or a limitation on the rate of change of the control action. TeCoLab provides a platform for delving into these aspects and experimenting with various configurations.

Chapter 2

TeCoLab Specifications

TeCoLab is designed with specific hardware specifications to ensure optimal performance and compatibility. Below are the key specifications for TeCoLab:

Item	**Specification**
Recommended Power Supply	12 V DC, 2 A
Primary Power Supply Connector	J1 - DC Jack (positive in the middle)
Alternative Power Supply Connector	J2 - Bornier Terminal
Microcontroller	Arduino Nano
Communication Connector	USB-B
Maximum Power Consumption (@12 V)	8.4 W
Maximum Power per Individual Heater (@12 V)	1.8 W

Table 2.1: TeCoLab specifications.

Chapter 3

Python Dependencies

To ensure the proper functionality of TeCoLab, the following Python 3 packages are required, each serving a specific purpose:

- **importlib:** Loads the user's controller modules in execution time (see Chapter 5);
- **argparse:** Parses command-line arguments for program execution;
- **control:** Provides tools regarding control systems;
- **datetime:** Manipulates dates and times in Python;
- **numpy:** Offers support for data computation;
- **pandas:** Handles experiment and log data;
- **pathlib:** Manipulates file system paths to load/save files;
- **pyserial:** Provides serial tools used to locate if a TeCoLab board is connected;
- **serial:** Executes the communication with the TeCoLab board;
- **struct:** Handles binary data for the communication protocol;
- **time:** Manages time measurements.

Ensure that these dependencies are installed to guarantee the correct execution of TeCoLab. You can install them using your preferred package manager or the following command:

```
>> pip install name_of_the_package
```

Chapter 4

TeCoLab Files

TeCoLab operates with three distinct types of files, each playing a different role in the functionality of the system and analysis of the generated data. The relationships among these files and their connection to the hardware are detailed in Figure 4.1.

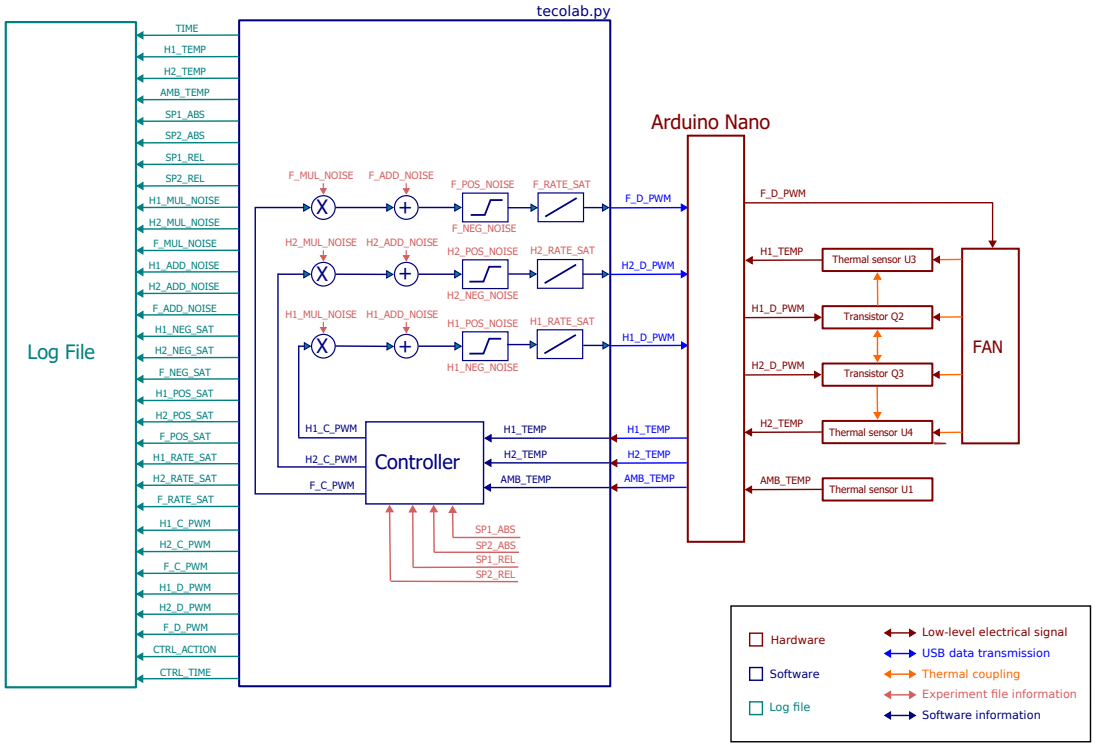

Figure 4.1: Block Diagram of TeCoLab Experiment

The three types of files in TeCoLab are:

1. **Experiment File (`.csv`):** Located in `TeCoLab/Software/Experiments`. This file, provided by the user, defines the experiment setup, which includes details such as temperature setpoints, disturbances, and other parameters necessary for the description of the experiment. In the Figure 4.1, the experiment file is responsible to provide the information in pink. Refer to Section 4.1 for a comprehensive guide on creating experiment files.

2. **Control File (`.py`):** Located in `TeCoLab/Software/Controllers`. This file, provided by the user, encapsulates the control algorithm for the experiment. Its purpose is to compute the control action for the two heaters and the fan based on the setpoints provided by the experiment file and the temperatures measurements from the board sensors. In the Figure 5.1 the control file is represented by the *Controller* block. Refer to Section 4.2 for a comprehensive guide on creating control files. Observe that TeCoLab provides auxiliary scripts to help on the creation of various classical control algorithms, which are described in Chapter 5.

3. **Log File (`.csv`):** located in `TeCoLab/Software/Logs`. This file, created by the TeCoLab, contains detailed information about the experiment's execution, including timestamps, temperatures, and control actions. It is named with the date and time of the experiment. Refer to Section 4.3 for a comprehensive guide on creating interpreting log files.

Understanding the roles and formats of these files is essential for effectively using TeCoLab. Consult the respective sections for each file type to ensure proper file creation and utilization.

4.1 The Experiment File

The Experiment File must be saved in `TeCoLab/Software/Experiments`. It is a CSV (Comma-Separated Values) which contains a detailed description of your experiment. The time indicated at the last row indicates the experiment's end in milliseconds (ms). For example, in the tutorial from Chapter 1, the final value is 900000, equivalent to 15 minutes (900000 ms).

The columns of the Experiment File are explained as follows, with bold variables accessible to your control algorithm (see Section 4.2 and Chapter 5 for details):

- TIME: The time when this experiment row is applied (in ms);
- **SP1_ABS:** Absolute setpoint of Heater 1 (in °C);
- **SP2_ABS:** Absolute setpoint of Heater 2 (in °C);
- **SP1_REL:** Relative setpoint of Heater 1 (in °C);
- **SP2_REL:** Relative setpoint of Heater 2 (in °C);
- H1_MUL_NOISE: Multiplicative noise for the control action of Heater 1 (dimensionless). The control action computed by the control algorithm is multiplied by this value. Default value is 1;
- H2_MUL_NOISE: Multiplicative noise for the control action of Heater 2 (dimensionless). The control action computed by the control algorithm is multiplied by this value. Default value is 1;
- F_MUL_NOISE: Multiplicative noise for the control action of the Fan (dimensionless). The control action computed by the control algorithm is multiplied by this value. Default value is 1;
- H1_ADD_NOISE: Additive noise for the control action of Heater 1 (in % of $P_{\max}$). This value is added to the control action computed by the control algorithm. Default value is 0;

- H2_ADD_NOISE: Additive noise for the control action of Heater 2 (in % of $P_{\max}$). This value is added to the control action computed by the control algorithm. Default value is 0;
- F_ADD_NOISE: Additive noise for the control action of the Fan (in % of $P_{\max}$). This value is added to the control action computed by the control algorithm. Default value is 0;
- H1_NEG_SAT: Negative saturation for the control action of Heater 1 (in % of $P_{\max}$). This represents the minimum possible value for the control action. Default value is 0;
- H2_NEG_SAT: Negative saturation for the control action of Heater 2 (in % of $P_{\max}$). This represents the minimum possible value for the control action. Default value is 0;
- F_NEG_SAT: Negative saturation for the control action of the Fan (in % of $P_{\max}$). This represents the minimum possible value for the control action. Default value is 0;
- H1_POS_SAT: Positive saturation for the control action of Heater 1 (in % of $P_{\max}$). This represents the maximum possible value for the control action. Default value is 100;
- H2_POS_SAT: Positive saturation for the control action of Heater 2 (in % of $P_{\max}$). This represents the maximum possible value for the control action. Default value is 100;
- F_POS_SAT: Positive saturation for the control action of the Fan (in % of $P_{\max}$). This represents the maximum possible value for the control action. Default value is 100;
- H1_RATE_SAT: Maximum rate of change for the control action of Heater 1 (in % of $P_{\max}$ per second). This represents the maximum possible rate of change for the control action. Default value is 1000;
- H2_RATE_SAT: Maximum rate of change for the control action of Heater 2 (in % of $P_{\max}$ per second). This represents the maximum possible rate of change for the control action. Default value is 1000;
- F_RATE_SAT: Maximum rate of change for the control action of the Fan (in % of $P_{\max}$ per second). This represents the maximum possible rate of change for the control action. Default value is 1000.

Note that there is no essential distinction among the parameters SP1_ABS, SP2_ABS, SP1_REL and SP2_REL. Your control algorithm can utilize the values specified in the experiment file for these fields according to your specific requirements.

4.2 The Control File

The control file in TeCoLab is a Python (`.py`) script in `TeCoLab/Software/Controllers`. Its primary purpose is to implement the control algorithm for your experiment. The control file interacts with the TeCoLab board to adjust the actuators (heaters and fan) based on the system's feedback.

Here are the steps to set up your control file from scratch (that is, without using the control scripts provided by TeCoLab, as explained in Chapter 5):

1. **Import the TeCoLab Default Controller Module**:

   Begin your Python controller script by importing the default controller class from TeCoLab.

```
from Modules.tecolab_controller import Controller
```

2. **Create a Class `Controller` that Inherits from `Controller`**:

   Define a class named `Controller` in your Python script, inheriting from the default `Controller` class. This new class will be the blueprint for your custom control algorithm.

```
class Controller(Controller):
  def __init__(self):
    super().__init__()
    # Create here your controller class attributes.
```

   Thanks to the inheritance, your `Controller` class will have access to the following attributes:

   - `setpoint_abs_1`: absolute setpoint temperature (in °C) for heater 1, obtained from the experiment file;
   - `setpoint_abs_2`: absolute setpoint temperature (in °C) for heater 2, obtained from the experiment file;
   - `setpoint_rel_1`: relative setpoint temperature (in °C) for heater 1, obtained from the experiment file;
   - `setpoint_rel_2`: relative setpoint temperature (in °C) for heater 2, obtained from the experiment file;
   - `temperature_heater_1`: measured temperature (in °C) for heater 1, obtained from the board sensors;
   - `temperature_heater_2`: measured temperature (in °C) for heater 2, obtained from the board sensors;
   - `temperature_ambient`: measured ambient temperature (in °C), obtained from the board sensors;
   - `actuator_heater_1`: control action for heater 1 (in % of $P_{\max}$), which will be applied to the board;
   - `actuator_heater_2`: control action for heater 2 (in % of $P_{\max}$), which will be applied to the board;
   - `actuator_fan`: control action for the fan (in % of $P_{\max}$), which will be applied to the board;

   If your control class uses a setpoint attribute whose value is not specified in the experiment file, an error will occur. For instance, if you utilize the attribute `self.setpoint_abs_2` in your control algorithm without specifying values in the column `SP2_ABS` of the experiment file, an error will be triggered.

3. **Implement the `control_setup()` Method**:

   The `control_setup()` method is executed once by TeCoLab at the start of your experiment. It's where you configure your controller, creating and setting up any specific attributes needed for your control algorithm.

   - **Parameters:** None;
   - **Returns:** None.
   - **Example:**

```
def control_setup(self):
    # Your configuration code here
```

4. **Implement the `control_action()` Method**:

   The `control_action()` method contains the computation of your control action. It must assign the calculated control actions to the following standard attributes from TeCoLab base controller class:

   - `self.actuator_heater_1`
   - `self.actuator_heater_2`
   - `self.actuator_fan`

   If your control algorithm do not use some actuator (for example, the fan), no value need to be assigned for that attribute;

   - **Parameters:** None;
   - **Returns:** None.
   - **Example:**

   ```
   def control_action(self):
       # Your control action computation code here
   ```

5. **Implement the `control_signal()` Method**:

   The `control_signal()` method returns a boolean value. True means that a new control action must be computed, that is, the method `control_action()` will be called in the current moment of the TeCoLab experiment. False means that a new control action will not be computed at this moment and, therefore, the last computed control action will be applied;

   - **Parameters:** None;
   - **Returns:** Boolean;
   - **Example:**

   ```
   def control_signal(self):
       # Your control signal decision code here
       return True  # or False based on your logic
   ```

Note: You don't have to build a controller file from scratch every time. TeCoLab provides tools that you can use. Refer to Chapter 5 for more information.

4.3 The Log File

The log file is a CSV (comma-separated values) file stored in the `TeCoLab/Software/Logs` folder. Its purpose is to register the results of your experiment. Each iteration of your experiment generates a new row in this file, with a default time interval of 200 milliseconds (configurable, see Chapter 6). The columns in the log file mirror those in the experiment file, with the following additional columns:

- H1_TEMP: Current temperature of Heater 1 (in °C).

- H2_TEMP: Current temperature of Heater 2 (in °C).
- AMB_TEMP: Current ambient temperature (in °C).
- H1_C_PWM: Computed value of the control action for Heater 1 (in % of $P_{\max}$).
- H2_C_PWM: Computed value of the control action for Heater 2 (in % of $P_{\max}$).
- F_C_PWM: Computed value of the control action for the fan (in % of $P_{\max}$).
- H1_D_PWM: Disturbed value of the control action for Heater 1, accounting for multiplicative noise, additive noise, negative and positive saturation, and rate of change limitation (in % of $P_{\max}$).
- H2_D_PWM: Disturbed value of the control action for Heater 2, accounting for multiplicative noise, additive noise, negative and positive saturation, and rate of change limitation (in % of $P_{\max}$).
- F_D_PWM: Disturbed value of the control action for the fan, accounting for multiplicative noise, additive noise, negative and positive saturation, and rate of change limitation (in % of $P_{\max}$).
- CTRL_ACTION: A flag where the value 1 indicates whether a new control action was computed at that instant. If the value is 0, a new control action was not computed and the last computed control action was applied. This is equivalent to the return value of the `control_signal()` method;
- CTRL_TIME: The time (in ms) taken to compute your control algorithm. This value can be useful for comparing the performance of different control algorithms.

Chapter 5

TeCoLab Control Tools

TeCoLab provides Python scripts, referred to as *modules*, designed to simplify the implementation of various classic control algorithms. These modules progressively introduce additional attributes, methods and functionalities to the `Controller` class in the structured manner displayed in Figure 5.1.

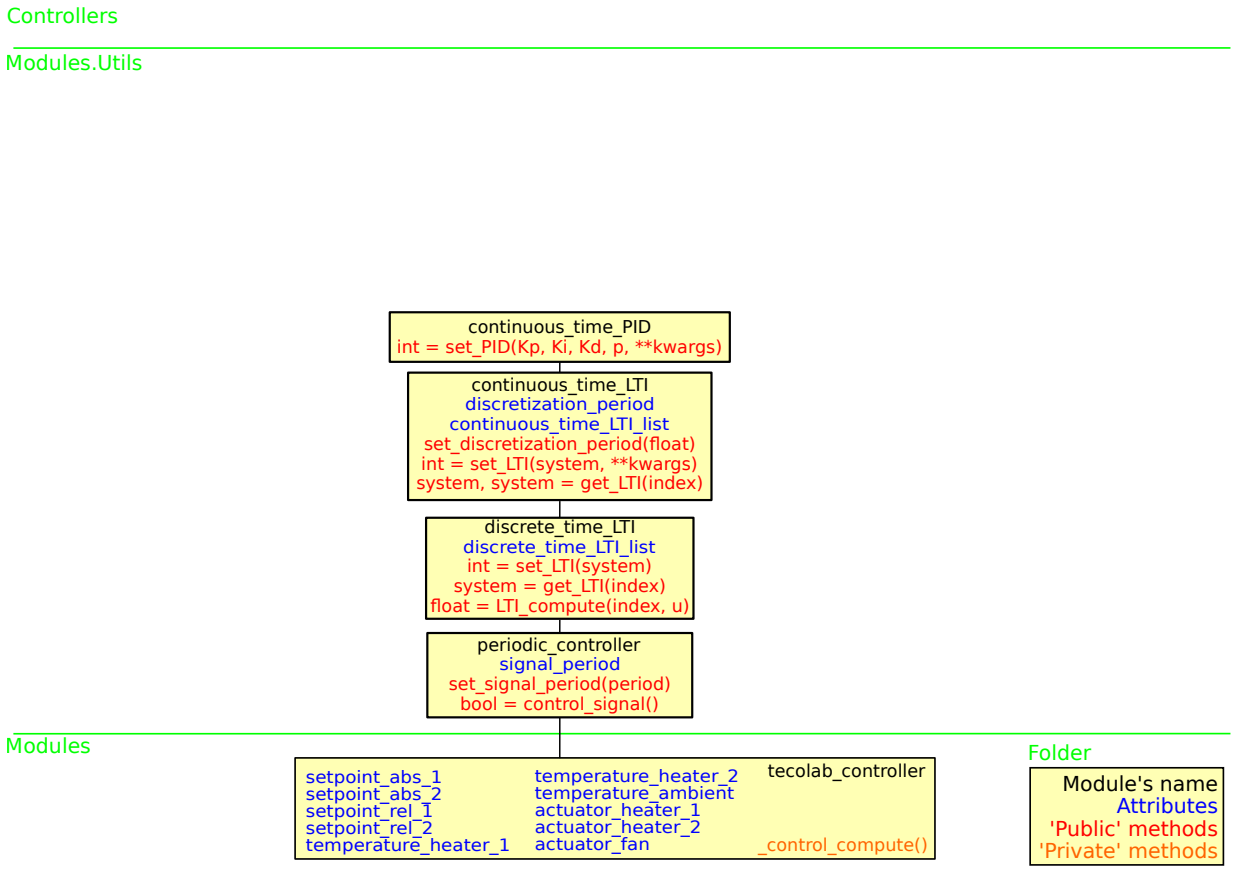

Figure 5.1: Structure of the Control Modules.

The structure of Figure 5.1 is divided in three layers, each one located in a different folder. The deepest layer can be found in `TeCoLab/Software/Modules` (the bottom one in Figure 5.1) and defines the base TeCoLab `Controller` class. Through its attributes, this class provides access to the values received from the TeCoLab board or that will be send to it. The middle layer is located in `TeCoLab/Sofware/Modules/Utils`. It provides tools to help you in the implementation of classical control algorithms, such as transfer functions. Finally, the last layer are the user's custom control scripts, which must be saved in `TeCoLab/Software/Controllers`

and can use any module from the two layers below. Observe that you don't need to use the modules from the middle layer to build your control algorithms. What is important is that your control algorithm, through the use of modules or not, define three methods: `control_signal()`, `control_setup()` and `control_action()` (see Section 4.2 for details about these methods).

In the script file of each module, the code structure shown below is followed. The concept is that every script creates a class named `Controller`, inheriting from the imported `Controller` class from the previous module. It's crucial for each subsequent script, including your custom ones, to create a class with the identical name (`Controller`).

```
from the_previous_module import Controller

class Controller(Controller):
    # Your control algorithm is coded inside this class.
```

5.1 `tecolab_controller` Module

The `tecolab_controller` module serves as the fundamental controller class in TeCoLab. Other classes build upon this base to provide additional attributes, methods and functionalities. This module defines the class `Controller`, which has attributes that receive values from the experiment table during the execution of an experiment, receive values from the sensors of the TeCoLab board or set values as control actions to the heaters and the fan. Next, the specifications of the class `Controller` defined by this module are given.

- Parameters: None
- Attributes:
  - `setpoint_abs_1` (float): Absolute setpoint temperature (in °C) for heater 1, obtained from the experiment file.
  - `setpoint_abs_2` (float): Absolute setpoint temperature (in °C) for heater 2, obtained from the experiment file.
  - `setpoint_rel_1` (float): Relative setpoint temperature (in °C) for heater 1, obtained from the experiment file.
  - `setpoint_rel_2` (float): Relative setpoint temperature (in °C) for heater 2, obtained from the experiment file.
  - `temperature_heater_1` (float): Measured temperature (in °C) for heater 1, obtained from the sensors.
  - `temperature_heater_2` (float): Measured temperature (in °C) for heater 2, obtained from the sensors.
  - `temperature_ambient` (float): Measured ambiente temperature (in °C), obtained from the sensors.
  - `actuator_heater_1` (float): Control action for heater 1 (in % of $P_{\max}$), which will be applied by the TeCoLab software to the board.
  - `actuator_heater_2` (float): Control action for heater 2 (in % of $P_{\max}$), which will be applied by the TeCoLab software to the board.
  - `actuator_fan`: Control action for the fan (in % of $P_{\max}$), which will be applied by the TeCoLab software to the board.

- Methods:

  `_control_compute(setPoints, temperatures)`:

  Used by the TeCoLab to call other methods of the class during the experiment to compute the control actions. As a user, never use this method in your code.

To incorporate this module into your Python script, use the following line of code:

```
from Modules.tecolab_controller import Controller
```

5.2 `periodic_controller` Module

The `periodic_controller` module builds upon the base controller class to facilitate the creation of periodic controllers within TeCoLab. These controllers compute new control actions at regular time intervals, defined by the period attribute `signal_period`, which is added by this module. The period is given in multiples of the base period of a TeCoLab experiment, which is by default 200 ms (configurable, see Chapter 6). For instance, setting `signal_period` to 5 results in a control action computed every 5 x 200 ms = 1 s. Next, the specifications of the class `Controller` defined by this module are given.

- Parameters: None
- Attributes:
  - `signal_period` (int, default value 1): The period for computing control actions, specified in multiples of the base TeCoLab period.
- Methods:

  `set_signal_period(period)`:

  Set the `signal_period` attribute. It checks if the argument is a number, throwing an error if it is not. Moreover, if the argument is not an integer, it is rounded to the closest integer.

  - Parameters:
    - period (integer): The value to be assigned to the `signal_period` attribute.
  - Returns: None

  `control_signal()`:

  Computes the periodic control signal to determine whether a new control action should be computed. See Section 4.2 for details about this method.

To incorporate this module into your Python script, use the following line of code:

```
from Modules.Utils.periodic_controller import Controller
```

5.3 `discrete_time_LTI` Module

The `discrete_time_LTI` module provides the base class for creating controllers based on discrete-time Linear Time Invariant (LTI) systems, such as transfer functions or state space representations. It allows you to create LTI objects using the standard Python control library and store them within the `Controller` class. Then, you can compute the output of the stored LTI systems based on a given input. Next, the specifications of the class `Controller` defined by this module are given.

- Parameters: None
- Attributes:
  - `discrete_time_LTI_list` (list, default value *None*): The list of internally saved discrete-time LTI systems.
- Methods:

  `set_LTI(system)`:

  Store a discrete-time LTI system, which can be created using the TransferFunction or the StateSpace functions from Python standard control library. The method returns an integer representing the index of the stored LTI system, which is used to identify one specific system within the list of stored systems.

  - Parameters:
    - system (LTI): LTI object that can be created using the TransferFunction or the StateSpace functions from Python standard control library.
  - Returns:
    - int: The index that identifies the stored LTI system.

  `get_LTI(index)`:

  Get a previously stored discrete-time LTI system based on its index. The method returns the LTI system used for output computation in the state-space representation. Note that this representation is usually different from the original system stored (although equivalent).

  - Parameters:
    - index (int): The index that identifies the stored LTI system.
  - Returns:
    - system (LTI): state space representation of the stored LTI object. This is the representation that the `Controller` class uses for output computation.

  `LTI_compute(index, u)`:

  Computes the output of a previously created and stored LTI system.

  - Parameters:
    - index (int): The index that identifies the stored LTI system.
    - u (float): The input of the LTI system.

- Returns:
  - Float: The computed output of the LTI system.

To incorporate this module into your Python script, use the following line of code:

```
from Modules.Utils.discrete_time_LTI import Controller
```

5.4 `continuous_time_LTI` Module

The `continuous_time_LTI` module provides the base class for creating controllers based on continuous-time LTI systems within TeCoLab. It allows you to create transfer functions or state space objects which are internally stored and then compute their output based on a given input. Note that, since TeCoLab is a digital plataform, the continuous-time LTI objects are later converted to discrete-time LTI objects in order to compute outputs. In order to have the correct behavior, your custom control algorithm should not change the `signal_period` attribute, which will be automatically set to 1. Next, the specifications of the class `Controller` defined by this module are given.

- Parameters: None
- Attributes:
  - `discretization_period` (float, default value 0.2): Period used in the discretization of the continuous-time LTI systems. To achieve the correct performance, this value must be equal to the TeCoLab iteration period (see Chapter 6 for more information).
  - `continuous_time_LTI_list` (list, default value *None*): The list of internally saved continuous-time LTI systems.
- Methods:

  `set_discretization_period(period)`:

  Set the discretization period used to convert the continuous-time LTI objects into discrete-time LTI objects. To achieve the correct performance, this value must be equal to the TeCoLab iteration period (see Chapter 6 for more information).

  - Parameters:
    - period (float): Period used in the discretization process.
  - Returns: None.

  `set_LTI(system, **kwargs)`:

  Store a continuous-time LTI system, which can be created using the TransferFunction or the StateSpace functions from Python standard control library. The method discretizes the continuous-time LTI system, stores it internally and returns an integer representing the index of the stored LTI system, which is used to identify one specific system within the list of stored systems.

  - Parameters:
    - system (LTI): continuous-time LTI object that can be created using the TransferFunction or the StateSpace functions from Python standard control library.

    - ∗ **kwargs: The additional arguments of the `control.sample_system()` function from the standard Python control library.
  - Returns:
    - ∗ int: The index that identifies the stored LTI system.

`get_LTI(index)`:

Get a previously stored continuous-time LTI system and its discretized version based on its index. The method returns the two LTI systems.

- Parameters:
  - ∗ index (int): The index that identifies the stored LTI system.
- Returns:
  - ∗ system (LTI): The original continuous-time LTI object stored.
  - ∗ system (LTI): The discretized LTI object. This is the representation used for computing the output of LTI objects.

To incorporate this module into your Python script, use the following line of code:

```
from Modules.Utils.continuous_time_LTI import Controller
```

5.5 `continuous_time_PID` Module

The `continuous_time_PID` module provides the base class for creating continuous-time PID controllers. Next, the specifications of the class `Controller` defined by this module are given.

- Parameters: None
- Attributes: None
- Methods:

  `set_PID(Kp, Ki, Kd, p, **kwargs)`:

  Creates a continuous-time PID transfer function and internally saves it. The PID format is given by

$$C(s) = K_p \left(1 + \frac{K_i}{s} + sK_d \frac{p}{s+p}\right).$$

  - Parameters:
    - ∗ Kp (float): Proportional gain.
    - ∗ Ki (float): Integral gain.
    - ∗ Kd (float): Derivative gain.
    - ∗ p (float, default value 100): Pole for the low-pass filter used to make the derivative part causal.
    - ∗ **kwargs: The additional arguments of the `control.sample_system()` function from the Python control module.
  - Returns:
    - ∗ int: The index that identifies the stored LTI system.

To incorporate this module into your Python script, use the following line of code:

```
from Modules.Utils.continuous_time_PID import Controller
```

Chapter 6

TeCoLab Advanced Configurations

TeCoLab provides advanced configurations that can be specified as parameters when running the program from the terminal. These advanced configurations provide flexibility in adapting TeCoLab to specific requirements and computational constraints.

6.1 Help

To access the help menu for TeCoLab, use the following command:

```
>> python3 tecolab.py -h
```

This command displays information about available options and how to use them.

6.2 TeCoLab Experiment Period

You can adjust the base period of the TeCoLab experiment using the following command:

```
>> python3 tecolab.py ExperimentFile ControlleFile -t XXX
```

Replace XXX with the desired period in milliseconds. For example, setting XXX to 500 means that an experiment iteration will occur every 500 ms, instead of the default 200 ms. Adjusting this period can be beneficial if your control algorithm's computation takes more than the default 200 ms. However, be cautious, as changing this value will also affect the period of control algorithms.

Chapter 7

FAQ - Frequently Asked Questions

7.1 Why is the green LED blinking?

The green LED blinks to indicate that the TeCoLab is awaiting the execution of a new experiment.

7.2 Why is the green LED on?

The green LED on indicates that the TeCoLab is currently executing an experiment.

7.3 Why is the red LED blinking?

The red LED blinks to indicate that the TeCoLab's heater elements are warm within a safe temperature range. Please avoid touching the heater elements when the red LED is blinking.

7.4 Why is the red LED on?

The red LED on indicates that the TeCoLab's heater elements are dangerously hot. In such a situation, TeCoLab will halt the current experiment and activate the fan to cool down the heater elements. It is crucial not to touch the heater elements when the red LED is on.

7.5 Why the fan turned on without a command?

See Section 7.4.

7.6 Why the heaters are not heating?

Check if you executed an experiment without connecting the 12 V DC power supply. In such case, the experiment will run normally, but without the power supply, the heaters and the fan will not activate.

7.7 Why the time in log files are not equally spaced?

Ideally, the time values in the log file should be evenly spaced, with a default spacing of 200 ms (see Section 6.2). However, some variation (around 10 ms) is not uncommon.

If the variation is excessive, potential causes include:

- Your computer is overloaded: Try closing other applications or restarting your computer before running a TeCoLab experiment.
- Your control algorithm takes too long: Check the control algorithm computation time in the log file (see Section 4.3). If the time is excessive, consider optimizing your algorithm or adjusting the experiment period (see Chapter 6). Note: changing the experiment period may also impact your algorithm's execution period.

7.8 Why a log file was not saved for my experiment?

An experiment must run for more than 5 seconds to save a log file.

7.9 Why don't all the temperatures match at the beginning of the experiment?

A slight temperature difference (around 1 °C) between the initial heater temperatures and the ambient temperature is common due to sensor variations. If the difference is greater, ensure you are not running consecutive experiments. After one experiment, wait for sufficient time for the heater elements to cool down before starting another one.

7.10 Why the temperature experienced a non specified disturbance?

Note that TeCoLab is a physical system, which means that external disturbances, not specified in the experiment file, might affect its sensors. For example, if a gust of wind blows on TeCoLab's temperature sensors, it can introduce unexpected disturbances. To prevent such scenarios, it is recommended to run TeCoLab experiments in an environment where the system will not be subject to external disturbances.