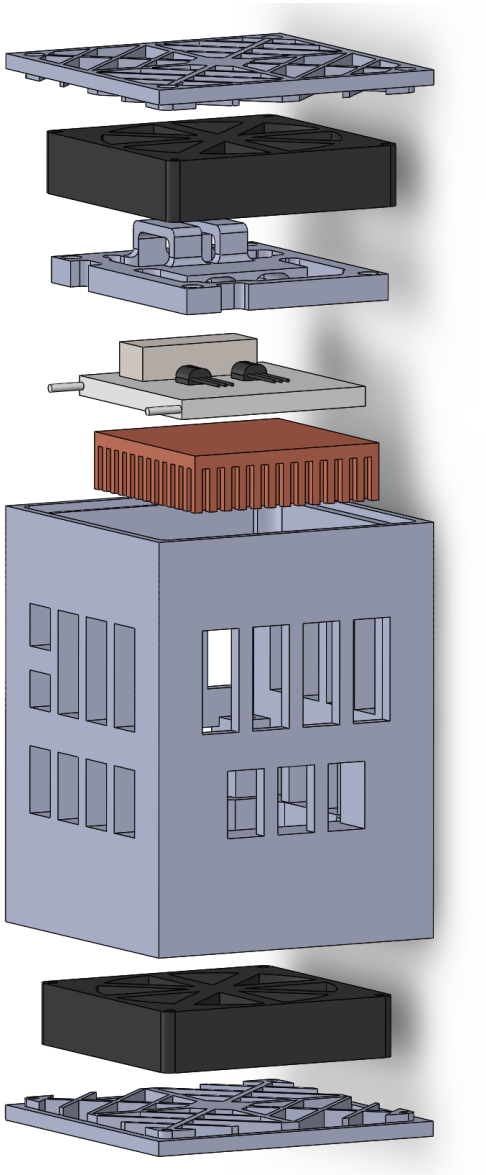


PELTIER CELL TAKE-HOME KIT
USER MANUAL

Version 1.0

June 14, 2025

Contents

1 Introduction

This document provides the instructions for properly setting up and operating the Peltier cell-based take-home kit for control education. It includes the steps required for hardware connection, software communication, and basic usage through MATLAB/Simulink or manual serial commands.
The interface circuit can work in two different modes; first it can act as interface, so the command for the actuator must be sent from a software running on a personal computer, and in this scenario the control loop is closed from the software itself; this is the most common mode of operation.
The second mode of operation is dedicated to experienced users and allows you to operate the circuit directly as a standalone controller. The user can program the controller in the C++ language by closing the control loop directly using the microcontroller. Variables such as u and y can be displayed on a serial monitor since a data stream is available.

1.1 Important Safety Notice

Never connect the interface to the PC via the USB cable while powered by the 12V power supply. In case of circuit failure, you risk to damage your own PC! If you need to connect it through USB, ensure that the circuit is not powered.

2 Bluetooth Connection

To establish a wireless connection between the take-home kit and a PC, a Bluetooth module (HC-05) is integrated into the interface circuit.
Follow the steps below to connect the device:

1. On your laptop go to *Settings*;
2. next click on *Bluetooth* & *devices*;
3. next click on *Add a new device* $\rightarrow$ *Bluetooth* $\rightarrow$ *Show all devices*;
4. click on the device named *Peltier cell BT interface*;
5. digit the password **1234**

The connection is established, now we have to find the *COM PORT* assigned to the device, to do that we have to follow this:

1. On your laptop go to *Settings*;
2. next click on *Bluetooth* & *devices*;
3. next click on *Show other devices*;
4. next click on *Other Bluetooth settings*;
5. in the window that opens click on the section named *COM PORTS*
6. here you have to find the device named *Peltier cell BT interface 'SPP Dev'*, the port COM assigned to this device has to be used for communicating with the device.

In Figure 1 an example, in this case the port number is *COM18*

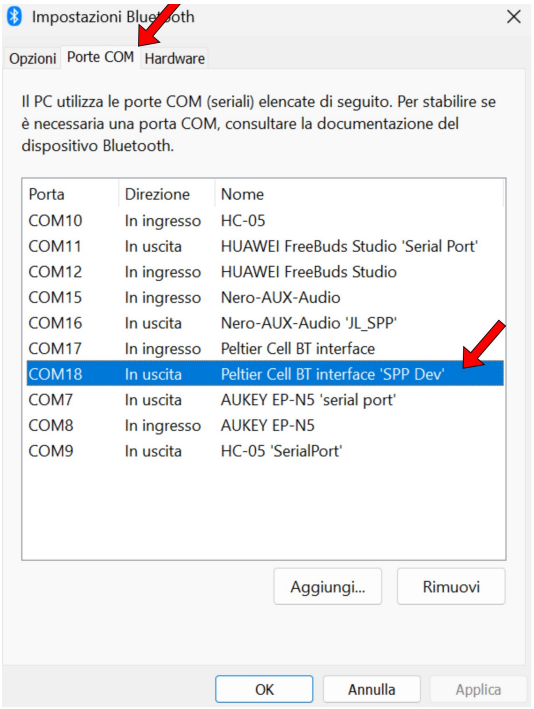


Figure 1: Example Bluetooth settings

3 Interface mode MATLAB/Simulink

The interface mode consists in sending commands to the circuit from software running on a personal computer. In order to establish a connection with MATLAB, you have to import into your workspace the folder named *MATLAB_PELTIER_CELL*, within the folder are provided all files necessary to communicate with the interface.
By default, at the startup the interface will be activated in this mode.

3.1 MATLAB script

Within the folder are provided two scripts:

- *start_communication.m*
- *step_test_u.m*

start_communication.m handles the serial communication. At line *14* you have to change the COM port number with the number assigned to the device during the first Bluetooth connection described in the previous section. This script should be run whenever you want to establish communication with the device.
The script *step_test_u.m* performs a step test on *u* automatically. In lines *14* and *15* you can change the step test parameters (amplitude and duration). When you run this script, you do not need to open the communication beforehand, since the script calls *start_communication.m* itself. Running this script you will see a plot that will update real-time with the temperature value read from the cell.

3.2 MATLAB function

The following functions are available to the user to interact with the cell. This functions can be called manually using the *Command Windows* or can be used to build a script that automatically performs actions in the cell, such as step tests.
In Table 1, the available functions are reported.

INPUT PARAMETERS:

- S: type of reading [1 - digital; 2 - analog]
- X: Peltier action [-100%;+100%]
- Y: Resistor action [0%;+100%]
- Z: Fan state [0 - OFF; 1 - ON]

OUTPUT PARAMETERS:

- t1: digital reading
- t2: analog reading

`t* = readTemp(S)`
`[t1, t2] = setActuatorsAndReadTemperatureSIM([X, Y, Z])`
`t1 = setPeltierActionAndReadTemperature(X)`
`t1 = setActuatorsAndReadTemperature(X, Y, Z)`
`setActuators(X, Y, Z)`
`setPeltierAction(X)`
`setResistor(Y)`
`setFan(Z)`

Table 1: MATLAB functions available

Below is a brief description:

Read temperature sensors

The function *readTemp(sensor)* returns the temperature reading, the argument passed to the function indicates the type of the temperature reading (analog or digital).

Set Peltier power action

The function *setPeltierAction(X)* allows setting the Peltier control action, the argument passed to the function must be in the range [-100%;+100%]

Set Peltier power action and read temperature

The function *setPeltierActionAndReadTemperature(X)* is a combination of the first two. It allows setting the Peltier action and reading the temperature in a single shot. By default, the sensor used is the digital one.

Set actuators and read temperature

The function *setActuatorsAndReadTemperature* allows to control all the actuators and read the temperature.

3.3 Simulink scheme

Two Simulink files are available; the first is ***open_loop_process.slx***. Using this scheme, you can test the actuators in open-loop mode, as shown in Figure 2. You can use the green block to implement your own control structure.
In the second file named ***PID_test.slx*** you can see an example of PID implementation.
Before run the Simulink scheme you have to open the serial connection running the MATLAB script ***start_communication.m***

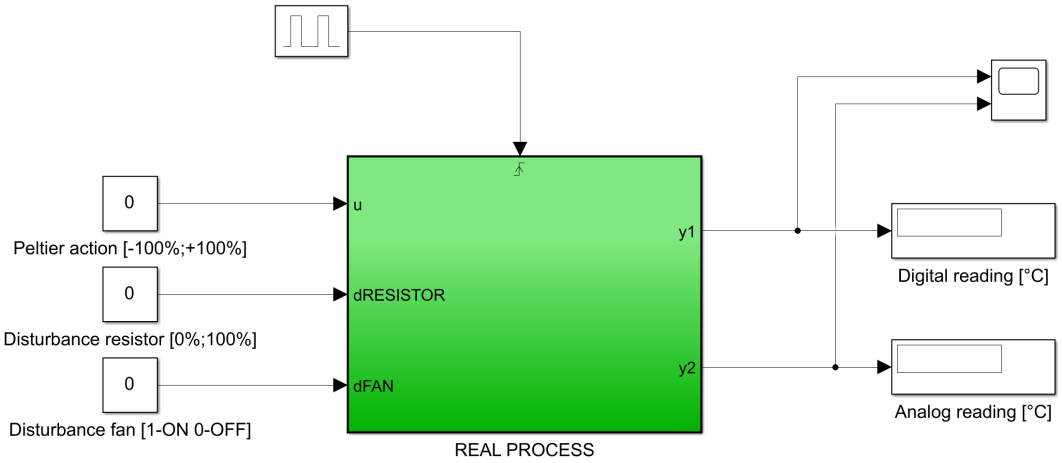


Figure 2: Open loop process Simulink

4 Standalone controller mode

In this mode, the controller has to be implemented directly inside the firmware. In the firmware, which is written in C++, a dedicated section has been set up, in order to allow the user to easily implement a digital controller. Some variables have been prepared, they allow the user to act directly on the actuators. By assigning a value to the variables, the firmware will pilot the actuators properly.

4.1 Serial communication parameters

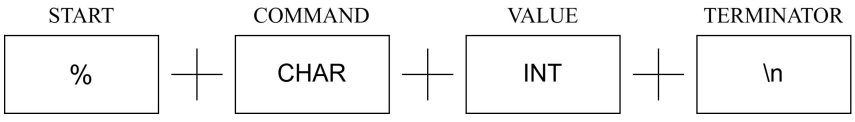
The parameters of serial communication are as follows:

- Baud Rate: 115200 bps;
- Data Bits: 8 bit;
- Parity: None;
- Stop Bits: 1 bit;
- Flow Control: None.

4.2 Communication protocol

In order to exchange information with the interface at *low level* we have to follow the communication protocol implemented in the firmware.

The structure of the string to be sent is as follows:



the char *"%"* is the start of the string, next we have another char that indicates the command to be executed and then the value that has to be sent.

The following are the commands implemented in the communication protocol:

Switch to *interface mode*:

- String to be sent: %M
- String returned: *Interface mode - Waiting for actuator commands*

The interface mode is set as default when the circuit is powered on

Switch to *standalone controller mode*:

- String to be sent: %A
- String returned: *Standalone controller mode - Waiting for start command*

Set setpoint for standalone controller:

- String to be sent: %s##
- Value ##: setpoint FLOAT [15°C;40°C]
- String returned: setpoint just set [°C]

Activate the standalone controller:

- String to be sent: %T
- String returned: *Standalone controller started*

Stop the standalone controller:

- String to be sent: %S
- String returned: *Standalone controller stopped*

Start data stream for serial plotter:

- String to be sent: %K
- String returned: *setpoint, y, u*

Stop data stream for serial plotter:

- String to be sent: %H
- String returned: *none*

Digital temperature reading:

- String to be sent: %b
- String returned: *float value* [°C]

Analog temperature reading:

- String to be sent: %a
- String returned: *float value* [°C]

Set power Peltier action:

- String to be sent: %p##
- Value ##: power Peltier action INT [-100%;+100%]
- String returned: power Peltier action just set [%]

Set power resistor action (load disturbance):

- String to be sent: %r##
- Value ##: power resistor action INT [0%;+100%]
- String returned: power resistor action just set [%]

Set fan (load disturbance):

- String to be sent: %f#
- Value #: INT 0 - OFF 1 - ON
- String returned: actual state just sent

Set all actuators and request analog temperature reading:

- String to be sent: %c##r##f#a
- Value 1 ##: power Peltier action INT [-100%;+100%]
- Value 2 ##: power resistor action INT [0%;+100%]
- Value 3 #: INT 0 - OFF 1 - ON
- String returned: *float value* [°C]

Set all actuators and request digital temperature reading:

- String to be sent: %c##r##f##b
- Value 1 ##: power Peltier action INT [-100%;+100%]
- Value 2 ##: power resistor action INT [0%;+100%]
- Value 3 #: INT 0 - OFF 1 - ON
- String returned: *float value* [°C]

Display overall variables - *for debug*

- String to be sent: %d
- String returned:
  - Mode: *interface* or *standalone controller*
  - Setpoint: *float value* [°C] (for standalone controller mode)
  - Peltier power action: *int value* [%] (for interface mode)
  - Resistor power action: *int value* [%] (for interface mode)
  - Fan state: *OFF* or *ON* [%] (for interface mode)

This string is designed to be displayed on a serial monitor, such as the one available within Arduino IDE.

In Table 2, all the commands summarized:

String	Description
%A	Set standalone controller mode
%M	Set interface mode
%T	Start standalone controller
%S	Stop standalone controller
%s##	Set setpoint standalone controller
%b	Request digital temperature
%a	Request analog temperature
%p##	Set Peltier action [-100%; +100%]
%r##	Set resistor action [0%; 100%] (load disturbance)
%f#	Set fan state [0-OFF, 1-ON] (load disturbance)
%c##r##f#a	Set Peltier; resistor; fan state + request temperature
%d	Display the overall variables for debug
%K	Enable stream for standalone controller
%H	Disable stream for standalone controller

Table 2: Serial commands

In order to interact with the interface, you can use the serial monitor built in *Arduino IDE* or any serial monitor that you prefer.

Note that the strings are case-sensitive

4.3 C++ controller programming

Using *Arduino IDE* open the file named ***Firmware_Vx.x.ino***. Inside the program, two sections are available for the user. The first is on the top of the program, where the user can declare the variables that he/she needs to use to implement the controller, while the second one is at the end of the program, here the user can implement the controller in the function named ***stand_alone_controller()***.
The variables available are the following:

- setPoint $\rightarrow$ the setpoint set *read only*;
- dt $\rightarrow$ iteration time *read only*;
- pwmRES $\rightarrow$ value power resistor action *read only*;
- fan $\rightarrow$ state fan action *read only*;
- $y_1 \rightarrow$ digital temperature reading *read only*;
- $y_2 \rightarrow$ analog temperature reading *read only*;
- $u \rightarrow$ Peltier power action *write allowed*.

The variables *read only* are updated automatically from the firmware, while the user can act on u. With the code structured in this way, the user does not need to know how to manage the actuators and the sensors, as will the firmware.

Here is an extract of the firmware in which the controller code have to be written:

```
/* THIS FUNCTION IS DEDICATED FOR USER THAT WANTS TO IMPLEMENT
   A STANDALONE CONTROLLER
 There are available this variables:

 setPoint -> setpoint setted [Celsius] READ ONLY
 dt       -> time between iterations [ms] READ ONLY
 pwmRES   -> resistor load disturbance action [%] READ ONLY
 fan      -> state fan load disturbance [0-1] READ ONLY
 y1       -> digital temperature reading [Celsius] READ ONLY
 y2       -> analog temperature reading [Celsius] READ ONLY
 u        -> Peltier control action [-100%;+100%] WRITE ALLOWED

 If you need to declare new global variable you can do at the
    top of the program. In order to avoid any conflicts start
    all names with underscore sign _

 An iteration take about 500ms. If you need more you can
    increase the delay time adding a "delay(time)" directly
    into the function
*/

void stand_alone_controller()
{
  float Kp=1.0;
  float Ti=2.0;

  _error = setPoint-y1;
  _sumError += _error*dt/1000;

  u = Kp*(_error + _sumError/Ti);  // PI controller with
     positional algorithm

  delay(500);   // extra delay

}
```

Listing 1: Firmware extract: standalone controller function.

4.4 Firmware Upload

Once the code has been written, it is necessary to upload it into the micro-controller. In order to do this, before, you need to follow these steps:

1. Open *Arduino IDE*;
2. go to *Tool* $\rightarrow$ *Processor*;
3. select: *ATmega328P (Old bootloader)*.

In figure 3 you can see the section where you have to act.

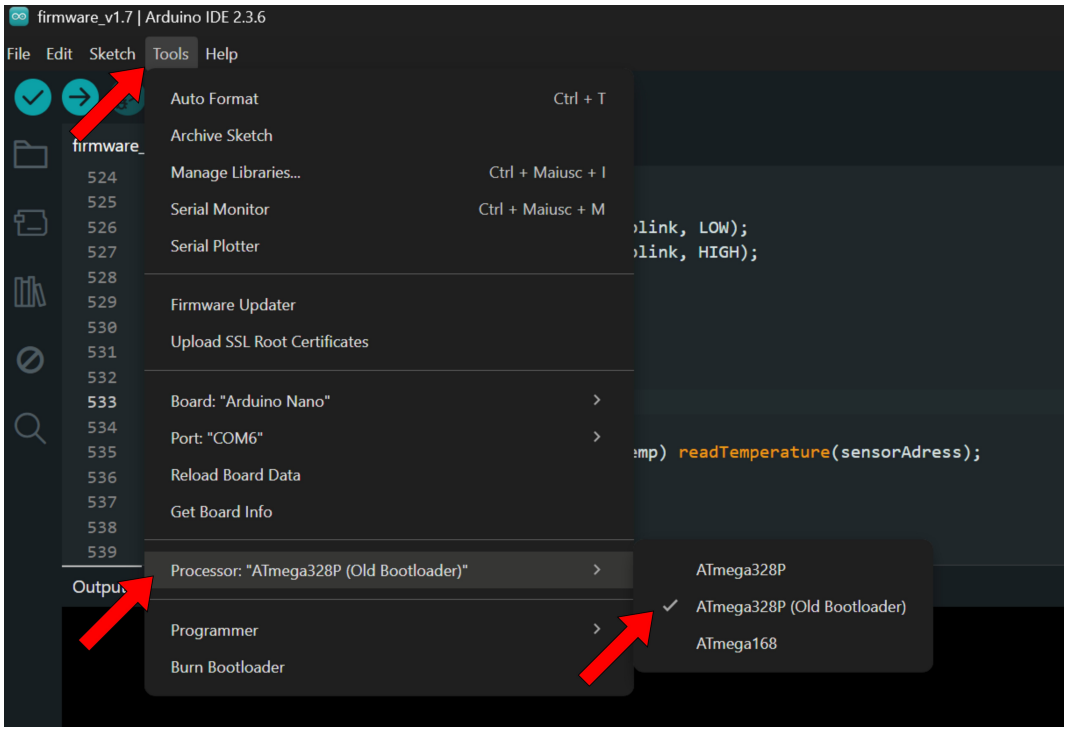


Figure 3: Bootloader setup

Now you have to install two libraries necessary to run the firmware, follow these steps:

1. Click the button with a library icon on the left;
2. digit *OneWire*;
3. click on *INSTALL*;
4. repeat the steps installing the library named *DallasTemperature*.

In figure 4 you can see the section where you have to act.

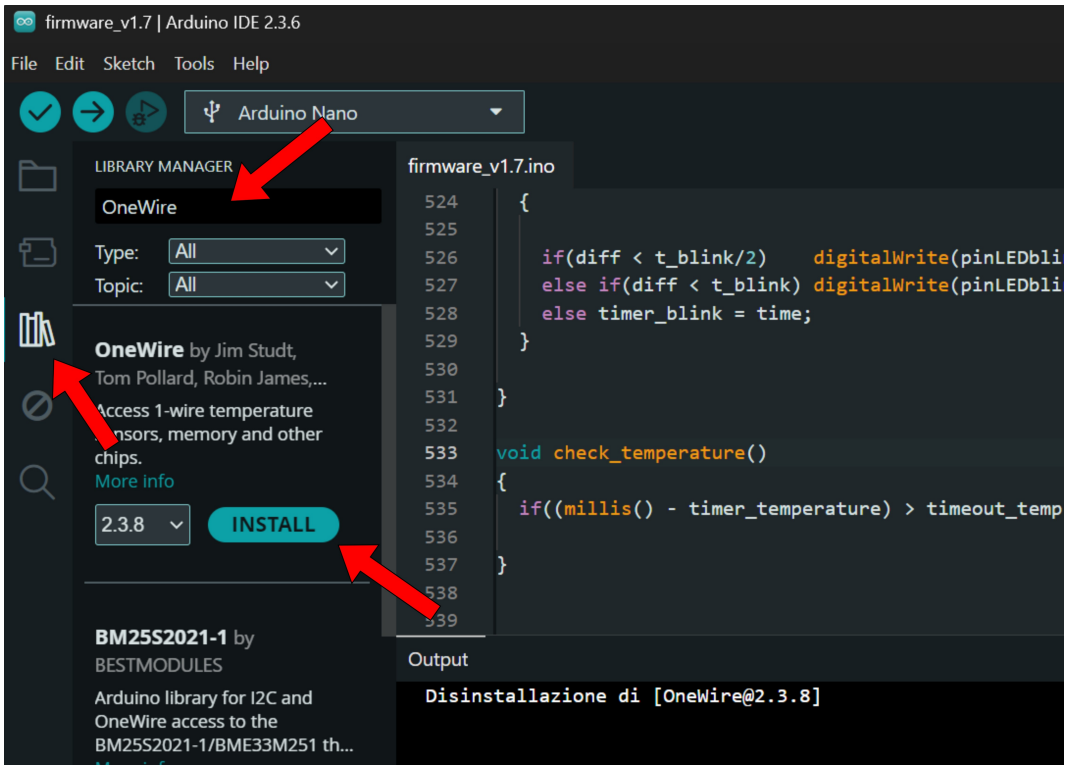


Figure 4: Library installation

Once you have followed the instructions above, you can proceed with the firmware upload: as explained in the *important safety notice*, ensure that the circuit is not powered, then plug the circuit with a USB cable into your PC.
Into *Arduino IDE* select the COM port assigned to your device, to discover the port number, you can look at which port appears after USB connection.
To upload the firmware, follow these steps:

1. Keep pushed the programming button on the interface circuit;
2. click the upload button on *Arduino IDE*;
3. wait for the end of upload;
4. relase the programming button.

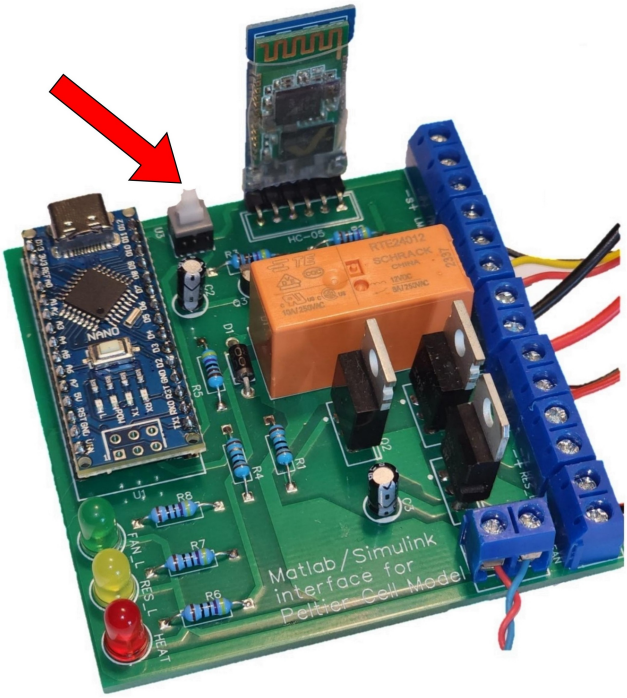


Figure 5: Programming button

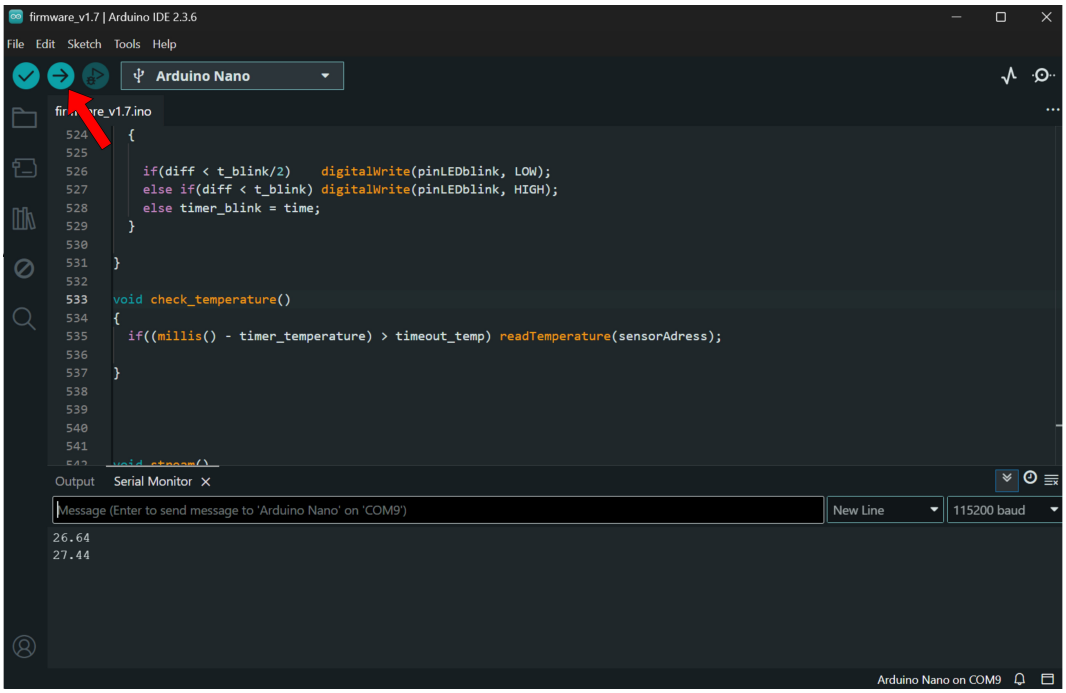


Figure 6: Upload button

4.5 Controller test

Once you have uploaded the new firmware, you can test the controller. Unplug the USB cable and power the interface with a 12V power supply. Now follow these steps into the *Arduino IDE*:

1. Open the serial connection by selecting the *COM PORT* and board type *Arduino Nano* (the COM port is the once that you have discovered during the Bluetooth connection);
2. select the *baud rate* at 115200 bps;
3. activate the standalone controller mode sending the string *%A*;
4. start the controller sending the string *%T*, the led placed on the micro-controller will start blink slowly;
5. open the serial plotter and look at the graph (setpoint, *y*, *u*);
6. now you can change the setpoint or the load disturbance using the serial command explained above;
7. if you want to stop the controller send the string *%S*.

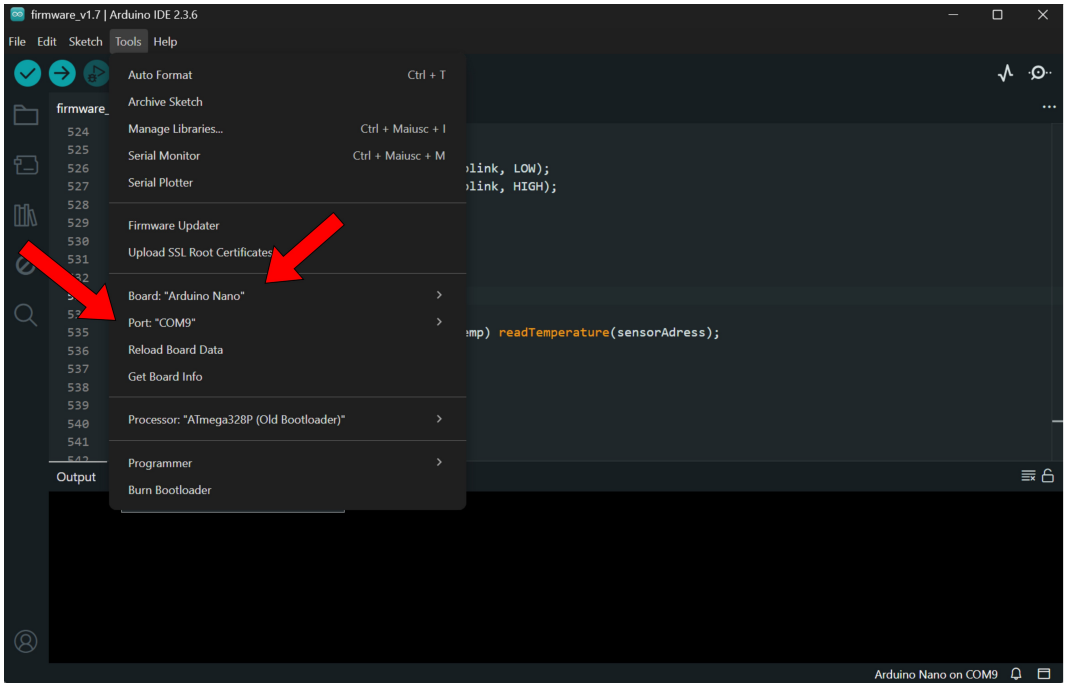


Figure 7: COM port setup

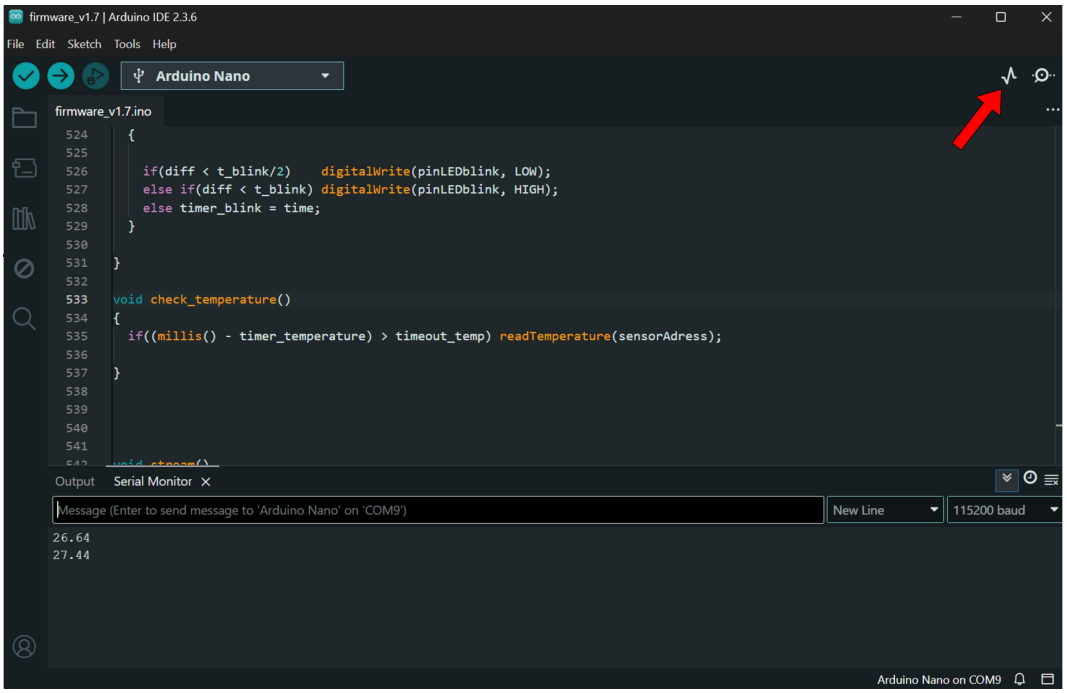


Figure 8: Plotter button

Figure 9: Serial plotter