Summary

The presence of the documentation proposes the design of an automatic system with temperature regulation of an electric furnace through the PID algorithm and using methods of the maximum degree of stability by performing the tuning of the parameters.

To control the temperatures, I used a PID controller. This controller uses gain constants k_p, k_i, k_d, these parameters determine the behavior of the PID controller.

Each gain constant has its role as follows:

- Proportional gain (K_P) determines the response of the controller to the difference between the reference value and the current temperature. It is the main component of the PID controller and contributes to the immediate response to the error changes. A large proportional gain value results in a stronger error response by making the system react more aggressively to deviations from the reference value. However, very high proportional gain values can lead to large oscillations or instability.
- The integral gain (K_I) takes into account the error accumulated over time and ensures the elimination of steady-state errors that cannot be corrected by the proportional component alone. A larger value of integral gain increases the effect of past errors, helping the controller to eliminate any remaining steady-state error. However, too high integral gain values can lead to overshoot.
- Derivative gain (K_D) is used to predict the future trend of the error based on the current rate of change. It dampens the response to rapid error changes and helps prevent overshoots and oscillations. A larger derivative gain value adds more damping to the system, reducing the response to rapid changes in error. However, very high derivative gain values can make the control too slow to react to sudden changes.

The working methodology of the PID controller can be seen in Figure 1.

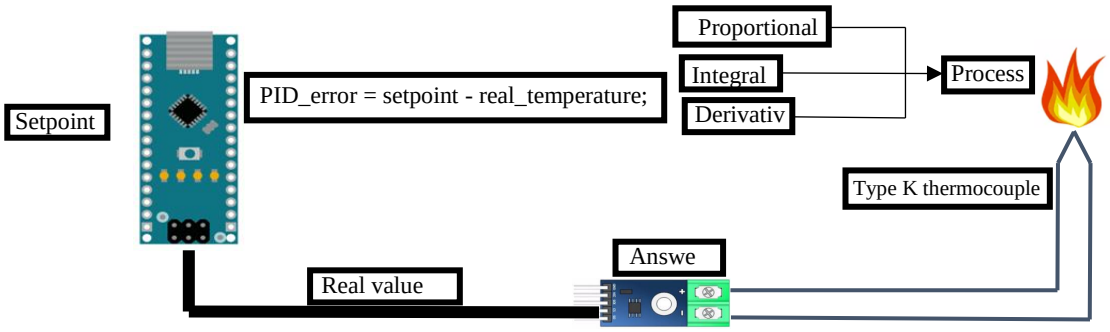

Figure 1. PID control process

All these constants make a PID control. In order to have the most effective temperature control you will have to choose values for each constant.

For any PID control we need to define the process, in our case it will be the final temperature we want to set.

The system will make the difference between the set value and the actual value, using the three constants: proportional, integral and derivative. With these we can change the output response and it will limit the power of the heating element.

To control the temperature we will need an answer. This response will be the actual temperature reading taken with the thermocouple.

Temperature measurement is done using the MAX6675 module and the K-type thermocouple.

The MAX6675 module is an integrated circuit that serves as a thermoelectric converter for K-type thermocouples. This module takes the thermoelectric voltage generated by the K-type thermocouple junction and converts it to a digital value representing temperature.

The MAX6675 is connected to the digital pins of the arduino to transmit the data from the thermocouple. The SCK pin is used to synchronize data transfer in serial mode. The CS pin is used to enable or disable the MAX6675. When CS is enabled (often via a LOW value on the pin), the MAX6675 will start converting the thermocouple data and send the data to the arduino. When CS is disabled (typically by a HIGH value on the pin), the MAX6675 enters a sleep mode and does not transmit data. The SO pin is the serial output of the temperature data converted by the MAX6675.

MAX6675 module which is connected SCK to pin D7, CS to pin D6 and SO to pin D5.

The mounting of the MAX6675 module to the arduino nano can be seen in Figure 2.

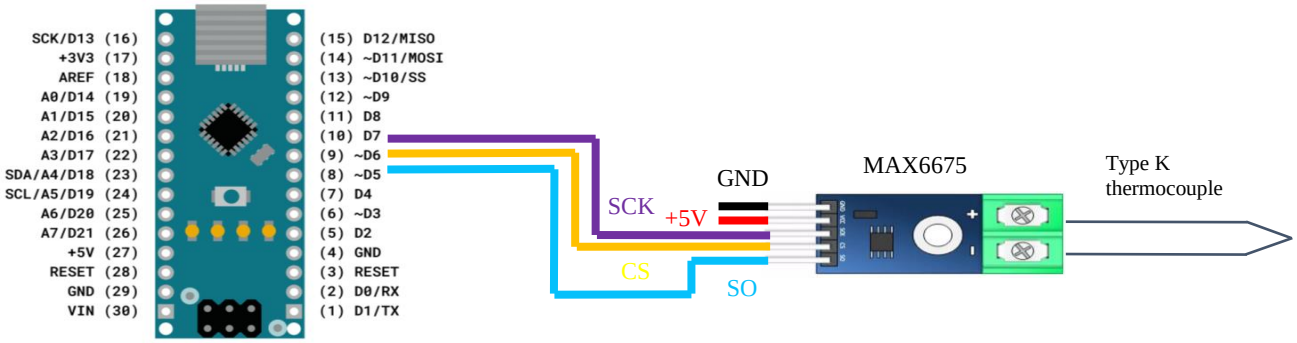

Figure 2. Connecting MAX6675 to arduino nano

Temperature measurement is done using the MAX6675 and K-type thermocouple. This came with a standard thermocouple but I chose to change it because the response to the temperature reading was very slow making the control difficult.

I chose to change the thermocouple that came with the MAX6675 module, this is still a type K thermocouple, but it is much more sensitive making the temperature reading faster helping to make its correction in a shorter time.

As a heating element for the test I used an electric grid. This is a simple electric grid quite old model as seen from 1975, as specified by the techie it has 600W and works at 220V. It has resistances as a heating element surrounded by ceramic chamotte.

The thermocouple is left on the network to be able to measure the temperature in real time, so the PID controller can act faster to control the temperature.

In the case of this network we can put the thermocouple in two locations. One location would be directly on the resistance between the ceramic fireclay, in this case we can read the direct temperature of the resistance and we can act quickly on its temperature changes, but the disadvantage is that the temperature of the resistance is not the same if we measure the temperature of the surface

of example on ceramic fireclay. The second location is to put the thermocouple directly on the ceramic fireclay, it heats up more difficult, but also retains heat making the temperature control and changeover very slow. We are interested in the temperature control on the chamotte if we want to use it in practice and to do this we use the mobile media function.

In order to be able to control the positive and the negative side of the AC signal, a triac is used. The triac will remain off until it receives a pulse to its gate and will remain on until the main input reverses polarity.

We will use the triac to control the AC voltage, to do this we need to detect the zero cross because our pulse needs to be in phase with the AC voltage and we will need to detect when the voltage goes from positive to negative or from negative to positive and be synchronized with the pulse so that it always goes off at the same time.

To be able to detect the zero cross we use two components, these are a full-bridge rectifier and a optocoupler. This is done because the microcontroller cannot work with negative voltage and voltage higher than 5V. The rectifier will only give a positive wave and the optocoupler will drop the voltage to 5VPP, by doing this we can read the signal with an Arduino. I added the 47kΩ resistor to limit the current, we have 230V and with a resistance of 47kΩ we will have a current of 4.89mA to protect the rectifier and the optocoupler which works with a current of 50mA and a 1kΩ pull-down resistor.

The mounting the optocoupler to the arduino and full bride rectifier can be seen in Figure 3.

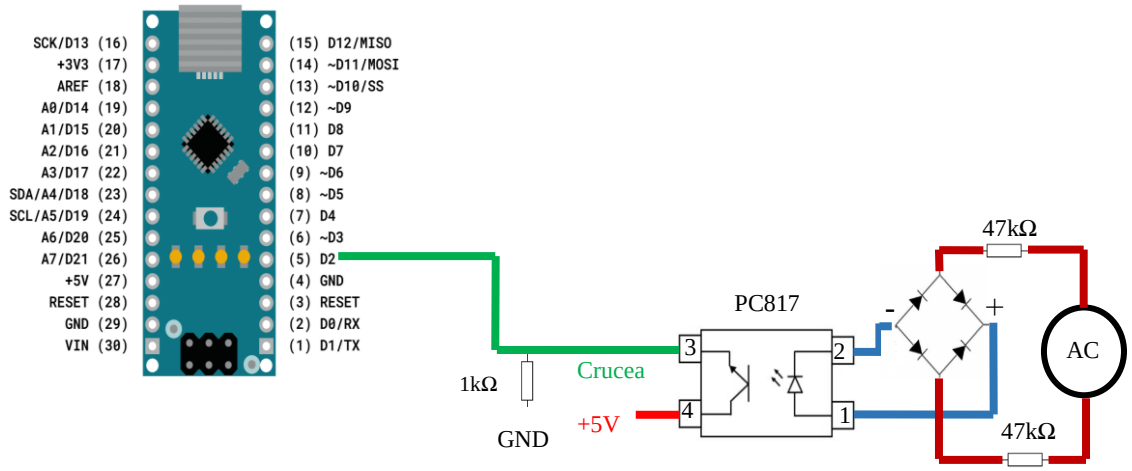

Figure 3. Mounting the optocoupler to the arduino and full bride rectifier

To control the triac a MOC3020 is used which is an optically isolated triac driver device, it contains a GaAs infrared emitting diode and a light-activated silicon bilateral switch which works like a triac due to its negative resistance characteristics which they allow it to turn on quickly once a certain level of applied voltage is reached.

How does this work:

- A little power is given to the MOC3020 LED and it lights up.
- When the LED lights up, the phototriac on the other side wants to turn on.
- The phototriac sends a small signal to the gate of the triac, When the gate of the triac receives this signal, it decides to open its "door" and let electricity flow.
- So by lighting the MOC3020 LED, you are sending a message to the triac to let electricity flow through it.

- It's like flipping a switch from afar without touching anything dangerous.

The Mounting MOC3020 to arduino nano and triac connected to the oven and socket can be seen in Figure 4.

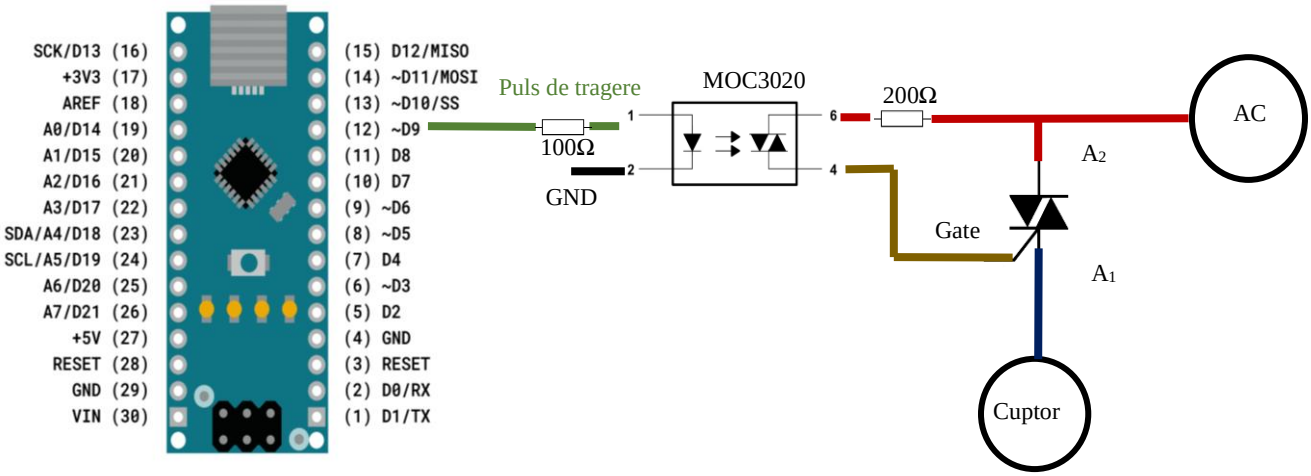

Figure 4. Mounting MOC3020 to arduino nano and triac

The 200Ω resistance allows sufficient current to pass through the MOC3020 LED to ensure correct operation as well as for triac.

The 100Ω resistor is to lower the current, since 20mA is recommended on the arduino.

Circuit diagram with all components connected to each other can be seen in Figure 5.

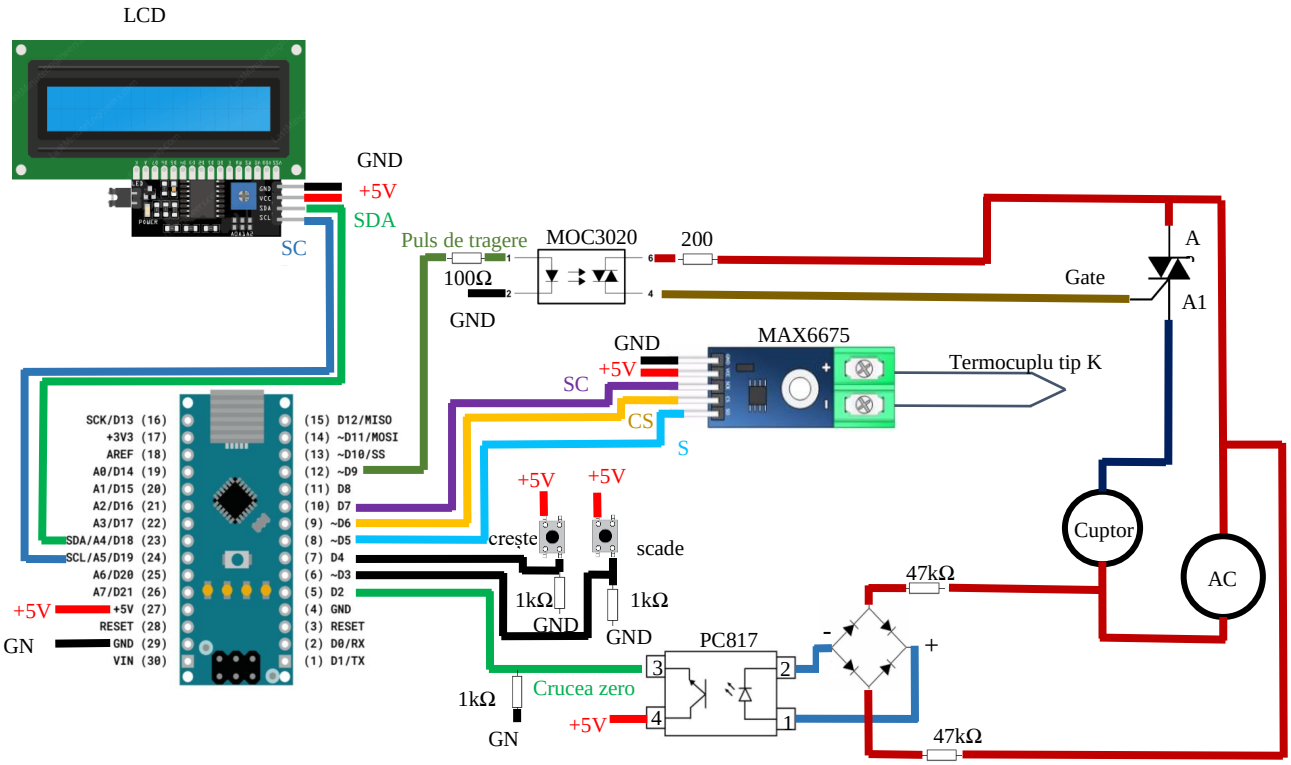

Figure 5. Representative circuit diagram

The final circuit is printed with the tinned components on it, minus the arduino nano board, which is soldered separately.

The circuit has the necessary holes to mount an arduino nano board as well as a heatsink for the triac.

The printed circuit has specific place for connecting LCDI2C module and Max6675 module.

All that remains to be done is to find the constant gain values of the PID controller. I'll start by leaving all the values at zero and just deal with the proportional gain. I will keep an eye on the temperature change and test to see how close they are to the correct value. I will change the setpoint using the buttons to see how much the temperature fluctuates around the setpoint. I'll start increasing the value of the constant in the code little by little and when I see the temperature value fluctuate as little as possible it means I'm getting close to a good value.

If I see that the temperature remains constant but does not yet reach the setpoint, it means that we will need full gain. It has the role of eliminating the stationary error.

Again I'll start at a small value like 0.1 and watch to see if the stationary error goes away. If I see that the stationary value disappears, i.e. increases to the setpoint value, it means that I am getting closer to the desired value. If I exceed the setpoint and it remains with oscillation, it means that I have a much too high value. If the temperature is going to have an overcorrection then we will use the derivative gain.

Again we're going to start doing tests but this time for k_d the derivative gain, we're going to start from a small value and keep increasing it value little by little until we see that it holds at the set temperature.

The adjustment of these values can be redone as many times as needed until we have a satisfactory temperature control.

The final values of the network test are $k_p = 79$; $k_i = 0.98$; $k_d = 0.49$;

About the second printed circuit board, Since the oven in the lab has a lot of power and the resistance in it can reach up to 24A, I wasn't sure if the printed circuit board I made at first would hold up, so I made some changes.

I changed the full bridge rectifier with a more powerful one, as well as the triac. I did this by removing the rectifier and triac from the original printed circuit board and fitting the new parts, another change was to the resistors at the input of the rectifier I used 68kΩ this will limit the current to 3.38mA.

The code is the same but because the triac and the rectifier were changed, I had to change the delay time for the triac trigger has a value of 7700us, and the gain values of the PID contoller are $k_p = 79$; $k_i = 1.25$; $k_d = 0.53$. This experiment with the second printed circuit board was also done with the same electrical grid, so the value of the PID controller might need to be change.