Arduino Temperature Controller

This repository contains a large Arduino sketch and a set of Python scripts that create a temperature controller using the *CN-0391 sensor shield* . The sketch and the Python scripts are meant to run in unison. Once the sketch is uploaded to the Arduino, the Python script communicates with the Arduino via serial to plot the data measured by the Temperature shield. The sketch is designed such that it can control up to four different devices and accurately control their temperature. Each port in the temperature shield can be connected to a thermistor to complete the circuit for maintaining a target temperature.

Dependencies

The code requires the following external dependencies:

Arduino: - None. All the code is included locally and simply needs to be uploaded to an *Arduino Uno* or a similar board with an *ATmega328P* microcontroller.

Python: The scripts were written with *Python 3.11* and will work with *Python 3.12* . They require the following libraries that can be installed using **pip** :

```
pip install numpy==2.3.0
pip install pyserial==3.5
pip install matplotlib==3.10.3
```

How to run the software

To run the software follow these steps:

i. Open the folder `arduino` and upload the sketch `arduino.ino` to the board using the Arduino IDE.
ii. Open a terminal on Unix-based systems or use IDLE (included with Python) on Windows, and run the script `main.py` inside the python folder with the command:

```
python3 main.py
```

Serial Communication API

Inputs: The Arduino can receive a string of numbers (separated by commas) that represent the target temperatures for the PID controllers associated with each port on the sensor module. For example:

```
30, 20, 63, 10
```

In this case, the first number corresponds to port 1, the second to port 2, and so on. If an input at a specific position is not a number, it will be ignored. For example:

```
na, 20, 30, na
```

This string will only update the PID controllers for ports 2 and 3. Not all ports need to have their target temperature set at the same time. For example:

```
20, 30
```

This command will only update ports 1 and 2.

Outputs: The output of the Arduino can be changed using three commands:

- `target` returns the current target temperatures for the PID controllers.
- `raw` returns unmodified measurements from the enabled sensor ports.
- `filter` outputs filtered measurements that go through a one-dimensional Kalman filter, reducing noise at the cost of a small delay.

By default, the Arduino will reply with **filtered** measurements. The serial output of the Arduino will be a string containing a series of numbers separated by commas, where each number represents the measurement for a specific port on the sensor shield. For example:

```
20.53, 31.36, 58.63, 100.34
```

In this case, the first number corresponds to port 1, the third number corresponds to port 3, and so on.

```
port1 = 20.53
port2 = 31.36
port3 = 58.63
port4 = 100.34
```

The meaning of these numbers depends on the previous command issued to the Arduino.

Power Control

The user can choose which digital output pins on the Arduino will be used to control a solid-state relay or a MOSFET. The Arduino activates the chosen electrical switch using a modified pulse-frequecy modulation (PFM) wave, controlling the average power dissipated by the heating element and precisely controlling the temperature.

This precise control is achieved through a PID controller that employs an alpha-beta filter to smooth raw temperature sensor measurements and obtain a low-noise approximation of the derivative of the temperature. Depending on the thermal inertia of the object being controlled, and the underlying system's time constant, the software can achieve precision of ~**0.5 degrees Celsius** near the target temperature. This will depend on the noise of the sensor attached to the temperature shield, as some sensors have more noise than others.

By using the serial interface to view raw temperature measurements and manually fine-tuning the PID and filter coefficients loaded onto the Arduino, it is possible to accurately control the temperature of an object with acceptable overshoot and response time.

Configuring PID Controllers and Signal Filters

Controller:

The temperature controllers utilize individual PID controllers to reach a target temperature. These controllers have three primary parameters that determine their behavior:

- **GAIN_P (Proportional Coefficient):** The proportional coefficient influences the response of the controller based on the measured temperature. A larger value causes the controller to inject heat into the system when the temperature is lower than the setpoint. It plays a significant role in controlling the temperature.

- **GAIN_I (Integral Coefficient):** The integral coefficient determines the magnitude of the time integral of the error between the target temperature and the measured temperature. This term eliminates any long-term, steady-state error at the cost of greater overshoot and ringing in the step response.

- **GAIN_D (Derivative Coefficient):** The derivative coefficient determines the magnitude of the time derivative of the measured temperature. This term is highly sensitive to noise in the measured temperature and is influenced significantly by the coefficients chosen for the Alpha-Beta filter. When tuned correctly, it damps the response of the controller, slowing its reactions while removing ringing from the system. However, if too large, it can cause high-frequency oscillations or instability.

- **MIN and MAX:** These are the saturation limits of a PID controller that clamps its output to avoid integral windup, wherein the integral term cannot change if the output of the controller is saturated.

Filters:

The Alpha/Beta coefficients smooth raw temperature readings and are part of the PID controller. This filter is essential in reducing noise in the derivative term.

- **ALPHA:** This coefficient smooths any abrupt changes in the raw measurements. A value closer to zero causes a slower but smoother response. Constraints: `0 <= ALPHA <= 1`

- **BETA:** This is the derivative filter. It increases the delay and overshoot in the signal response due to ALPHA, causing a phase delay between the real derivative and smoothed estimate. A value closer to zero causes a smoother and more delayed response. Constraints: `0 <= BETA <= 1`

The Kalman Filter modifies the Arduino's serial output and requires two coefficients:

- **ERROR:** If the filter's value is within this error band relative to the unfiltered value, the filter will respond very slowly to any changes in the raw signal. This is equivalent to the standard deviation of the unfiltered signal. In other words, measurements within this error band are suppressed.

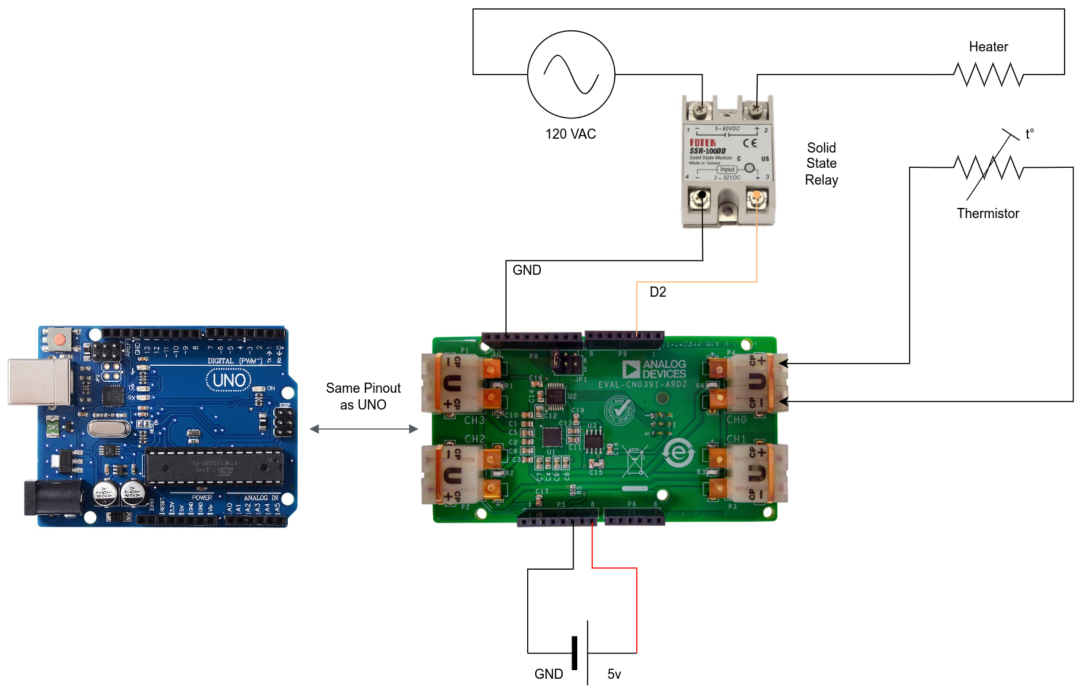
- **NOISE:** This coefficient determines the general response of the filter. A larger number makes the filter react more quickly to changes in the raw signal. A smaller value makes the filter react more slowly. A smaller value also smooths out any fluctuations in the raw signal at the cost of response time. Constraints: `0 <= NOISE <= 1`

Note: Different sensor types can be connected to the CN-0391. These sensors are physically distinct, and they require unique calibration parameters to accurately measure temperature. The user must identify the type of sensor they are using, either through documentation or by comparing the output of CN-0391 with another temperature sensor that is accurately calibrated.

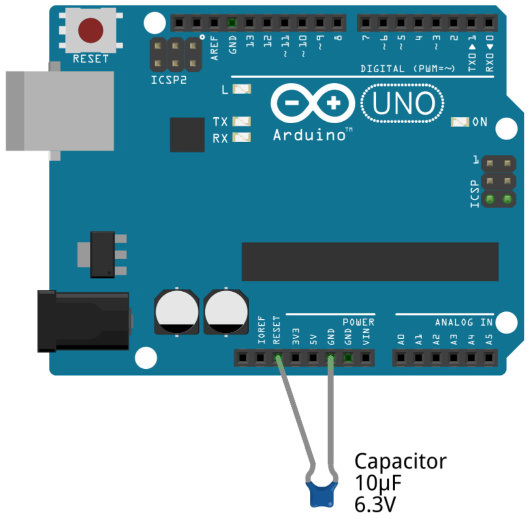
Circuit

The Arduino should be connected to a 5V power supply, and the heating elements should be connected to a solid-state relay (SSR) and an alternating current (AC) source. The SSR must be connected in series between the power source and a heating element. Note that the input side of the SSR module should be connected to the ground of the 5V supply and to one of the digital output pins on the Arduino. The input to the SSR can be attached to any digital pin that is enabled through software. See the attached schematic:

CN0391 Diagram



Due to how Arduinos are electrically connected, they will reset whenever a new serial communication is established. There is no straightforward method to prevent this behavior with software. To avoid this issue, add a small capacitor between the reset pin and ground on the Arduino. Alternatively, cut the traces that connect to the reset pin to permanently disable it; however, this interferes with uploading new firmware to the Arduino. For an easier solution, physically connect a capacitor to the Arduino *after* firmware is loaded and remove the capacitor *before* reprogramming it.



In practice, this behavior means that whenever a serial connection is lost and reestablished, the Arduino will reset and a new target temperature must be specified. This discontinuity could cause the device being controlled by the Arduino to experience a significant change in temperature.

References

Filters :

- Alpha Beta Filter
- Arduino - 1D Kalman Filter
- Kalman Filter for 1D Motion with Acceleration and Bias

Signals :

- Pulse Frequency Modulation (PFM)

Hardware :

- Analog Systems CN-0391 Module
- Disable Arduino Auto-Reset