Connor McPartland

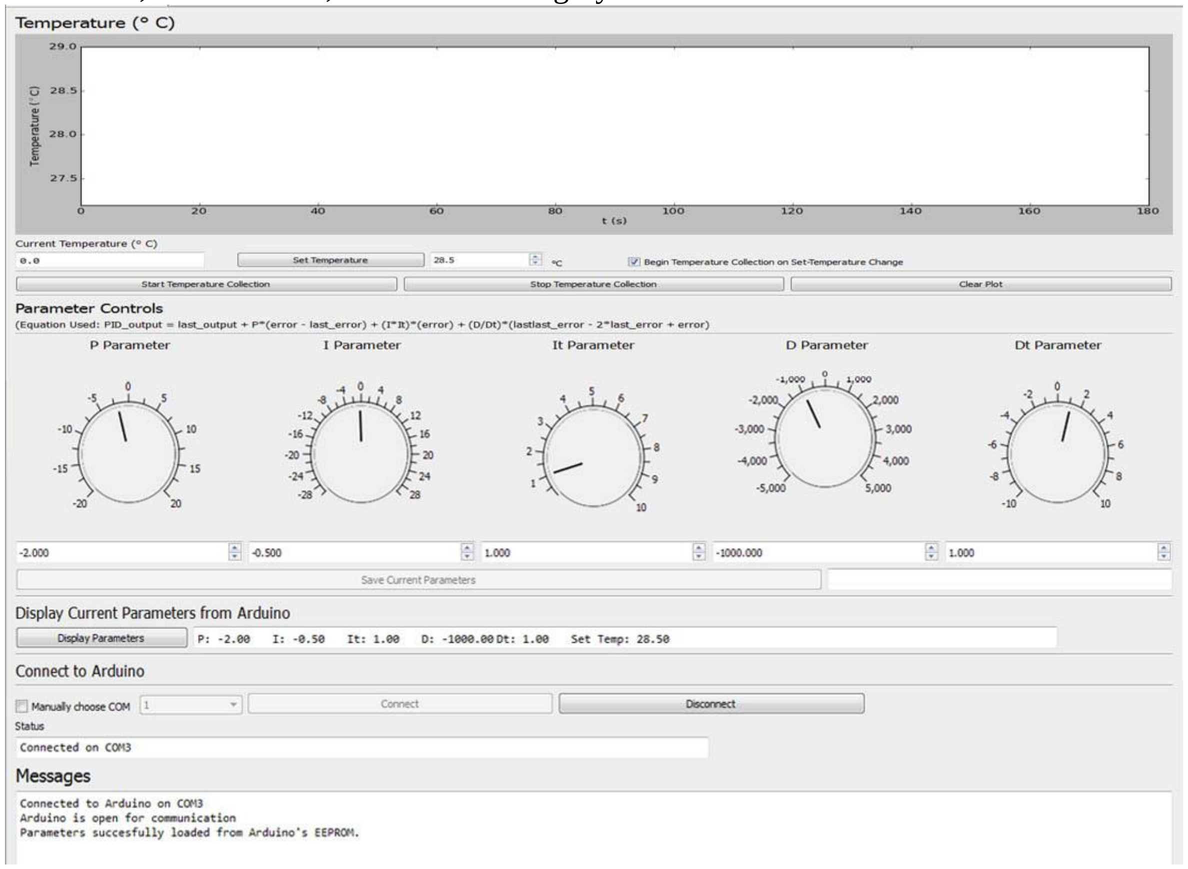
Spring 2013 Project Conclusion

Introduction

This semester I worked on the completion of a PID temperature controller. The project consists of an Arduino, which controls the heating/cooling element, and a circuit that uses the output from the Arduino to appropriately power the heating/cooling element. Before I started my work, the project consisted of the circuit mentioned above and an Arduino sketch that measured the ambient temperature using a thermistor. The sketch also used the settings on 5 potentiometers to control the 5 parameters of the PID equation: P, I, It, D, and Dt. The addition I've made is a user interface that allows the user to set the set-temperature, plot the temperature over time, change the PID parameters, and save parameters for future use. The Arduino sketch has been completed to include a PID algorithm and properly communicate with the interface.

Interface

The interface, shown below, was created using Pythics:



Current Temperature (° C): 0.0

Set Temperature: 28.5 °C

[✓] Begin Temperature Collection on Set-Temperature Change

Parameter Controls

(Equation Used: PID_output = last_output + P*(error - last_error) + (I*It)*(error) + (D/Dt)*(lastlast_error - 2*last_error + error))

P Parameter	I Parameter	It Parameter	D Parameter	Dt Parameter
-2.000	-0.500	1.000	-1000.000	1.000

Display Current Parameters from Arduino

```
P: -2.00   I: -0.50   It: 1.00   D: -1000.00 Dt: 1.00   Set Temp: 28.50
```

Connect to Arduino

[ ] Manually choose COM 1

Status

```
Connected on COM3
```

Messages

```
Connected to Arduino on COM3
Arduino is open for communication
Parameters succesfully loaded from Arduino's EEPROM.
```

The controls include:

- Temperature:
    - “Start Temperature Collection”: being temperature plotting
    - “Stop Temperature Collection”: end temperature plotting
    - “Clear Plot”: clear the plot
- Parameter Controls:
    - “Set Temperature”: a box to enter a chosen set-temperature, and a button to set the set-temperature to this value. There is a checkbox to turn on/off automatic plotting on a set-temperature change
    - 5 knobs to control the 5 parameters, as well as text boxes to enter a number for each parameter
    - “Save Current Parameters”: save the current parameter to the Arduino’s memory for future use
- “Display Parameters”: display the parameters currently being used by the Arduino
- Connect to Arduino:
    - “Connect”: connect to the Arduino
    - “Disconnect”: disconnect to the Arduino
    - “Manually Choose Com”: manually choose the com to connect to using the drop down box next to the checkbox
    - A small text-box indicates the status of the connection

The graph at the top of the interface plots temperature versus time, where $\boldsymbol{t}$ **=** 0 s is the time when temperature collection is started. The current temperature is displayed in the box below the graph. The ‘Messages’ box includes a record of the executed commands, i.e., “Connected to Arduino on COM3”, “P set to 1.5”, etc.

Python Code

The commands made through the interface are carried out in Python code. The program connects to the Arduino using the pySerial library. If no manual COM choice is chosen, the program will cycle through COMs 1 through 10 until a connection is made. If a connection is made, initialization begins. Initialization includes two steps: interface controls are enabled, and the last saved parameters are loaded and the parameters are changed to these values. Any errors in this process will be printed to the ‘Messages’ box. Once initialization is finished, the user is free to change the PID parameters or to start temperature collection.

The program and the Arduino interact with simple character commands – the user interacts with the interface to begin a command, the program sends a character to the Arduino, and the Arduino, continuously checking the serial port, reads the character and executes a command based on that character. A simple example: the user clicks the “Start Temperature Collection” button, the program sends the character ‘t’ to the Arduino, the Arduino begins writing the recorded temperature to the serial, and this temperature is plotted.

PID parameters are changed by turning the appropriate knob and this change is reflected in the text-box below the knob, or by entering the desired value directly into the text-box, in which case the knob setting is changed to reflect this value. The program handles this command by writing the appropriate character (program writes ‘p’ to change the P parameter) followed by the new value of this parameter to the Arduino.

Temperature collection is started by pressing the “Start Temperature Collection” button. When this button is pressed, the Arduino is commanded to write the current recorded temperature to the serial port. To speed up plotting, the program collects the data from the serial port for 0.5 seconds and compiles this data (temperature and time-stamp) into an array, and then plots the data.

Some user-friendliness has been added to the program and interface. Normally, problems will arise when a serial connection is made and then unexpectedly broken. The COM port can get stuck open and will be blocked to further interaction. The interface has a “Disconnect” button to close the Arduino, but the program will also properly close the serial port if the user exits the program. If parameters are changed on the interface, the “Save Current Parameter” button will become enabled. When it is clicked, the parameters are saved and the button becomes disabled. If the parameters have changed and have not been saved and the user chooses to exit the program, the interface will prompt the user to save the parameters or continue exiting the program.

Arduino Sketch

The Arduino sketch is a continuously-run loop. This is what is carried out within the loop:

- Check the serial port to see if any data is available.
- If data is available for reading from the serial port, read it. The only thing written to the serial port should be a character sent by the Python program
- Based on the character, execute a certain command (if ‘p’, read serial again for float value and change P to this value, if ‘s’, read serial again for float value and change the set-temperature to this value). Here are some special cases:
    - User clicks “Start Temperature Collection”; change Boolean variable **python_wants_data** to true. Clicking “Stop Temperature Collection” will set **python_wants_data** to false.
    - User clicks “Save Current Parameters”; save the P, I, It, D, Dt variables to the Arduino EEPROM.
- Calculate the temperature using the resistance across the thermistor
- Based on the most recent error (current_temperature – set_temperature), the last error, and the error prior to the last one, calculate the appropriate output using the PID algorithm:

$$\text{PID\_output} = \text{last\_output} + P * (\text{error} - \text{last\_error}) + (I * \text{It}) * (\text{error}) + (D/\text{Dt}) * (\text{lastlast\_error} - 2 * \text{last\_error} + \text{error})$$

- Map PID_output to an appropriate range for PWM output (80 to 220, for example), and write PID_output to the appropriate Arduino pin to drive the TEC
- If **python_wants_data** is true, write the current temperature to the serial port.

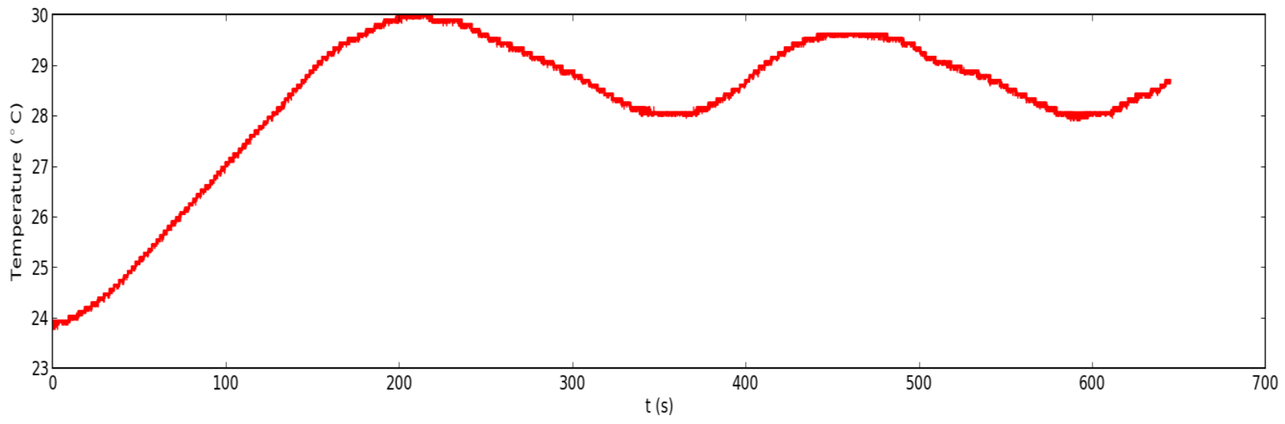
Circuit

The circuit has not been changed, but here is an overview of how it works:

1) The Arduino measures the resistance across the thermistor (a TH10K), and the temperature is calculated from this value. The temperature determines the output of the PID algorithm.
2) The Arduino outputs a PWM signal whose duty cycle is based on the output of the PID algorithm. A PWM signal is a 5V peak-to-peak signal whose duty cycle depends on the value written to the specific Arduino pin (values range from 0, resulting in 0% duty cycle, to 255, resulting in 100% duty-cycle).
3) The PWM signal is passed through a low-pass filter to convert it to a DC signal whose amplitude is directly proportional to the duty cycle, with a range of 0 to 5V.
4) The PWM is divided by 10 using a voltage divider.
5) An op-amp subtractor serves to subtract some voltage from this DC signal. Because this process is not exactly linear, the result is usually an output range of -.3V to +.3V.
6) This DC signal, now taking on negative and positive values, is multiplied by 10 using an op-amp to a range of -3V to 3V. The signal powers the heating/cooling element, nominally a thermo-electric cooler (TEC).

Results

A 1 Ohm resistor has been used as a heating element in place of a TEC to test the circuit. The resistor was kept in thermal contact with one end of a metal set-screw while the other end of the set-screw was kept in contact with the thermistor. Since the resistor has no cooling function, the Arduino sketch was slightly modified so that the low-end of the output cuts off the current going through the resistor so that the resistor cools by releasing energy to the environment. Here is the result when the set-temperature is set to 28.5° C:



**Note: The PID parameters are not properly tuned, so the temperature overshoots and then oscillates about the set-temperature.