Final Project Report: *Temperature Controlled by a PID*

Shane Price [Section 1]
Kieran O'Connor [Section 1]
Rowan University

May 9, 2019

1 Design Overview

The team worked to control the temperature in a confined space to have fluctuation of less than 1°C. The team was able to achieve this task by utilizing a PID controller, the controller was implemented using code on an Arduino. The purpose of this was for error correction, decreasing the rise time and improving the transient response to have as minimal oscillation as possible. These topics will be discussed more thoroughly throughout the continuation of the report.

1.1 Design Features

The system was implemented using an Arduino, a LM35 (PTAT temperature sensor) and a 5V regulator along with many common components. The features of this system can be seen listed below:

Design Features:

- Temperature range of 72°C to 118°C
- Maintain a transient temperature (oscillation of +- 0.5°C)
- Deadzone control, for the cooling element (fan)

1.2 Featured Applications

Below you can find a list detailing a handful of the applications this system can work with to help improve thos applications. This system is a first order system which allows for this to work with many first order systems which can be very common in the real world.

- Accurate Thermostats
- Carbon Fiber Oven Control
  A carbon fiber oven needs to maintain a specific temperature and reach that temperature quickly for optimal curing strength of the carbon fiber, this system meets those requirements to a T.
- Cruise Control
  Most cars with cruise control have a speed range of +-2 to 3mph, this system could help keep the car traveling at the specific set point.

1.3 Design Resources

The link attached will direct you to the code used in the system along with a README, and the schematics; all necessary to recreate the system:
`https://github.com/oconnork9/PIDTemperatureControl_O-Connor_Price`

2 Key System Specifications

The system created was able to be considered a fairly stable system due to the error correction and ability to maintain the set point temperature. This system is not perfect by not having as small of a rise time and % overshoot as desired. However it seemed the correction of the % overshoot would only create a larger error in the system, therefore it was decided among the team to keep the 'k' values that were causing these issues.

PARAMETER	SPECIFICATIONS	DETAILS
Transient Response	+-0.5 °C	The system can maintain a set temperature by the specification listed.
Temperature Range	72°C to 118°C	The system can not reach a lower temperature due to the close proximity of the heating element, while the heating element also maxes at 118.
Rise Time	2 minutes 7 seconds	The system was able to achieve a 2 minute rise time before a large overshoot would occur.
% Overshoot	+-2°C	The system attained a small portion of overshoot caused by the kp and ki values. Attaining higher temperatures it would overshoot by 2 degrees, going to lower temperatures it would go under by 2 degrees but quickly fix this in the transient response.

3 System Description

This system was used to maintain a set temperature, input by a potentiometer, this system can not only be used in the case of temperature control but many other forms of control for first order systems. This system was created to lower the rise time of a system and create a better transient response of the system. This section will be used to detail how this was done from a top level perspective with schematics and block diagrams.

3.1 Schematics and Implementation

Below the schematics used for this system can be found. Schematic 1, shows how the 5V regulator (LM7805) has a resistive load, a 10W 10 ohm power resistor to help heat the 5V regulator quicker, this is the schematic for the heating element of the system. Observing Schematic 2 it can be seen the connection of the PWM from the Arduino of pin 13 to the L272M operational amplifier to control a 3 pin fan as opposed to using a low side switch. Schematic 3 demonstrates how the potentiometer connects to the Arduino, the potentiometer is used to change the set temperature that the system is attempting to reach. While Schematic 4 demonstrates the connection of the temperature sensor, and the output to a simple RC filter to the arduino. The RC filter is currently a low pass filter, filtering out all frequencies above 25Hz.

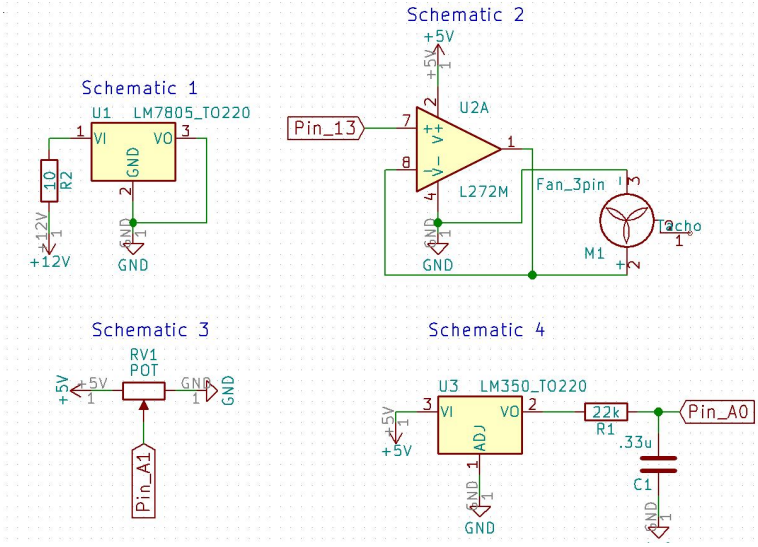

Figure 1: Set Of Schematics

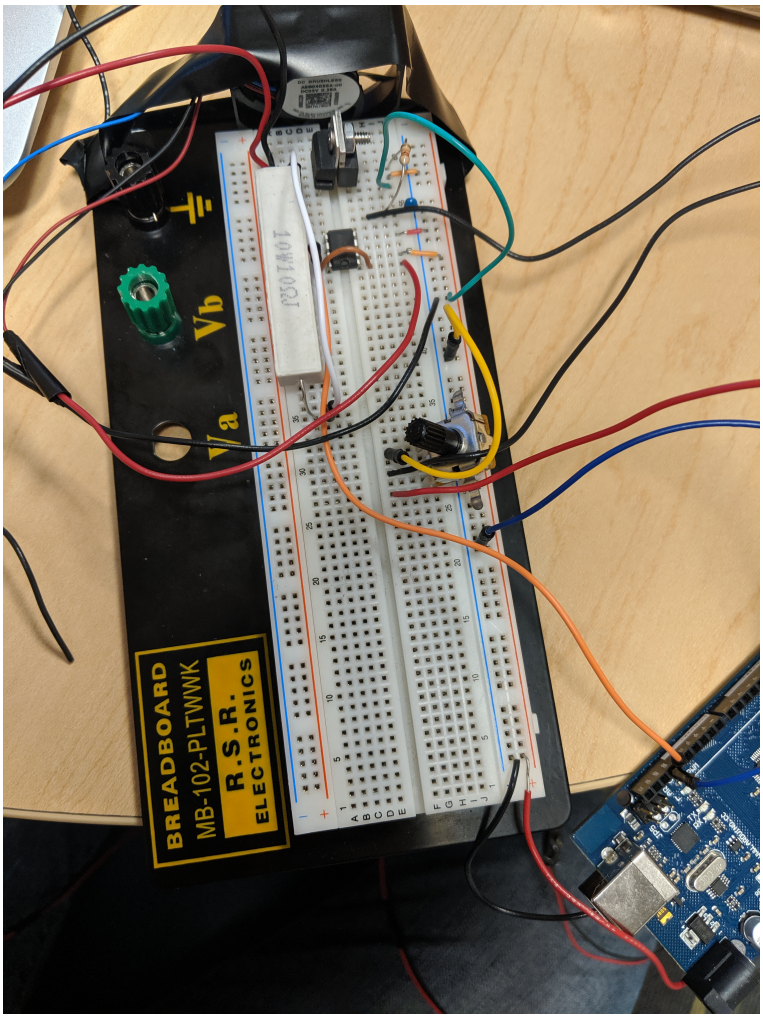

Figure 2: Picture of Live Circuit

3.2 Detailed Block Diagram

Figure 3, below, depicts the top level block diagram of how the system created functions. This shows the connections and their direction (input or output) in regards to the Arduino. The dashed line depicts heat transfer from the 5V regulator (LM7805) to the PTAT Temperature Sensor (LM35).

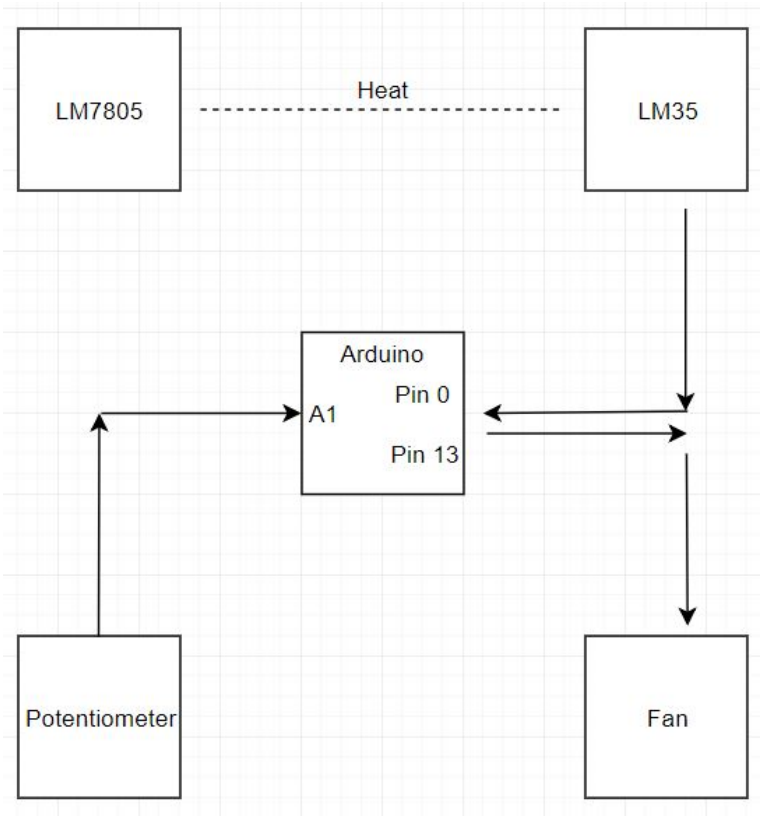

Figure 3: Block Diagram of Entire System

3.2.1 LM7805 Block

The LM7805 is the 5V regulator, referencing Figure 1 and Schematic 1, it can be understood all the components that create the LM7805 block. The function of this block is to heat the system, the 5V regulator will be receiving voltage continuously so that the regulator will consistently be exuding heat. In series to the input voltage of the regulator and the power supply is a 10W 10 ohm power resistor, this resistor is responsible for putting a load on the system allowing the regulator to dissipate heat to the system.

3.2.2 LM35 Block

The LM355 is the temperature sensor, referencing Figure 1 and Schematic 4, you can notice that the output of the temperature goes through an analog low pass filter before entering pin 0 on the Arduino. The filter is intended to block noise from external sources, the main source for noise is the fan. The fan creates a large amount of noise when it is being PWM'd, to counter this noise that was corrupting our signal, the filter was put in place with a cutoff frequency of 25Hz. Filtering this noise out helps the system gather more accurate data, in turn allowing the system to more accurately account for the error correction.

3.2.3 Potentiometer Block

The potentiometer is used in the system to set a temperature to reach and maintain. The potentiometer is used simply as a voltage divider with the output voltage entering the analog pin 1 on the Arduino. The microcontroller then uses the value it receives to determine the setpoint temperature desired.

3.2.4 Fan Block

Referencing Figure 1 and Schematic 2, it can be seen the fan is connected to an operational amplifier as a voltage follower. The output of the opamp triggers the fan to 'kick' on for the PWM. This circuit is working similarly to a low side switch for a PWM.

3.2.5 Arduino Block

The Arduino is being used to control the entire system. The Arduino takes the data and calculates the temperature reading, the setpoint, and then sends a PWM signal based on the implemented PID controller. This will be discussed in greater detail further in the report.

3.3 Highlighted Devices

Below is a list of the main components used to create the system:

- Arduino Mega
- LM35
- LM7805
- L272M
- CPU Fan

3.3.1 Arduino Mega

It was determined to use an Arduino due to the copious amounts of example code as well as the ease of implementing the PID control. This does not need to be extremely detailed. Just talk about the part, what is it doing in your system, and why did you pick it (if you had to spec it out). If you didn't have to spec it out, then talk about what makes it suitable for your application.

3.3.2 LM35

This component was used due to the simplicity of determining the temperature it is reading, with a linear slope. This was chosen due to its cost effectiveness, and the power requirements easily meshed with our low power system.

3.4 LM7805

This voltage regulator was chosen to be the heating element to the system due to it being able to dissipate a large amount of power, in turn creating a large amount of heat for the system. The regulator was ideal, as it is a 5V regulator and supplying more than 5V was readily available to the team.

3.5 L272M

The operational amplifier was chosen to be used to help reduce interference from external sources and it can also be used to trigger the fan by an input, the PWM from the Arduino. This was chosen as it can help maintain a signal without external interference which in turn can increase the accuracy of the system.

3.6 CPU Fan

The cpu fan was chosen as it can be run off of 5V and does not draw a lot of current. It fits well for this system as the PWM controlling the fan matches the needs of the fan, allowing for the fan to span its' entire range of speed.

4 SYSTEM DESIGN THEORY

This project was created in several different steps. The project started as an open loop system reading the temperature from the heat source, then the team designed the closed loop system to control the fan for the PWM, then the team implemented a PID controller to correct the error between the set point temperature and the read temperature.

Below each portion of the project will be discussed in greater detail:

1. Open Loop System Design
   The open loop system portion of the project, took data from the LM35 to process in the Arduino to determine the temperature of the system. This portion of the system also included the implementation of a 10k ohm potentiometer as a voltage divider to give the user the ability to set a specific temperature for the system to reach.

2. Initial Control Design
   The initial control of the system was implemented by using PWM to a CPU fan to help cool the system to reach lower set points. This control was created as a 'bang bang' type system, if the temperature read was higher than the set point the fan kicked on, if the temperature read was lower than the set point the fan turned off. At this stage the team determined that the PWM of the fan worked and the PID controller could now be tuned to meet the specifications of the project.

3. Iterative Steps PID Tuning
   During this stage of the project the PID controller started at 0 for each kp, ki, and kd; the team began by tuning the kp value first. The team increased the kp value until the system began to have overshoot, once overshoot appeared the team went to the previous value. Tuning the ki value was the next logical choice, this also helped lower the rise time and helped eliminate the steady state error. The ki value was increased until overshoot began to occur as well and taken the previous value before overshoot occurred. Finally the kd value was tuned, tuning the kd value was meant to decrease the oscillation in the transient response.

4. Final Design
   Using these steps the team was able to build a closed loop system that could maintain a transient response within +-0.5°C with a rise time of a bit over 2 minutes. The transient response of the system can be seen in Figure 4.

5 Code Review

5.1 Initialization

To use the the code that team provided an Arduino was used with a library by Brett Beauregard. This can be done by going to the Arduino IDE going to "Sketch-Include Library-Manage Libraries" then searching PID and look for Brett Beauregard's name. The user will also have to set the COM port of the Arduino and the type of Arduino being used this can be done in the Arduino IDE in "Tools" in "Port" and "Board".Other than this for the setup of the system as long as the circuits are followed exactly no other initialization is necessary. However, if the supplied voltage is changed for the conversion equation for the temperature measurement and desired temperature readings. If another temperature reading method is use this will greatly increase the difference the conversion would have to change completely rather than just the voltage from 5V to

Figure 4: Transient Response

let say 3.3V. If a different heating method is used and/or a different cooling method is used other than a CPU fan the steps described in Section 4: System Design Theory would have to be repeated.

5.2 Sensor Acquisition

The LM35 was used as the temperature sensor for the system, this sensor would send an analog signal to the micro-controller. This voltage input to the Arduino would be used to determine the temperature by using Equation 1. The analog signal had to be passed through an ADC, the ADC mapped the voltage input to the system to the 10 bit data value of 0 to 1023, then utilized Equation 1. This was also completed for the desired temperature, set from the potentiometer.

$$V_out = 10mv/^\circ C * Temperature \tag{1}$$

5.2.1 Signal Conditioning

The system passes through two separate filters, the first filter is from the temperature sensor to the Arduino. The initial filter is a low pass filter with a cutoff frequency of 25 Hz. The purpose of this filter is to filter the noise introduced into the system from the fan when the fan is turned on trying to cool the system. This analog filter works very well in this case, there is still noise in the system but at a minimal amount where it does not greatly affect the data reading to the Arduino. The second filter implemented in the system is a digital moving average filter. This filter takes the average temperature reading of the last five samples. This helps the system get a more accurate reading instead of having wide fluctuations. The fluctuations without the digital filter were cut down drastically by the analog filter, together they are two forms to get more accurate results.

5.3 Error Control Calculation

The error in the system was calculated by taking the difference of the temperature being read by the temperature sensor and the desired temperature set from the potentiometer. Using these two values the PID control loop would then adjust the fan PWM accordingly to meet the desired temperature. The PID control loop would use the kp, ki, and kd values we set in the system to match temperatures, in this systems case the kp, ki, and kd values were 64, 4, and 12 respectively. The kp was used to lower the rise time and would introduce overshoot at larger values of kp. The ki was used to help reduce the rise time as well and to help eliminate the steady state error. While the kd was used for future values, attempting to introduce the system to a transient response.

5.4 Actuation

The actuation of the fan is determined by the PID Controller however since there was a few errors needing to be worked around this is not completely true. Since there was a dead zone of the CPU fan a solution needed to be implemented into the code. To do this a kick function was applied and a certain range was sent on the PWM making the fan was set to 20 percent duty cycle if the Output of the PID was lower than 35 percent duty cycle and above 0 percent duty cycle. This greatly reduced the error making the system accounting for the PWM percentages that would not work. The kick function was also implemented to make the 20 percent duty cycle possible because of its lack of inertia the dead zone of the CPU fan is larger trying to pick up speed rather than decreasing. To get around this the kick function increased the fan duty cycle to 100 percent for around a one fiftieth of a second to increase the speed making lower percentages of PWM possible.

6 Design Files

6.1 Schematics

Below the schematics for the circuitry can be seen.

6.2 Bill of Materials

- 1 - Arduino Mega
- 1 - 10k ohm Potentiometer
- 1 - 22,000 ohm Resistor
- 1 - 0.33uF Capacitor
- 1 - LM35 PTAT
- 1 - LM7805 5V Regulator

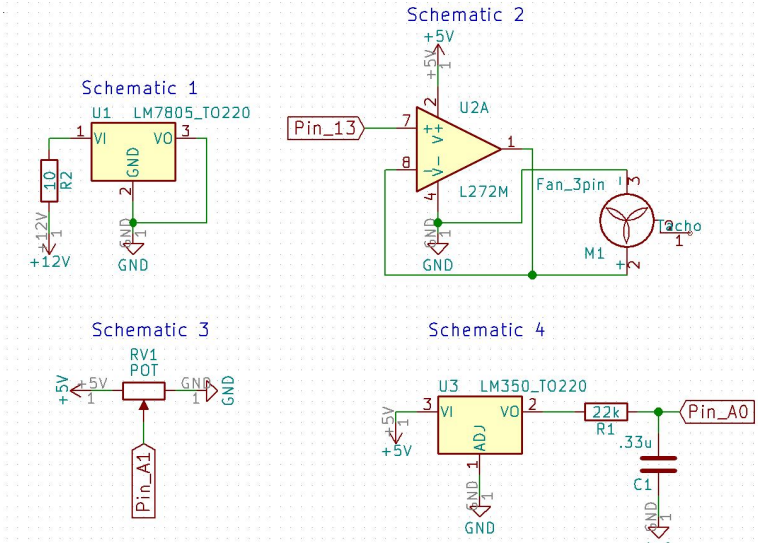

Figure 5: Set Of Schematics

- 1 - 10W 10 ohm Power Resistor
- 1 - L272M Operational Amplifier
- 1 - 5V CPU Fan

6.3 Code

The code can be accessed using the link provided to the teams Github.
`https://github.com/oconnork9/PIDTemperatureControl_O-Connor_Price`