Final Project Report: *Closed-Loop PID Fan Control System*

James Cipparrone [4], Alex Revolus [4], David Sheppard [1]
Rowan University

May 10, 2019

1 Design Overview

In today's modern world closed-loop control systems are incorporated into every crucial device out there. For example when applying cruise control to a car, the driver decides what speed the vehicle is supposed to maintain and the vehicle needs to autonomously hold the desired speed no matter what situation. Situations varying from driving up a steep hill or going down the hill, the vehicle will adapt to its conditions. That's what makes closed-looped systems so important. A closed-loop system is able to correct itself without the need of a user continuously changing the system. In this final project, the closed-loop system that is implemented is a temperature control system.

1.1 Design Features

The heart of the control system lies in a software PID control, which determines the PWM duty cycle produced by a Texas Instruments MSP430F5529LP microcontroller. The PID controller is entirely software-based and is written in C++ using TI's Energia development environment.

General design features:

- Temperature setpoint adjustable from 50°C to 80°C
- Software-based PID Controller
- Real-time system monitoring on PC via serial communications from microcontroller
- Two 12-bit successive-approximation ADCs
- Protection: Flyback diode, Bulk and Bypass Capacitors ensure stability, longevity, and reduced electrical noise

1.2 Featured Applications

While this particular system was designed for temperature control, it could be modified to act as a controller from many other first-order systems.

Other potential applications for this system design style:

- Cruise control systems
- Motor control for an electric car
- Robotic arm or hand control
- LED light display control

Features of the System:

- Temperature setpoint adjustable from 50°C to 80°C
- Software-based PID Controller
- Real-time system monitoring on PC via serial communications from microcontroller
- Two 12-bit successive-approximation ADCs
- Protection: Flyback diode, Bulk and Bypass Capacitors ensure stability, longevity, and reduced electrical noise

1.3 Design Resources

The entire software implementation and all accompanying documentation is stored on GitHub and can be found at
https://github.com/sheppardd1/Controls_Final_Project

2 Key System Specifications

This temperature control problem was assigned with one particular specification held paramount: low steady-state error. The design was made such that the steady-state temperature was achieved within 1°C of the desired temperature. This goal was ultimately achieved using the K_p, K_i, and K_d values specified in Table 2. Due to the nature of this system the ultimate settling time, T_s, ended up being approximately one minute for the system to stabilize after a change in setpoint of the system's full range, that is, from 50 to 80°C or vice versa. Smaller changes to the setpoint resulted in much smaller settling times, typically in direct relation to the magnitude of the change in desired temperature.

PARAMETER	SPECIFICATIONS	DETAILS
PID Constants	$K_p = 1700, K_i = 1, K_d = 1800$	PID Constant Multipliers
MCU	MSP430F5529	Made by Texas Instruments
ADC	12-bit SAR	Reference voltage = 3.3 V
Temperature Range	50-80 °C	Arbitrarily assigned
Steady State Error	± 1°C	Maximum error allowed
Rise Time (T_r)	$\approx$ 60s	When going from 80–50 °C
Overshoot	$\approx 1 - 2\%$	Excess cooling or heating
Computational Delay	$\approx$ 8 ms	Time for MCU to perform computations and calculations

3 System Description

The overall design serves to connect the LM35 temperature sensor and the potentiometer to two separate analog-to-digital converters (ADCs) of the microprocessor and to connect the PWM output of the microprocessor to the power to the fan. The PWM control is established using a low side switch with a 2N7000 NMOS small-signal transistor. The gate of the NMOS is directly driven by the PWM output pin of the microprocessor and is also connected to a 43 kΩ resistor to ground to ensure that the gate is driven to ground when the microcontroller's PWM output is low.

3.1 Schematics and Implementation

The full details of the control system's physical implementation can be seen in Fig. 7. As the figure shows, the circuitry consists of three main connections to the MCU: the LM35 and potentiometer outputs to the ADC and the PWM signal generated by the MCU and set to the low-side switch. The general breadboard layout of the implementation can be seen in Fig. 2.

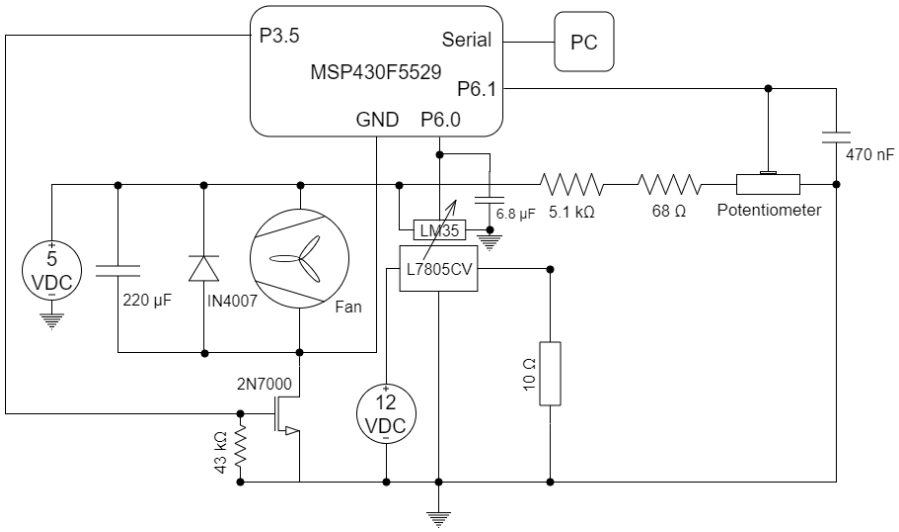

Figure 1: Schematic of the closed-loop system

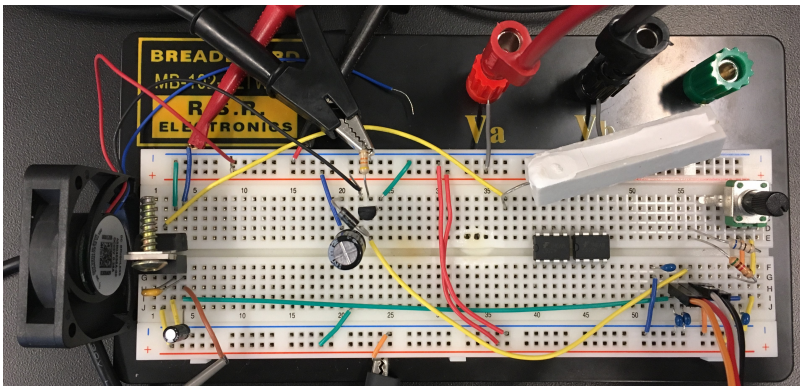

Figure 2: Photograph of the breadboard layout of all analog components.

3.2 Detailed Block Diagram

To demonstrate the overall functionality and layout of the control system, the block diagram seen in Fig. 3 was constructed. As this figure shows, the entire PID system is implemented as code in the MCU. The MCu accepts input from the LM35 and the potentiometer and it produces output that is sent to the fan.

3.2.1 General System Block Diagram

A more detailed outline of the system can be seen in Fig. 4. Here, the PID controller is further split into the separate sections corresponding to the proportional, integral,

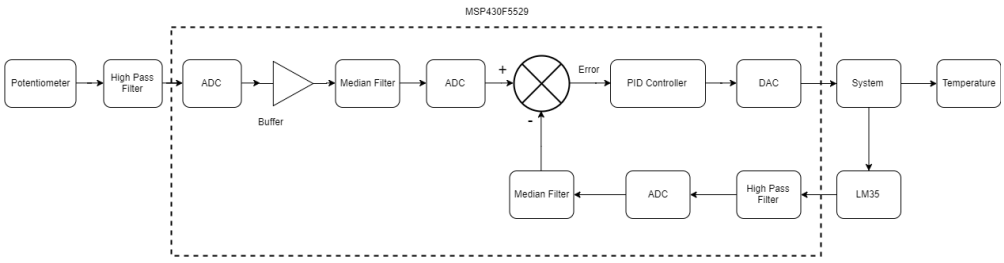

Figure 3: System Block Diagram

and derivative controls. The functions of each block are outlined in the following subsections.

3.2.2 Detailed System Block Diagram

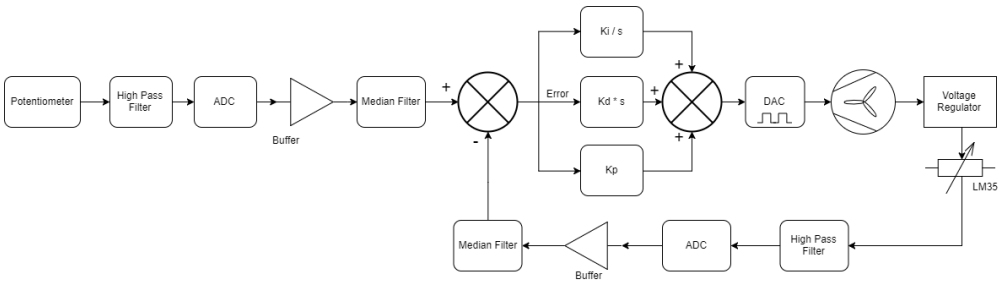

Figure 4: System Block Diagram

3.2.3 Potentiometer

The potentiometer acts as the input to the overall system. The potentiometer is connected such that its wiper sweeps the output between 3.3 V and 0 V, where 3.3 V corresponds to a temperature of 50°C and 0 V corresponds to a temperature of 80°C.

3.2.4 High-Pass Filter

Due to the fact that the system is subject to noise, a simple highpass filter was constructed with multiple 470 nF capacitors placed between the potentiometer output and ground. This helps to prevent high-frequency noise from the fan from interfering with the ADC in the MCU.

3.2.5 Analog to Digital Converter

The analog to digital converters were used to take the voltage reading from the LM35 and the potentiometer. The ADC on the MCU device that was used is 12 bits with a reference voltage of 3.3 V. Therefore, the readings from the ADC can be converted back into analog values by multiplying the ADC reading by $\frac{3.3}{2^{12}}$. The ADC reading from the potentiometer serves as the main input to the digital system.

3.2.6 Median Filter and Buffer

When both the potentiometer and LM35 voltages were read, it was decided that a median filter would be used to counteract any extraneous values that might be encountered. In order to do this, the ADC readings would enter a buffer (one buffer for the potentiometer and one for the LM35). Once a buffer was filled with three values, the median of those values would be determined as outputted from the filter as the true value of the voltage reading.

3.2.7 Negative Feedback Summing Junction

The negative feedback summing junction is the location at which the error signal (more on this later) is subtracted from the input signal. This negative feedback helps to keep the system dynamic and ensures that the true results mach with expected ones.

3.2.8 Proportional Control

The proportional control aspect of the system serves to add to or subtract from the system's response in a manner proportional to the error. This is the simplest of the PID control's components and is used to tune the system's overshoot to desired levels. This component is simply modeled as a constant in the s-domain.

3.2.9 Integral Control

The integrator of the system serves to reduce steady state error of the system. This is achieved by accumulating the error of the system over time using an integral. If an error develops that is not altered by the proportional or derivative controls, it gradually accumulates due to the integrator so that the integrator will alter the system and ultimately reduce steady-state error. The integral is modeled as $\frac{\mathrm{kp}}{s}$ in the s-domain.

3.2.10 Derivative

The derivative control of the system serves to alter the transient response of the system. This term helps to ensure that the system does not change too quickly when it approaches the necessary value. This helps to prevent oscillations as the system settles close to the desired value. This component is modeled as kd$s$ in the s-domain.

3.2.11 PID Summing Junction

The individual PID components are finally summed together in a summing junction so that the entire PID controller's output can be used to modify the system. Once the PID controller's output is determined, the PWM duty cycle is either increased or decreased by this value (multiplied by a scaling factor).

3.2.12 Digital to Analog Converter

Following the determination of the new PWM value, this new PWM signal is outputted to the fan using the MCU's built-in DAC. This marks the point at which the digital system ends and the analog system components resume. The process by which this is performed digitally is discussed in a later section.

3.2.13 Fan

The fan that was used for the project is manufactured by Delta electronics and was run off of a 5 V source. The fan is capable of handling up to 280 mA but in practice, only drew about 180 mA. The fan is ultimately the device that converts PWM into temperature.

3.3 Voltage Regulator and LM35 Temperature Sensor

The 5V voltage regulator was connected to a 12 V source so as to draw about 0.5 A of current in order to produce excessive amounts of heat. This device was attached to an LM35 temperature sensor which read the device's temperature. This temperature reading then went back through the ADC and into the MCU as a type of negative feedback.

3.4 Highlighted Devices

- MSP-EXP430F5529LP development kit from Texas Instruments
- 2N7000 small-signal nMOS transistor
- LM35DT temperature sensor IC
- L7805CV voltage regulator IC

3.4.1 Texas Instruments MSP-EXP430F5529LP development kit

The MSP-EXP430F5529LP is one of the more capable models in Texas Instruments' "LaunchPad" series of development and education kits. This device converts the control input from the potentiometer and the into the digital domain, processes those signals numerically, and converts its results into a PWM to control the fan's speed. Since this device was required for Introduction to Embedded Systems in the fall semester, it was already at hand and available for use. Since TI's Energia environment is mostly

Arduino-compatible, programming went from a tedious chore to a walk in the park. The device is both convenient and affordable at about $13.

3.4.2 2N7000 nMOS

The 2N7000 NMOS was chosen because it was readily available and fit the requirements. It can handle up to 200 mA (in practive the fan drew about 180 mA) and is power-efficient. It is an enhancement-mode N-type MOSFET and is RoHS-compliant. It serves as a low-side switch to control the pan using the MCU's PWM signal.

3.4.3 LM35DT

The LM35DT temperature senors is both resilient and easy to use. Its temperature-to-voltage relation is straightforward at 10 mV / °C and the device was easily attached to the voltage regulator by a small screw to get an accurate temperature reading.

3.4.4 L7805CV Voltage Regulator

The L7805CV voltage regulator was used for this project as well as in Embedded Systems. It is able to handle large amounts of current and was easily attached to the LM35 for an easy temperature reading.

4 System Design Theory

4.1 Open-Loop System Design

Without control by negative feedback, the temperature control system would simply be controlled by an approximated relationship between PWM duty cycle and temperature. An open-loop system like this would have to rely solely on previously determined data to set the PWM of the fan to the "optimal" value. This PWM would be constant for a given desired temperature. In other words, a desired temperature of 60°C might be achieved by a duty cycle of 50% and a temperature of 80°C might be realized with a duty cycle of 10 %.

To create an open-loop system, a relationship like this would have to be derived and a PWM value based solely on the desired temperature would have to be set whenever the desired temperature changed. This type of system would be very simple to implement, but very prone to error. Errors could arise from an open loop system like this because the conditions under which the temperature vs. PWM relationship would be derived would likely be different that those under which it would be tested. As such, external factor such as air temperature and distance between the fan and the heat source could create a steady-state error that could not be changed due to the lack of feedback.

While the system was not designed to be open-loop, an arbitrary limit was set for the desired temperature range such that each temperature could be achieved with

the given supplies. Through testing, it was seen that the fan could drop the heating source to a temperature of about 50°C at full power and that the system would settle at about 80°C with the fan at its minimum operating PWM value (more on this later). Therefore, it was determined that a range of 50-80°C could be obtained and maintained reasonably well with the supplies. These open loop properties were then used to develop a system that could be realistically constructed.

4.2 Initial Control Design

When setting up the original system, the control began as a simple proportional system; the integral and derivative components were then phased in to achieve optimal results. As previously discussed, the overall design consists of the communication between the circuit and the MCU via the potentiometer and LM35 voltage readings. The MCU was set up to act on these readings where the potentiometer acted as the primary input and the LM35 reading was taken into account as feedback. The difference between the input potentiometer signal and the normalized LM35 signal was taken into the PID controller as the error signal.

4.3 Iterative Steps

To begin developing a system with optimal results, the proportional component (kp) of the system was first tuned before adding the derivative constant(kd) and finally the integral constant (kp). The entire PID controller's transfer function T(s) can be modeled in the s-domain given the following equation:

$$\mathrm{T(s)} = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_{p^s} s + K_i}{s} \tag{1}$$

To begin the process, kd and ki were set to zero so that the proportional control constant kp could be altered to produce a response with an overshoot within the desired value of 1°C. While the value started off small, it immediately found that a relatively large value of kp would be needed to give the system both a quick settling time and reasonable overshoot. The final value for kp ended up being 1700.

One challenge that was faced while determining kp was discovering that the value needed to be much higher than we initially expected. For much lower values of kp, the system took far too long to alter the PWM of the fan. Not only did this lead to a slow response time, it also created excessive overshoot and oscillations because the system also took very long to settle at a PWM value that would maintain the desired temperature. This eventually led to the conclusion that kp would need to be placed at a value much higher than initially anticipated in order to attain a much more dynamic system.

Following the determination of an appropriate kp value, the value of kd was gradually changed until the system produced the desired transient response, i.e. one with an acceptable settling time with minimal oscillation. Like kp, kd also ended up as a

large value: 1800 to be exact.

Lastly, the value of ki was altered until the system produces a steady-state error that was within the bounds of $\pm 1^{\circ}$C. Since the system had virtually no steady-state error, a relatively small ki value was used. It was ultimately decided that a value of ki=1 best suited the application, as any value above ki=1 produced undesirable oscillations.

4.4 Final Design

Ultimately, it was determined that the proportional and derivative constants needed to be set to 1700 and 1800, respectively. These values allowed for the system to respond reasonably quickly, given the physical limitations of the system. One limitation that had to be overcome was the fact that the fan experienced a dead zone when being turned on. In other words, the idle fan could not be turned on with a PWM duty cycle of 5%. Due to this, it was determined that the fan should not be fully turned off because starting and stopping the fan would lead to unwanted temperature fluctuations which would lead to oscillations within the control system. Additionally, it did take considerable time for the fan to cool off the heating element, even when set at full power. As a result, the settling time as greatly limited by the physical properties of the system.

Overall, the system was able to drop the heat source from a temperature of 80°C to 50°C is about 60 seconds. This transition was viewed on the oscilloscope and can be seen in Fig. 5. This figure plots the voltage reading from the LM35 temperature sensor in relation to time. As the plots shows, the heat source cools at a relatively constant rate before finally settling at the desired temperature with a few minor corrections.

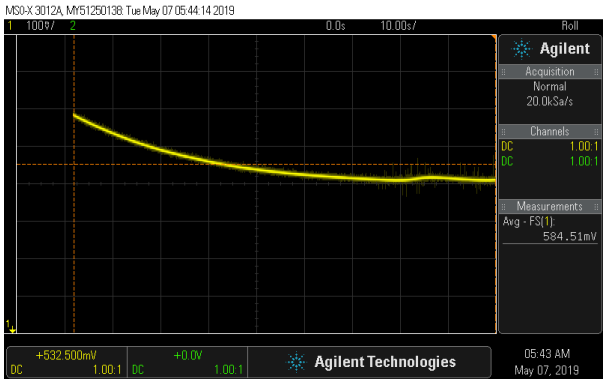

Figure 5: Visualization of the Temperature Falling from Maximum to Minimum Values in the Range

4.5 Second-Order Design Principles

To ensure optimal results when driving the fan, the fan was placed in parallel with a flyback diode and a bulk capacitor. The diode, which is reverse-biased, prevents unwanted changes in voltage and reduces strain on the fan while it is switching on and off. This safely shunts the high-voltage transient spikes that the fan's motor - an inductor - generates when the current through it is interrupted, in accordance with the relation $v_L = -L\frac{di_L}{dt}$. The capacitor helps to both reduce this strain and to reduce high-frequency noise caused by the fan. Additionally, multiple 470 nF ceramic capacitors were placed along the potentiometer, and a 6.8 μF electrolytic capacitor was connected between the LM35's output and ground, to act as analog low pass filters, shorting high frequencies to ground. Additionally, a median filter with a window size of 3 was designed in software to counteract any potentially skewed reading coming into the ADC.

5 Code Review

5.1 Initialization

The system begins by initializing all of the variables, most of which are initialized to zero so that errors can easily be caught. The vast majority of the variables in the program are declared globally so that they can be easily read via serial communications on the PC during runtime. The pins are assigned such that pins 6.0 and 6.1 are the ADC inputs of from the LM35 and potentiometer and pin 3.5 is the digital PWM output from the microcontroller. For testing purposes, pin 2.7 was toggled at the beginning and end of the system's loop to determine computation time. The computation time was determined to be 8 ms based on the fact that it took 308 ms to the entire loop (lines 55-91) to run including the three 100 ms delays. This was determined by monitoring P2.7 on the oscilloscope, as seen in Fig. 6.

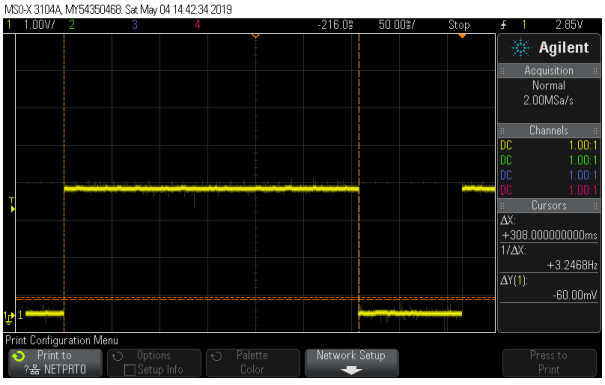

Figure 6: Visualizing the Computational Delay with the Oscilloscope

The variables representing the derivative, integral, error, summing junction, potentiometer reading, and LM35 reading were all initialized to 0. Since these values were often viewed on the PC using serial communications, the initialization to zero allowed for errors (such as a lack of proper variable assignment in the code) to be easily caught. An array of three values for the potentiometer and LM35 readings was also set up and initialized to zero. Furthermore, the sampling period of the device was set to be 102.7 ms to account for computational delay (100ms intentional delay + $\frac{\text{8ms delay}}{\text{3samples per loop}} \approx$ 102.7ms). The initial PWM value was set to 150 (on a scale of 0-255).

To improve computational speed, the value of $\frac{kd}{\Delta t}$ was computed and stored as a constant (kdt). This improves computation time because the derivative of the error would be calculated as $\frac{V_1 - V_0}{\Delta t} \times k_d$. Since $\frac{kd}{\Delta t}$ is a constant (assuming consistent computation times), it is much more efficient to perform this division once instead of every loop iteration. As a result, the value of $V_1 - V_0$ (the change in LM35 voltage readings) can simply be multiplied by the constant kdt each time.

As similar computation was performed to produce a variable used for converting between the digital voltage reading and the original analog voltage read from the LM35 and the potentiometer. Since the MCU's ADC is 12 bits with a reference of 3.3 V, the number of volts per bit can be calculated by dividing $\frac{3.3}{2^{12}}$. This value was also set as a constant to be used during the conversion.

Finally, an equation to represent the relationship between the potentiometer voltage and the desired LM35 voltage was determined as:

$$\text{desired LM35 voltage} = -\frac{0.3}{3.3} \times \text{potentiometer voltage} + 0.8 \tag{2}$$

since the range of temperature requests in 50-80°C. In a similar manner to setting kdt and the volts per bit value (mentioned previously), the slope $\frac{0.3}{3.3}$ was also calculated and set as a constant. Additionally, a PWM scale of 2.55 was set to scale the PID output such that the response would be normalized, i.e. the output of the PID would correspond to change in duty cycle and the scale would convert duty cycle to PWM value (0-255).

The outputs were initialized using basic syntax found in Energia example code online [2]. The setup syntax is given as:

```
void setup() {
  // put your setup code here, to run once:
  pinMode(pwm_out, OUTPUT);        // sets the pin as output
  //pinMode(toggle_pin, OUTPUT);  // usedfor determining computation time
   Serial.begin(9600);             // setup serial
}
```

Here, the pwm output pin (3.5) is declared as an output (as well as 2.7 during testing). Likewise, the serial communication system is set up between the PC and the MCU. This allows for desired variables to be read in the terminal. The baud rate of the serial communications is set to 9600, which must be specified in the terminal used to read

the values. The program used for testing was RealTerm.

5.2 Sensor Acquisition

The sensors were read in the program's main loop three times in a row before acting on their values. To ensure that outliers were not taken into the PID controller, a median filter was developed to only output the median of the three values into the PID control system. The sensor values were read in accordance with sample code found in Energia's reference documentation [2].

```
for(int i = 0; i < 3; ++i){
    ptat_array[i] = analogRead(lm35_pin); // read 3 ptat values
    pot_array[i] = analogRead(pot_pin);   // read 3 pot values
    delay(sampling_delay);                // wait
  }
```

5.2.1 Signal Conditioning

To prevent erroneous values from giving incorrect results, the voltage readings from both the LM35 and the potentiometer were median filtered. This was done using a median filter function that was designed for a window with a width of three items. The function checks each element in the array to determine if that element's value is numerically between the other two. While the code is not easy to read, it is relatively efficient. The function is written as:

```
float medianFilter(float a[]){
  if((a[0] <= a[1] && a[0] >= a[2]) || (a[0] >= a[1] && a[0] <= a[2]))      //if a0 is
     between a1 and a2
       return a[0];
  else if((a[1] <= a[0] && a[1] >= a[2]) || (a[1] >= a[0] && a[1] <= a[2])) //if a1 is
     between a0 and a2
       return a[1];
  else                                                                      // if a2 is
     between a0 and a1
       return a[2];
}
The function returns the value that is numerically in the middle. This value is then sent
      into the PID control function.
```

5.3 Error Calculation

The error signal is calculated in a rather simple manner. First, the previous LM35 analog voltage reading is stored in a variable called old_ptat_voltage (for use in the derivative calculation). Then, the new LM35 analog reading is calculated using the digital value along with the volts per bit number derived earlier. The equation for this is given as:

$$V_{analog} = V_{digital} \times \frac{3.3}{2^{12}} \tag{3}$$

For the sake of computation speed, the value of $\frac{3.3}{2^{12}}$ was declared as a constant during initialization, as previously discussed. The error is then calculated by subtracting the

new analog voltage reading from the desired analog voltage. Based on this calculation, if the current temperature is too high, then the error value will positive and will produce an increase in fan speed. If the new temperature is too low, the the error will be negative, resulting in a lower fan speed.

5.4 Control Calculation

Following the calculation of the error value, the PID function is called. This function determines the derivative and integral of the changing error. The derivative is given by $\frac{V_1 - V_0}{\Delta t}$ and is then multiplied by the scaling factor kd. As previously discussed, the value of $\frac{kd}{\Delta t}$ is initialized at the start of runtime to decrease computation time. To prevent the derivative value from spiking at initialization (when old_ptat_voltage is 0), the derivative is set to zero after the first run because old_ptat_votlage has not been given a real nonzeros value yet. This is accomplished as follows:

```
bool first_time = true; //initialization at runtime
// ...
if(!first_time){
    derivative = (kdt * (ptat_voltage - old_ptat_voltage));
  }
  else{
    derivative = 0;
    first_time = false;
  }
```

The integral of the error is approximated using Riemann sums. This is accomplished by multiplying the change in time by the error value to approximate the area under the error curve. To ensure that the integral does not unnecessarily accumulate when the fan is at its maximum speed, the integral is not calculated when the PWM value is set to the maximum of 255. If this were not accounted for, the system would take a long time to react to a sudden change in temperature or desired temperature. The integral calculation is given as:

```
  if(pwm_val != 255){
  //ensure integral does not accumulate if system is alreasy at max PWM
//using Reimann sums, and multiplying period by 3 since using median filter
    integral = ki * (integral + (error * adc_sampling_period*3));
  }
```

The final output of the PID controller is determined by summing the integral, derivative, and proportional values together after they have been multiplied by the error. (The proportion value is simply error $\times$ kp. The output of the PID controller is then multiplied by the scaling factor of 2.55 to determine the amount by which the PWM value will increase or decrease.

The value of ki, kp, and kd were determined through a trial and error method. first, the values of ki and kd were set to zeros while the vvalue of kp was adjusted to provide until the system experienced and overshoot of less than or equal to 1 °C. Then, the derivative constant kd was adjusted until the system's oscillation was kept to a minimum. Finally, ki was slightly increased until the steady-state error of the system was under control. Ultimately, the values that were used for the system were ki=1, kd=1800 and kp = 1700. The value of ki is very low because the system behaved

very well even without an integral component. The high values of kd and kp were needed because the system took very long to response and experienced significant oscillation when their values were much lower.

5.5 Actuation

Finally, the system was able to act on the results given by the PID controller. The new PWM value was set by multiplying the PID control output to a scaling factor of 2.55 and adding the result to the old PWM value. A positive error value would correspond to a temperature that was over the desired value leading to an increase in the PWM duty cycle and vice versa. In order to prevent excessive oscillations, it was determined that the fan should never be turned off because turning the fan on requires that a large deadzone be overcome which leads to oscillations. In other words, if the fan is off, it cannot be turned on by applying a PWM signal with a 5% duty cycle. As a result, the PWM value was never allowed to go below 30 (or a $\approx$ 12% duty cycle). Additionally, the PWM can never rise above 255, so this must be accounted for as well. This implementation is given as follows:

```
int new_pwm = pwm_val + sum * pwm_scale;
// new_ccr must be an int, so decimals purposely get truncated;
// using scale = 2.55 since pwm_val goes from 0 to 255, so 2.55 normalizes it
// account for out of bounds values
if(new_pwm > 255)
  return 255;
else if (new_pwm < 30)
// don't want fan to turn off, because turning completely off and on causes bad
   oscillations
  return 30;
else
  return new_pwm;
```

6 Design Files

6.1 Schematics

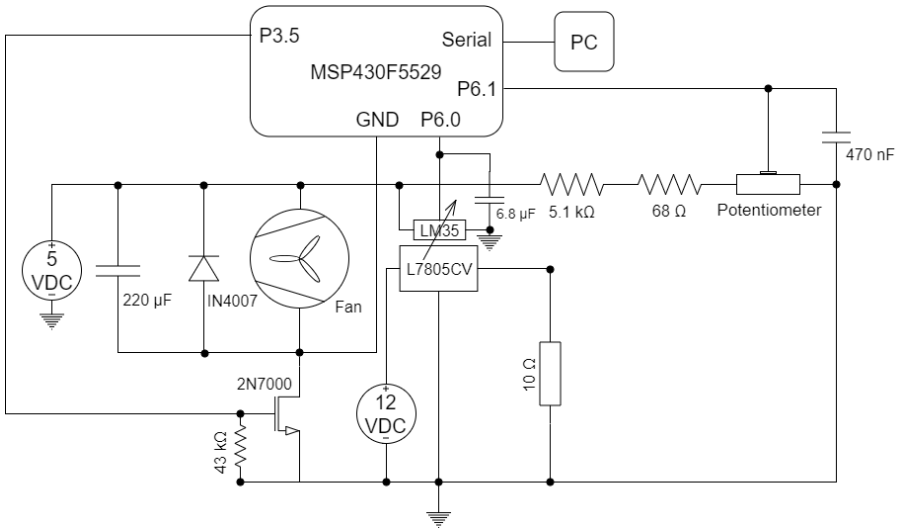

Figure 7: Schematic of the closed-loop system

6.2 Bill of Materials

Table 1: Bill of Materials

Item	Cost
TI MSP-EXP430F5529LP	$12.99
Delta Electronics fan	$12.08
LM35DT temperature sensor	$2.90
L7805CV 5V voltage regulator	$1.70
220 μF electrolytic capacitor	$0.47
470 nF ceramic capacitors (4)	$1.20
6.8 μF electrolytic capacitor	$0.47
1N4007 rectifier diode	$0.19
10 Ω, 10 W power resistor	$1.25
5.1 kΩ, 68 Ω, and 43 kΩ resistors	$0.05 ea. $\times 3$
10 kΩ potentiometer (linear taper)	$0.76
2N7000 NMOS transistor	$0.42

6.3 Complete Code Listing

```
// Ports
int lm35_pin = P6_0;              // connect PTAT output to A0 (6.0 on MSP430F5529LP)
int pot_pin = P6_1;               // connect potentiometer to A1 (6.1)
int pwm_out = P3_5;
//int toggle_pin = P2_7;          // used for determining computation time

// PID Variables
const float kp = 1700;            // proportional constant
const float ki = 1;               // integral constant
const float kd = 1800;            // derivative constant
float integral = 0;               // integral value that accumulates over time
float derivative = 0;             // derivative value (past reading vs. current)
float error = 0;                  // real - desired ptat voltage
float sum = 0;
// result of summing juction (integral + derivative + protortion)

// ADC Readings and constants
const float volts_per_bit = 3.3 / 4096;
// reference is 3.3 V and ADC is 12 bits (2^12 = 4096)
float pot_array [3] = {0,0,0};    // 3 readings from pot
float pot = 0;                    // median filtered pot readings
float ptat_array [3] = {0,0,0};   // 3 readings from ptat
float ptat = 0;                   // median fitlered ptat value
const int sampling_delay = 100;   // delay between taking samples in ms
const float adc_sampling_period = 0.0027 + (sampling_delay * 0.001);
// rate at which ADC samples

// Calculated from ADC
float ptat_voltage = 0;           // voltage read from ptat
float old_ptat_voltage = 0;       // previous votlage read from ptat
float desired_ptat_voltage = 0.5; // deisred ptat votlage
float pot_voltage = 0;            // votlage read from pot

// Parameters
int pwm_val = 150;                // default to 150, value can be from 0 to 255
float kdt = kd / adc_sampling_period;
// calculating this now instead of recalculating makes derivative calculation faster
// derivative = [(new votlage - old votlage) / time] * kd
// Remove (kd / time) from equation since it's const so calculation is only done once
bool first_time = true;
// used to ensure derivative is calculated corectly on first run
const float slope = -0.3 / 3.3;
// equation: desired_ptat_voltage = slope * pot_voltage + 0.8
const float pwm_scale = 2.55;
// scale for pwm change; normalizes calculated pwm to 0 - 100% duty cycle


void setup() {
// put your setup code here, to run once:
pinMode(pwm_out, OUTPUT);       // sets the pin as output
//pinMode(toggle_pin, OUTPUT);  // usedfor determining computation time
Serial.begin(9600);             // setup serial
}

void loop() {

//digitalWrite(toggle_pin, HIGH);

// read the value from the sensor ************************************
for(int i = 0; i < 3; ++i){
ptat_array[i] = analogRead(lm35_pin); // read 3 ptat values
pot_array[i] = analogRead(pot_pin);   // read 3 pot values
delay(sampling_delay);                // wait
}

```

```
// apply median fitler to readings ************************************
ptat = medianFilter(ptat_array);
pot = medianFilter(pot_array);

// calcuate voltages and call PID controller to get new PWM value ******
setPWM();

// set PWM value ******************************************************
analogWrite(pwm_out, pwm_val);
//digitalWrite(toggle_pin, LOW);  // used for determining computation time
//delay(100);                     // used for determining computation time

//display readings and calculated values on terminal ******************
Serial.print("Temp: ");
Serial.println(ptat_voltage);
Serial.print("Pot: ");
Serial.println(pot_voltage);
//Serial.print("Derivative: ");
//Serial.println(derivative);
//Serial.print("Integral: ");
//Serial.println(integral);
//Serial.print("Error: ");
//Serial.println(error);
//Serial.print("Sum: ");
//Serial.println(sum);
}

float medianFilter(float a[]){
// ONLY WORKS IF WINDOW OF FILTER IS 3 NUMBERS WIDE
// The methodology is not easy to read, but is efficient
// Checks to see which of the values is between the other two (this is median)

if((a[0] <= a[1] && a[0] >= a[2]) || (a[0] >= a[1] && a[0] <= a[2]))
//if a0 is between a1 and a2
return a[0];
else if((a[1] <= a[0] && a[1] >= a[2]) || (a[1] >= a[0] && a[1] <= a[2]))
//if a1 is between a0 and a2
return a[1];
else
// if a2 is between a0 and a1
return a[2];
}


// set PWM values to change duty Cycle of PWM: Low DC = slow fan, High DC = fast fan
// PWM values range from 0 (off) to 255 (100% DC)
void setPWM()
{
// using the LM35 PTAT
// PTAT equation: Vo = 10 mV/C + 0 mV
// therefore, Temperature = Vo / 0.01
// however, system is based on voltage readings, not temperature

//calculations*********************************************************************
old_ptat_voltage = ptat_voltage;          // store previous ptat voltage
/* Notes on ADC:
*  reference PTAT voltage is 3.3 V
*  ADC is 12 bits (2^12 = 4096)
*  3.3 / 4096 ~= 0.000805664063 = volts_per_bit
*/
ptat_voltage = ptat * (volts_per_bit);
// convert digital ptat reading back to analog voltage value
pot_voltage = pot * (volts_per_bit);
// convert digital pot reading back to analog voltage value
desired_ptat_voltage = slope * pot_voltage + 0.8;
// use equation to find desired voltage

// if ptat_votlage > desired, then temp is too high,
```

```
// so increase duty cycle with positive error val
// if ptat_voltage < desired, then temp is too low,
// so decrease duty cycle with negative error val
error = ptat_voltage - desired_ptat_voltage;

//use PID controller ***************************************************************
pwm_val = PID(error);
}

float PID(float error){
// Derivative ***********************************************************************
// If system has just started (first_time == true), old_ptat_voltage will be 0,
// and would be giving incorrect derivative
if(!first_time){
derivative = (kdt * (ptat_voltage - old_ptat_voltage));
}
else{
derivative = 0;
first_time = false;
}

//Integral *************************************************************************
if(pwm_val != 255){
//ensure integral does not accumulate if system is already at max PWM
//using Reimann sums, and multiplying period by 3 since using median filter
integral = ki * (integral + (error * adc_sampling_period*3));
}

//Proportional *********************************************************************
float proportion = kp;

// summing junction ****************************************************************
sum = (derivative + proportion) * error + integral;

int new_pwm = pwm_val + sum * pwm_scale;
// new_ccr must be an int, so decimals purposely get truncated;
// using scale = 2.55 since pwm_val goes from 0 to 255, so 2.55 normalizes it
// account for out of bounds values
if(new_pwm > 255)
return 255;
else if (new_pwm < 30)
// don't want fan to turn off, because turingin ti completely off and on causes
// bad oscillations
return 30;
else
return new_pwm;
}
```

References

[1] Texas Instruments, Inc., "LM35 Precision Centigrade Temperature Sensors" LM35 datasheet, Nov. 1997 [Revised Dec. 2017].

[2] Energia, "analogRead()" *energia.nu*, [Online]. Available: https://energia.nu/reference/en/language/functions/analog-io/analogread/ [Accessed May 1, 2019].