University of The West Indies, Cave Hill Campus

Department of Computer Science, Math and Physics

End of Semester Project

PID-CONTROLLED FAN SYSTEM FOR TEMPERATURE REGULATION IN A CHICKEN COOP

Date of Submission: 30th November 2025

Control Systems-**ELET 3220**

ABSTRACT

This project presents the design and implementation of a PID-controlled fan system for temperature regulation in a chicken coop environment. Temperature measurements were obtained using an NTC thermistor, calibrated against a reference laboratory-grade glass thermometer, and processed through an Arduino MEGA 2560 microcontroller. A 4-wire PWM fan was used to regulate airflow, with fan speed adjusted according to the PID control algorithm. Experimental testing demonstrated that the thermistor provided accurate measurements, and the PID controller effectively maintained the setpoint temperature of 24°C. The system responded dynamically to temperature changes, with the fan PWM increasing during heating and decreasing as the environment cooled. Overall, the project demonstrates a reliable and practical solution for automated temperature regulation in small-scale poultry enclosures.

TABLE OF CONTENTS

INTRODUCTION

Temperature regulation for chicken coops is critical for the health and performance of chickens. Poultry are highly sensitive to temperature fluctuations and excessive heat can lead to heat stress, reduced feed intake, and lower egg production, while low temperatures can slow growth and increase energy consumption as the birds attempt to stay warm. Ensuring a stable environment is therefore essential for both chicken welfare and efficient farm management. Traditional ventilation systems in small-scale or low-cost poultry farms often rely on manual fan operation or simple on/off fan systems. These approaches are reactive systems and can result in temperature swings and delayed responses to environmental changes.

To address these challenges, this project develops a PID-controlled fan system designed to maintain a consistent temperature using real-time temperature readings. By continuously adjusting fan speed based on temperature, accumulated error, and the rate of temperature change, the PID controller offers smoother, more accurate regulation compared to conventional methods.

Key Objectives:

- To implement a PID (Proportional, Integral, Derivative) controller and observe how it affects temperature regulation.
- Evaluate how the system responds to increases in temperature.
- Assess the recovery time required for temperature to return to setpoint.
- Observe how steady the temperature stays once it is at setpoint.

THEORETICAL REVIEW

Thermistor Theory

The thermistor that was used for this project was a 10k NTC thermistor. A NTC thermistor is a temperature-sensitive resistor whose resistance decreases exponentially as its temperature increases. As temperature rises, more electrons gain energy to move into the conduction band, reducing the material's overall resistance. This thermistor is 10k ohms and this means that at this resistance value the standard reference temperature is at 25°C. The thermistor has a characteristic value of B (BETA), and this BETA value describes how a thermistor's resistance changes with temperature. Because thermistors are non-linear, their resistance does not increase or decrease in a straight line as temperature changes. The Beta value allows this non-linear behaviour to be mathematically modelled so the microcontroller can convert the measured resistance into an accurate temperature.

A thermistor is used as part of a voltage divider, where its resistance varies with temperature. A fixed supply voltage is applied across a series circuit with a 10k resistor and the thermistor. As the thermistor's resistance changes, the voltage across it changes proportionally. This voltage can be measured and converted to temperature using the Beta or Steinhart-Hart equation. For this project the BETA formula was used. Using the Beta equation, the resistance calculated from the voltage divider is compared to the thermistor's known resistance at a reference temperature which is usually 25°C, and Beta determines how steep the resistance changes. This relationship allows the system to accurately calculate the actual temperature from the analog voltage reading.

$$\beta = \frac{\ln ln \left(\frac{R1}{R2}\right)}{\left(\frac{1}{T1}\right) - \left(\frac{1}{T2}\right)}$$

,

Where:

- R1 and R2 are thermistor resistance values at T1 and T2.
- T1 and T2 are temperature values in Kelvin.

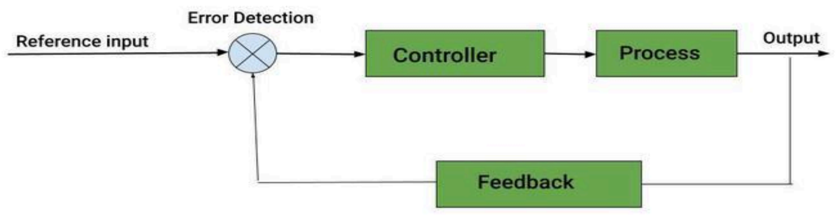
Temperature Control Theory

Temperature control is crucial for chicken coops in Barbados primarily to prevent heat stress, which can lead to severe health problems, decreased productivity (like egg production and growth rates), and even mortality in poultry. Chickens are highly sensitive to environmental temperatures and have difficulty regulating their body heat, especially in hot and humid conditions like those found in Barbados. According to Tpi-Polytechniek, poultry regulate their internal temperatures primarily through mechanisms such as convection, conduction, radiation, and evaporative cooling. However, when temperatures exceed 24°C, poultry primarily rely on evaporative cooling through panting. High humidity in tropical climates impedes evaporative cooling, reducing the effectiveness of this cooling mechanism. Farmers in Barbados use ventilation as a means of circulating air and regulating temperature by using natural ventilation and mechanical ventilation which uses fans and exhaust systems.

Thermal environments such as an enclosed coop are subject to unpredictable and constantly changing conditions, which makes open-loop control systems unreliable for temperature regulation. In an open-loop system, the fan speed or heater power is set manually or according to a predetermined schedule, without receiving any feedback from the actual temperature inside the enclosure. This means the system cannot detect when the temperature rises too high, cools too slowly, or deviates from expected behavior due to disturbances such as sudden heat load changes, variations in ambient temperature, or differences in airflow.

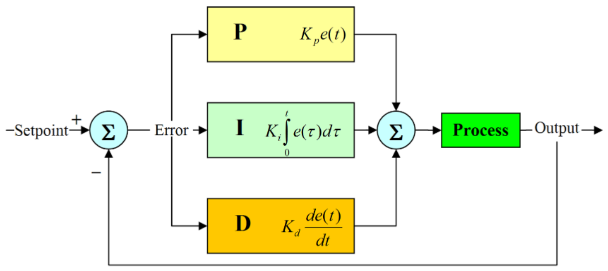
In addition, since the enclosure behaves like a first-order thermal system with a long time constant, its temperature responds slowly to changes in fan speed and heating. Small disturbances can accumulate over time, and an open-loop system has no mechanism to correct or compensate for unexpected conditions. As a result, the temperature may overshoot, remain above target, or fluctuate unpredictably. A closed-loop system however, continuously measures the internal temperature and adjusts fan speed automatically to maintain the desired setpoint. This feedback allows the controller to correct errors immediately when the temperature drifts away from the target, compensate for disturbances such as increased heat load or changing ambient conditions, reduce overshoot and oscillation by adjusting the output gradually and maintain a stable temperature despite slow thermal dynamics.

Overall, closed-loop control is better suited for thermal systems because it provides precision, stability, disturbance rejection, and adaptability, all of which are essential for maintaining consistent temperature in an environment with slow and nonlinear dynamics.



PID Control Theory

A PID (proportional, integral, derivative) controller is a feedback control mechanism that uses three terms to continuously calculate an output that minimises the error between a desired setpoint and a measured process variable. It works by summing a proportional response to current error, an integral response to accumulated errors, and a derivative response to future predicted error.



$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

,

Where:

- E(t) is the computed error (e(t) = measured_temperature – setpoint)

Parameter Roles

- Proportional (Kp): reacts to the current error. If the current error is large, then Kp will increase fan speed and if error is small then Kp will reduce the speed of the fan.

$$u(t) = K_p e(t)$$

- Integral (Ki): reacts to the accumulated error over time and removes any steady-state error.

$$u(t) = K_i \int_0^t e(t)dt$$

- Derivative (Kd): reacts to the rate of change of temperature and removes any overshoot.

$$u(t) = K_d \frac{de(t)}{dt}$$

METHODOLOGY

Components Used

Hardware:

- Arduino MEGA2560 microcontroller
- 16x2 LCD
- 10k Potentiometer
- Breadboard
- 10k N.T.C Thermistor
- 12V 4-wire PWM Brushless DC fan
- Back-UPS 12V 6.5A battery
- Lollipop sticks
- Polyethylene film
- Cardboard
- Glue gun and glue sticks
- Jumper Wires
- 100nf Ceramic Capacitor
- 2, 10k Resistors
- Multimeter

- Heating element

Software:

- Circuit Diagram
- Canva
- Arduino IDE

Block Diagram

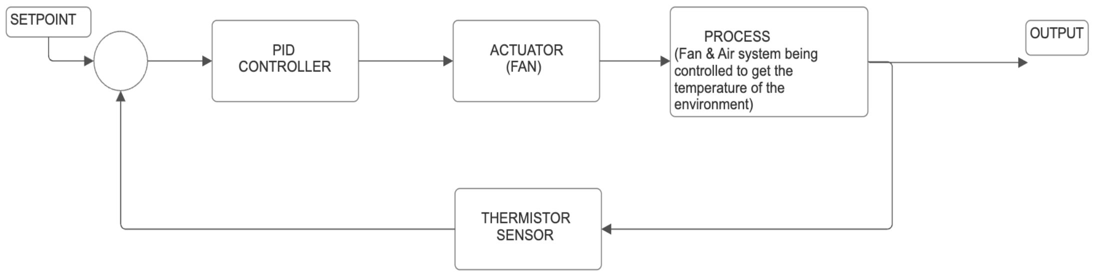

SETPOINT
PID CONTROLLER
ACTUATOR (FAN)
PROCESS (Fan & Air system being controlled to get the temperature of the environment)
OUTPUT
THERMISTOR SENSOR

Figure 1: Closed-loop system block diagram

Schematic

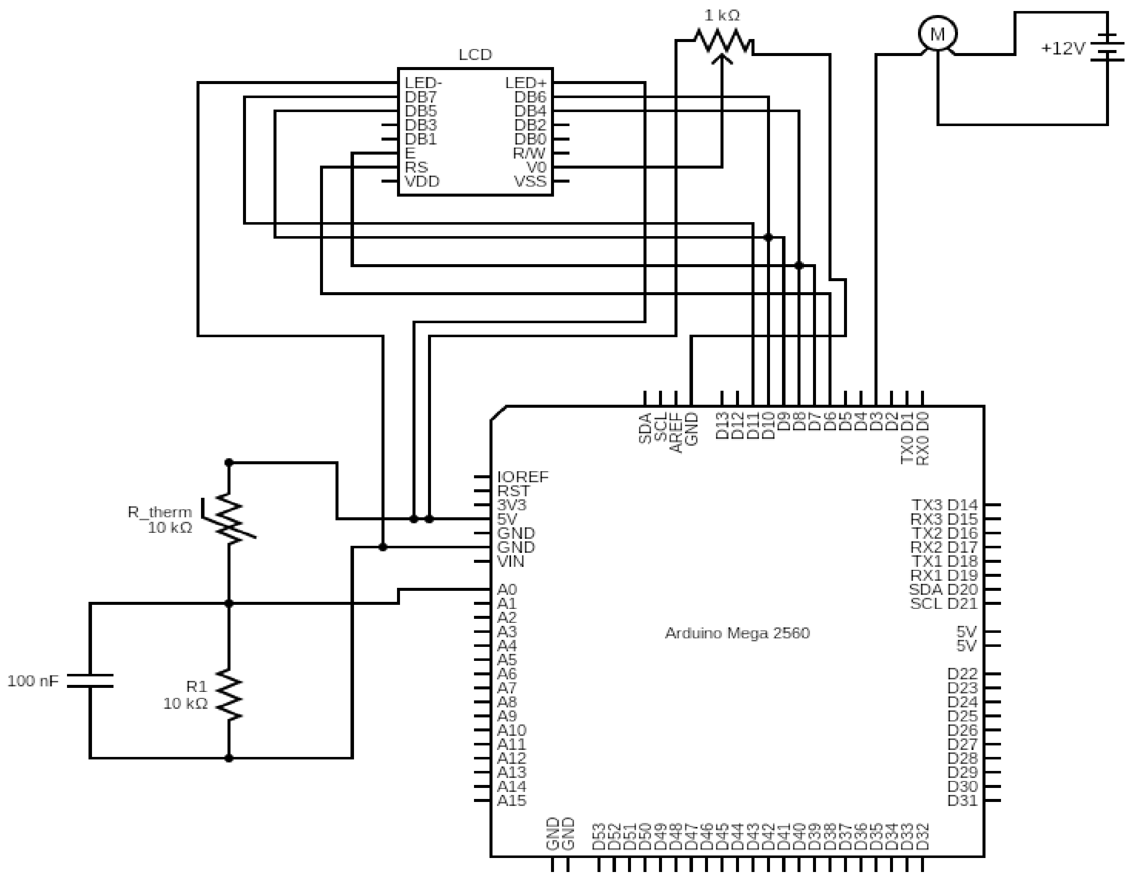


Figure 2: Schematic of the system

Physical placement

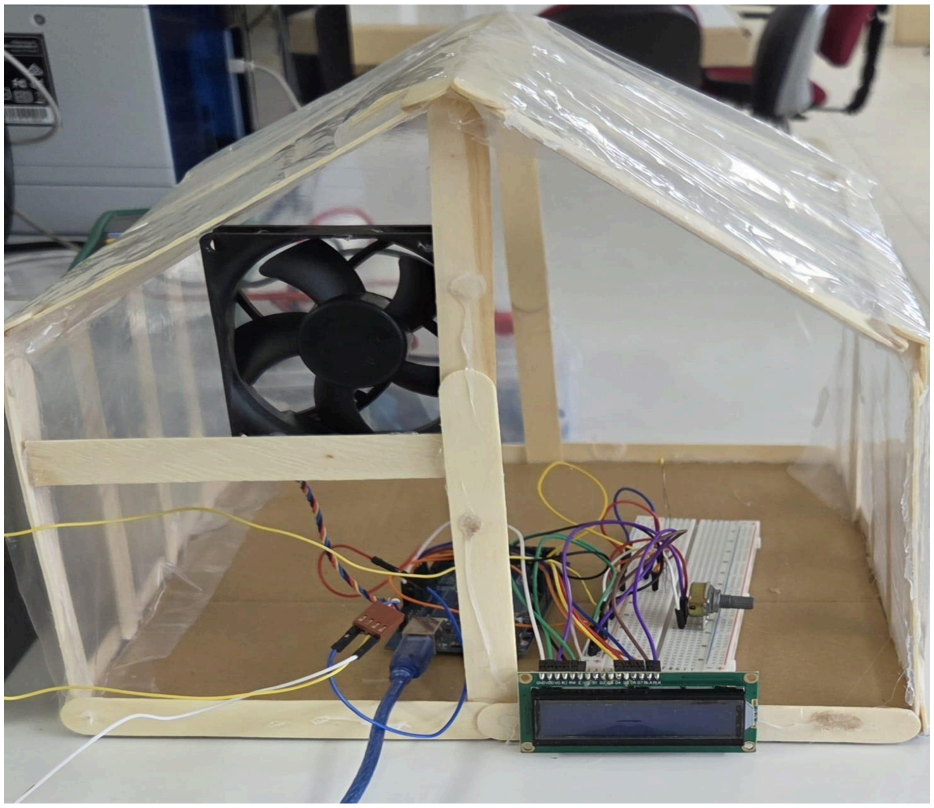


Figure 3: Physical placement of the fan (top) and thermistor (near centre)

Figure 3 above shows the placement of the components. The 12V DC Fan is placed high within the test enclosure to simulate realistic chicken-coop ventilation, where hot air naturally rises due to convection. The thermistor is placed in open air a little from the middle relative to the heat source, ensuring that it measures the true ambient temperature and was not directly affected by the airflow from the fan.

Environmental Conditions

A serialized glass thermometer (laboratory-grade) was used as the reference instrument for temperature calibration. All measurements were performed indoors at a room temperature of approximately 23° C. The Arduino sampled the analog input at Hz, applied the calibration equation, and displayed both the real-time temperature and PWM fan output on the LCD.

Control algorithm & Implementation

The fan control algorithm was implemented in Arduino C++ using the Arduino IDE. Temperature readings were sampled from the NTC thermistor connected in a voltage-divider configuration. To reduce measurement noise, each temperature was computed from the mean of 20 ADC samples spaced 10ms apart.

The analog reading (0-1023) was converted to voltage using the measured supply voltage, 5.04V, and the thermistor resistance was computed using the voltage-divider equation:

$$R_{therm} = R_1 \cdot \left(\frac{VCC}{voltage} - 1.0\right)$$

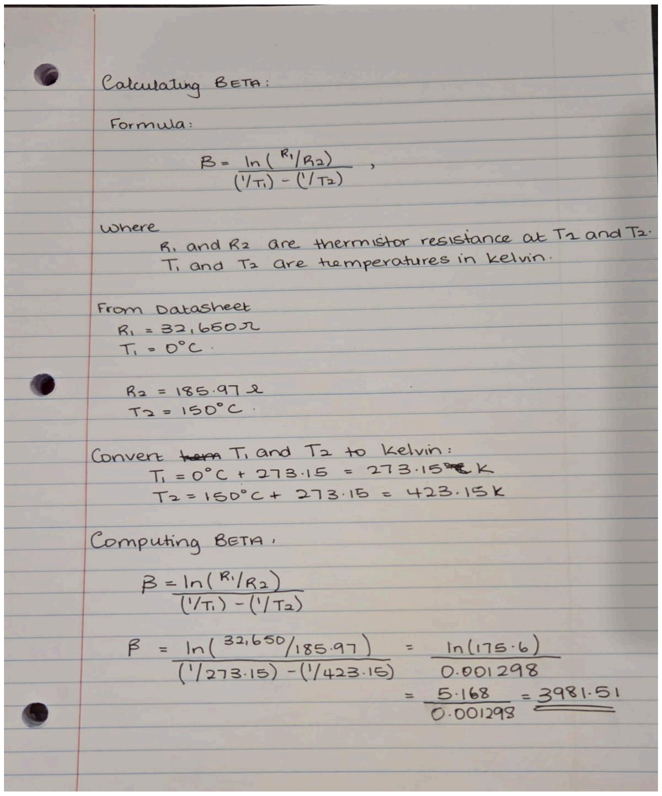
,

Where:

- VCC is the supply voltage from the Arduino
- Voltage is the digital voltage values; analog voltage from voltage divider is converted to digital voltage using the ADC conversion

Temperature in °C was then obtained using the Beta equation, where the measured resistor values (fixed resistor = 9960 ohms and thermistor value = 10910 ohms) and the calibrated Beta value β = 3981.51 were used.

Calculated BETA value



Calculating BETA:

Formula:

$$\beta = \frac{\ln(R_1/R_2)}{(1/T_1) - (1/T_2)},$$

where

R_1 and R_2 are thermistor resistance at T_1 and T_2.
T_1 and T_2 are temperatures in kelvin.

From Datasheet

$R_1 = 32{,}650\Omega$
$T_1 = 0°C$.

$R_2 = 185.97\Omega$
$T_2 = 150°C$.

Convert ~~term~~ T_1 and T_2 to kelvin:

$T_1 = 0°C + 273.15 = 273.15\,K$
$T_2 = 150°C + 273.15 = 423.15\,K$

Computing BETA,

$$\beta = \frac{\ln(R_1/R_2)}{(1/T_1) - (1/T_2)}$$

$$\beta = \frac{\ln(32{,}650/185.97)}{(1/273.15) - (1/423.15)} = \frac{\ln(175.6)}{0.001298} = \frac{5.168}{0.001298} = \underline{\underline{3981.51}}$$

Figure 4: Calculation of BETA

The PID control loop executed every iteration using the elapsed time Δt measured from millis(). The method used to find the values was a manual Ziegler-Nichols-style tuning.
The proportional, integral, and derivative gains used were:

- Kp = 10.0
- Ki = 0.0
- Kd = 0.1

The integral term accumulated error over time with an anti-windup limit of ± 300 to prevent runway output. The derivative term was computed using the difference between the current and previous error divided by Δt.

Experimental Procedure

The test enclosure was set up, and the thermistor and fan were placed as shown in figure 3. A controllable heat source (soldering iron) was placed at a fixed distance to simulate rising temperature inside a chicken coop. The initial temperature was recorded by both using the thermistor and the reference glass thermometer. The heat source was turned on, and the temperature was allowed to increase. As the temperature rose above the 24°C setpoint, the PID controller adjusted the PWM duty cycle to increase the fan speed. The system's response to the rise in temperature, activation of the fan, and the progression toward the setpoint were observed.

Once the temperature passed the setpoint, the PID controller was allowed to regulate the temperature back toward the setpoint. The time taken for temperature to stabilize near the setpoint was noted and the heat source was turned off and cooling behaviour was observed. The entire process was repeated three times to check consistency and the temperature and PWM readings were recorded from the LCD display and recorded manually into Microsoft Excel for plotting.

RESULTS

Sensor Calibration

A comparison was conducted between the thermistor readings and the serialized glass thermometer to evaluate the accuracy of the calibrated Beta value. Table 1 presents the measured temperatures and the corresponding error calculated using:

$$Error = thermistor_{temp} - thermometer_{temp}$$

Glass thermometer vs thermistor

Thermistor (°C)	Glass Thermometer (°C)	$Error = thermistor_{temp} - thermomete$
24	23	1°
23.8	23	0.8°
23.7	23	0.7°
23.6	24	-0.4°
23.5	24	-0.5°
23.4	24	-0.6°
23.3	24	-0.7°
23.3	24	-0.7°
23.3	24	-0.7°
23.3	24	-0.7°
23.3	23	0.3°

Table 1: Table with temperature data from thermistor and serialised glass thermometer

Average error = 1 + 0.8+ 0.7 -0.4 - 0.5 - 0.6 - 0.7 - 0.7 - 0.7 - 0.7 +0.3 = -1.5/11 = -0.14°C

Maximum error = 1°C

Minimum error = -0.7°C

These results indicate that the thermistor calibration is accurate, demonstrating only slight deviation from the reference thermometer and conforming suitability for use in the PID control system.

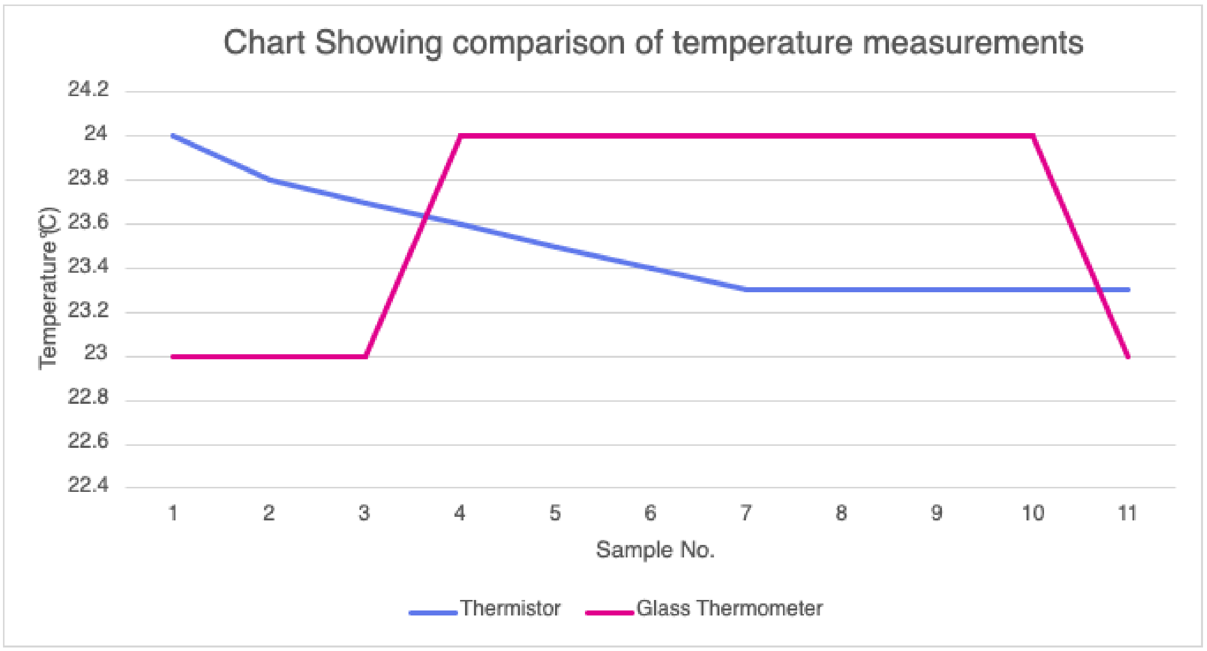


Figure 5: Chart comparing the readings from the thermistor vs the serialised glass thermometer

As shown in figure 5, the thermistor closely matched the reference glass thermometer readings across all observed points.

System Response to Heating

Table 2 below summarises the temperature readings obtained from the thermistor during the heating and cooling cycles, along with the PWM values applied to the fan.

Time	Temperature	Set point	PWM value
4	24	24	0
5	24.8	24	7
6	26.5	24	25
7	31.9	24	79
7	35.5	24	115
8	37.3	24	133
9	38.7	24	146
10	39.1	24	151
11	40	24	160
13	45.1	24	211
22	50	24	255
28	45.4	24	213
29	42.6	24	185
30	40.2	24	161
31	36.4	24	124
32	34.9	24	108
33	32.5	24	85
34	30.7	24	67
38	27.8	24	38
41	26.5	24	24

44	25.4	24	14
55	24	24	0
56	23.9	24	0
57	23.9	24	0
58	23.9	24	0
59	23.8	24	0
60	23.8	24	0
61	23.8	24	0
62	23.8	24	0
63	23.7	24	0
64	23.7	24	0

Table 2: Table summarising the temperature readings from thermistor, corresponding fan PWM values, and the setpoint.

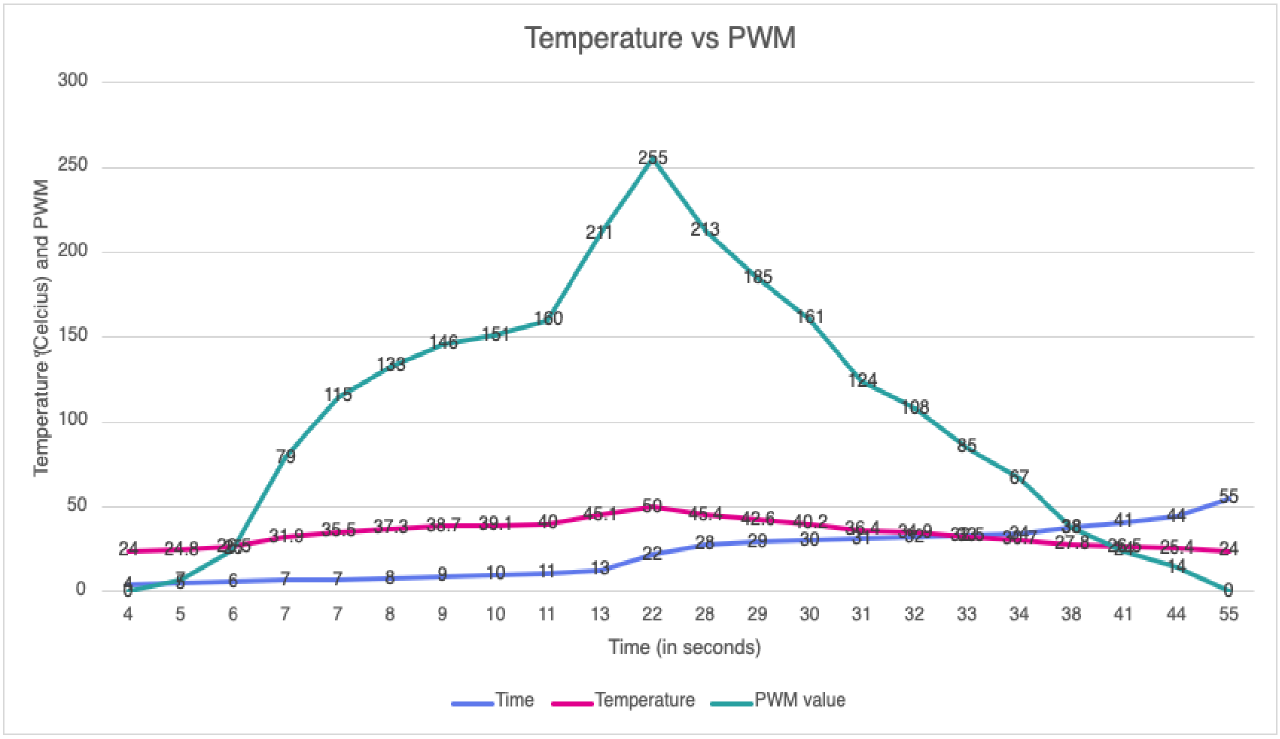


Figure 6: Chart showing behaviour of the fan as temperature increases overtime.

Figure 6 shows the relationship between temperature and fan PWM over time. As temperature increased, the PID controller responded by increasing the PWM duty cycle, increasing the fan speed. This demonstrates the expected proportional-derivative behaviour of the control loop.

System Stabilization and Return to Setpoint

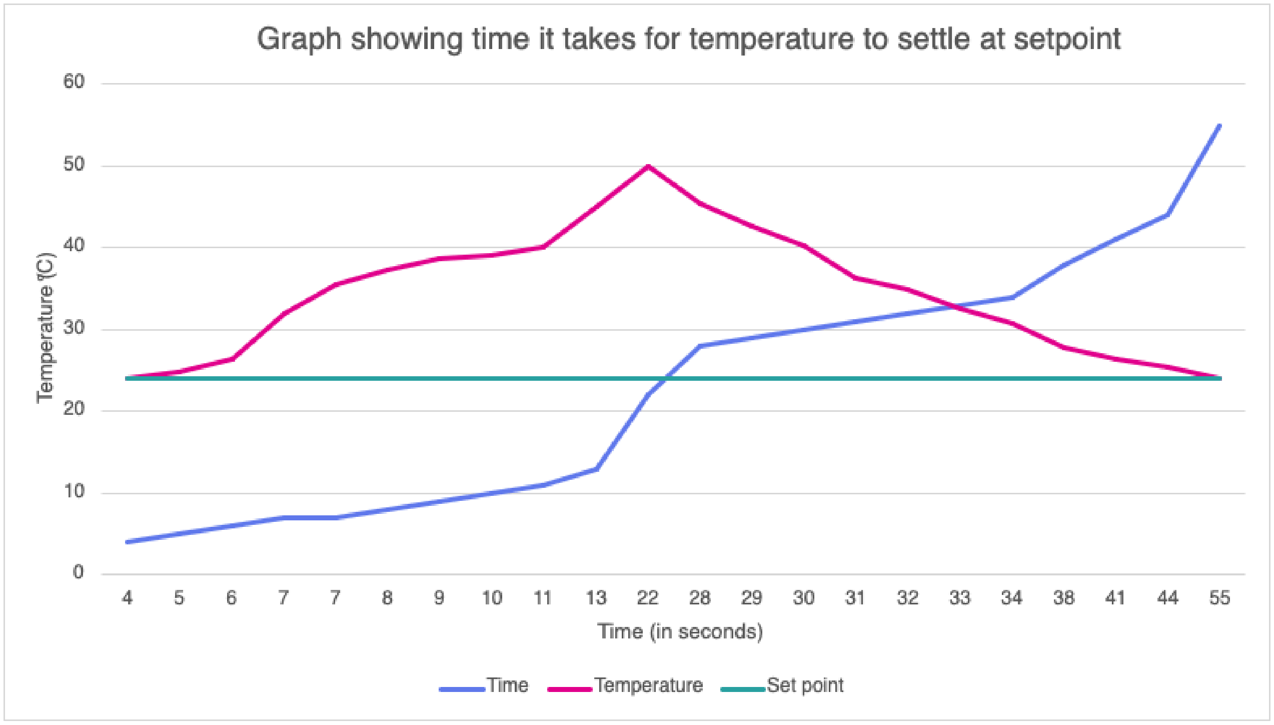


Figure 7: Chart showing the time it takes for the system to return to setpoint

Figure 7 displays the time required for the system to return to the setpoint after the heat source was removed. The controller reduced the PWM output as the enclosure cooled, allowing the temperature to approach 24°C without overshoot.

Settling Behaviour

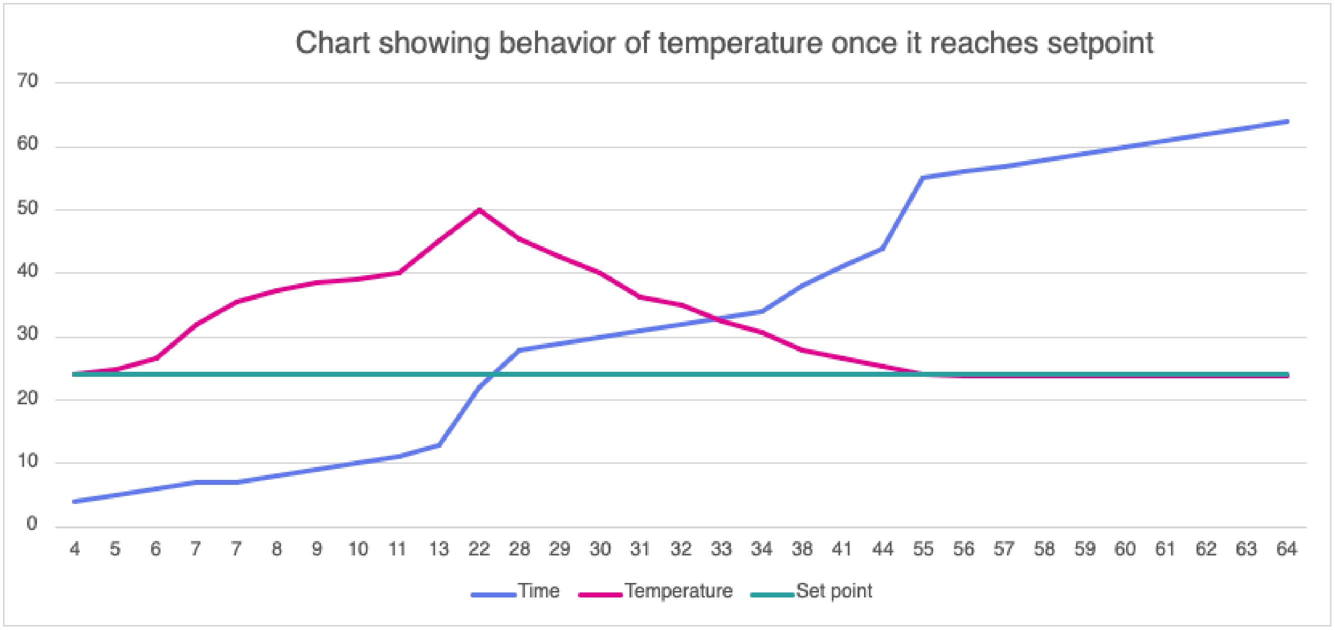


Figure 8: Chart showing the behaviour of the temperature once the system reaches settling time.

Figure 8 shows the system behaviour after stabilisation. Once the setpoint temperature was reached, the PID controller maintained the temperature within a tight band around 24°C (±0.2-0.3°C). The PWM value decreased significantly during this phase, indicating that the fan was only providing enough airflow to maintain thermal stability, and the controller avoided unnecessary oscillation.

DISCUSSION

Calibration Accuracy

The calibration results demonstrated that the thermistor closely matched the reference glass thermometer, with an average error of –0.14°C and a maximum error of ±0.1°C. This level of accuracy indicated that the calibrated Beta values and measured thermistor values produced reliable temperature estimates. The slight negative bias suggests a minor systematic offset, which is acceptable for environmental temperature control. Overall, the thermistor was sufficiently accurate for closed-loop PID regulation.

PID Controller Response

The PID controller responded effectively to increasing temperature within the test enclosure. As shown in figure 6, the PWM duty cycle increased proportionally as the temperature rose above the setpoint, demonstrating correct proportional-derivative behaviour. When the temperature exceeded 35-40°C, the controller increased the PWM toward its maximum value (255), resulting in the fan operating at full speed.

This increase in the fan's speed enabled the system to counteract the heat load and return the temperature toward the setpoint.

Stability at the Setpoint

Once the setpoint was reached, the controller maintained the temperature within approximately ±0.3°C of the 24°C setpoint. Figure 8 demonstrates that the PID system held the temperature in a narrow band with minimal fluctuation, indicating strong steady-state stability. The decreasing PWM output during this phase shows that the controller correctly reduced fan speed once only minimal airflow was required.

Limitations and Challenges

- Voltage Reference and ADC errors

  Initially, temperature readings were inaccurate because the Arduino's supply voltage was assumed to be 5.00V, and the thermistor and fixed resistor values were not measured. This caused errors in the calculated temperature.

  Resolution: All components were re-measured, and the actual values were updated in the code, resulting in accurate temperature readings.

- PWM Stuck at 0 or 255

  The fan's PWM output was initially non-functional, with the duty cycle remaining at 0 or 255.

  Resolution: It was found that the fan's PWM pin was not connected correctly. Reconnecting the PWM pin input directly to Arduino pin 3 restored proper control.

- DC fan Not fully Turning off at 0% PWM

  The 4-wire fan did not completely stop when the PWM signal was 0, this is likely due to the internal electronics maintaining a minimal standby rotation.

- Power Supply Challenge

Finding a suitable 12V power source for the fan was difficult, as standard batteries did not meet the required voltage and current rating.

Resolution: I bought a Back-UPS that comes with a battery rated 12V 6.5A.

Lessons Learnt

- A key lesson learnt was the value of using analog temperature sensors such as NTC thermistor compared to lower-resolution digital sensors like the DHT11. The thermistor allowed for precise calibration using known resistance values, high resolution ADC readings, smooth temperature measurement without quantization steps, and better integration with PID control. This resulted in more accurate temperature data and more stable PID performance.
- The project also reinforced the importance of component placement, wiring reliability, and the importance of measuring and using the measured value of each component.
- Another lesson learnt was regarding the tuning of the PID controller

Improvements

- Replacing the Soldering Iron with a proper internal heat source.
  The heating element used for generating internal heat during testing within the enclosure can be improved. Using a soldering iron was a hazard to begin with and could have caused a fire within such a small enclosure. Instead of using the soldering iron, an internal heat source such as a silicone heating pad, ceramic heater, or an infrared brooder lamp should be implemented. Using a multi-purpose internal heating source ensures more realistic heat distribution within the enclosure and enhances safety, allows more accurate characterisation of the system's thermal response, and better reflects real-world poultry environment conditions.
- Addition of Exhaust Air Temperature Monitoring
  An additional improvement is to incorporate an exhaust air temperature sensor positioned directly at the fan outlet. Measuring the temperature of the exiting airflow provides valuable information about the actual heat being removed from the enclosure. By comparing the internal ambient temperature to the exhaust temperature, the system can estimate cooling effectiveness and detect abnormal operating conditions such as blocked fan or reduced airflow.

CONCLUSION

In conclusion, the PID-controlled fan system successfully maintained a stable temperature within the test enclosure, demonstrating both accuracy and effective closed-loop control. The thermistor calibration provided reliable temperature readings, and the PID controller adjusted the fan speed based on the temperature changes. The system was able to achieve steady-state conditions and maintained the setpoint temperature within ±0.3°C. Despite the challenges related to component measurement, PWM connections, partial fan operation at 0 PWM, and power supply limitations, the project met its objectives and provided insights into sensor calibration, PID tuning, and thermal management.

REFERENCES

- *Thermistors*. (1997). https://www.thierry-lequeu.fr/data/NTC-RS.pdf
- Handson Technology. (n.d.). Data Specs (120x120x25) mm Brushless Fan 4-Wire PWM Control. In *Handsontec.com*. https://handsontec.com/dataspecs/motor_fan/12025%20Fan%204W.pdf
- Playduino. (2024, June 22). *Temperature measurements using Arduino - NTC thermistor - (Arduino Uno Programming for Beginners)* [Video]. YouTube. https://www.youtube.com/watch?v=Fs3_zl9GsGc
- Sophie. (2024, October 19). Arduino Mega 2560: Pinout, Features, datasheet, IDE, and Simulation. *Richard Electronics All Rights Reserved*. https://www.richardelectronics.com/blog/projects/arduino/arduino-mega-2560-pinout-features-datasheet-ide-and-simulation
- Digital Logic & Programming. (2024, October 28). *Intro to Arduino - PWM using analogWrite() and Dimming LEDs (Fall 2024)* [Video]. YouTube. https://www.youtube.com/watch?v=D7DbjbP6TgA
- *Thermistors*. (2025, November 4). Basic Electronics Tutorials. https://www.electronics-tutorials.ws/io/thermistors.html
- Tpi-Polytechniek. (2025, July 7). *Keeping optimal control over the indoor climate of poultry houses in tropical areas*. TPI. https://www.tpi-polytechniek.com/blog/keeping-control-over-climate-poultry-house-tropical-areas/#:~:text=Poultry%20regulate%20their%20internal%20temperatures,impacting%20feed%20intake%20and%20hydration.%E2%80%9D

APPENDIX

Code

```
#include <LiquidCrystal.h>
#include <math.h>

//Pin Assignments
const int analogPin = A0; //Reads temperature from thermistor
const int fanPin = 3; //Controls the fan via PWM

//Variables for PID controller
double setpoint = 24.0; //desired value

//Define PID parameters
double Kp = 10.0; //proportional gain
double Ki = 0.0; //integral gain
double Kd = 0.1; //derivative gain

float previous_error = 0; //Used to compute the derivative term
float integral = 0; // Initializes the integral term
unsigned long last_time = 0; //stores last loop timestamp in ms

//Create variables to store the temperature values
float temp = 0.00;

//LCD pin connections RS, E, D4, D5, D6, D7
LiquidCrystal lcd(6, 7, 8, 9, 10, 11);

//Declare variables to store values fixed resistor, VCC, max ADC value, thermistor, 25degrees in Kelvin, BETA value for
thermistor
const float fixed_resistor = 9960.0; //measure this actual value
const float VCC = 5.04;
const float MAX_ADC = 1023.0; // ADC resolution (0...1023)
const float thermistor_resistor = 10910.0; //measure this actual value
const float Kelvin_value = 298.15; // 25 degrees in Kelvin (25 + 273.15)
const float BETA = 3981.51; //Beta constant computed from datasheet (0 degrees, 32650 ohms to 150 degrees, 185.97ohms)


//average samples
const int SAMPLES = 20;
const int Samples_delay_ms = 10;
```

```
void setup(){
pinMode(fanPin, OUTPUT);
Serial.begin(9600);
Serial.println("Temperature: ");
Serial.print(temp);

lcd.begin(16,2);
last_time = millis();

}
void loop(){

//read all the raw analog values
long sum = 0;
for(int i = 0; i < SAMPLES; i++){
sum+= analogRead(analogPin); // reads the analogPin for the analog voltages
delay(Samples_delay_ms);
}

//store the raw values/analog voltage
float raw_ADC_values = sum / (float)SAMPLES;

//convert the raw analog voltage values to a digital voltage
float voltage = raw_ADC_values * (VCC/MAX_ADC);

//error message
if(voltage <= 0.0001){
Serial.println("Sensor error ");
lcd.setCursor(0,0);
lcd.print("Sensor Error.");
delay(1000);
return;
}

//compute thermistor resistance
float thermistor_resistance = fixed_resistor * (VCC/voltage - 1.0);

//convert thermistor resistance to temperature in Celcius
float InRatio = log(thermistor_resistance/thermistor_resistor);
float invT = (1.0/Kelvin_value) + (InRatio/BETA);
float temp_in_Kelvin = 1.0/invT;
float temp_in_Celcius = temp_in_Kelvin - 273.15;



//Time step calculation
```

```
unsigned long now = millis();
float time_difference = (now - last_time) / 1000.0; //Convert ms to seconds; dt is the time difference between current and
previous loop iteration
if(time_difference <= 0.00){
time_difference = 0.001; // this is to prevent dividing by 0
}
last_time = now;

//PID error terms
float error = temp_in_Celcius - setpoint;
integral += error * time_difference; // integral accumulates error over time
//anti wind up
if(integral > 300) integral = 300;
if(integral < -300) integral = -300;

float derivative = (error - previous_error) / time_difference; // rate of change of error

// PID output
float output = Kp * error + Ki * integral + Kd * derivative;
output = constrain(output, 0, 255);
int pwm_output = (int)output;

//Apply to the fan
analogWrite(fanPin, pwm_output);

//Update the error
previous_error = error;

Serial.print("RawADC: ");
Serial.print(raw_ADC_values, 0);
Serial.print(" Volt: ");
Serial.print(voltage, 3);
Serial.print(" Rtherm: ");
Serial.print(thermistor_resistance, 1);
Serial.print(" TempC: ");
Serial.println(temp_in_Celcius, 2);



//Display on LCD
lcd.setCursor(0, 0);
lcd.print("Temp: ");
lcd.print(temp_in_Celcius, 1);

lcd.setCursor(0, 1);
lcd.print("Fan: "); // clear old digits
lcd.setCursor(5, 1);
lcd.print(pwm_output);
```

```
delay(500);
}
```