MFA 600

Familiarization with thermocouples And PID controllers

(27- October- 2023)

Firas Ben Thayer

fibe0006@student.hv.se

University west

Firas Ben Thayer
fibe0006@student.hv.se

Project description

The project consists of two parts:

- Part 1 – Temperature measurement with thermocouple without compensation and with Cold junction compensation
- Part 2 – Report on DC Motor speed Control using arduino or any microcontroller

The temperature range you will use in the task can be found here (it is individual):

To get full marks for the tasks below, the answers must be well-reasoned!

- **Part 1:**

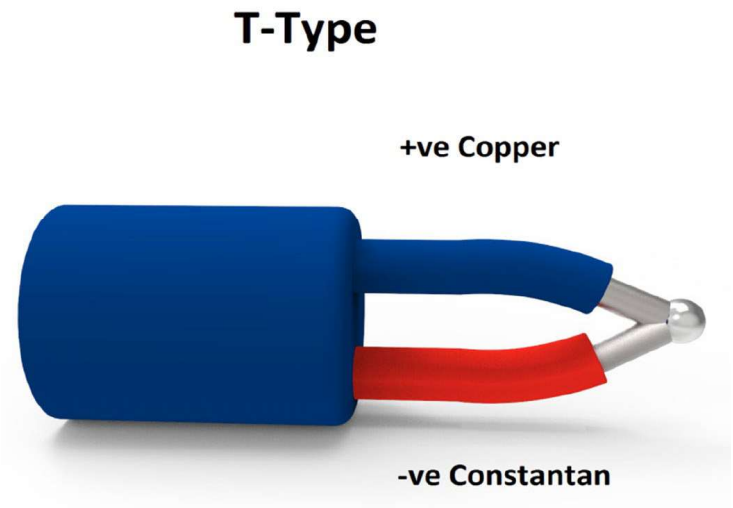
In this part, you must measure the temperature using a thermocouple. If you don't have the sensor, Use data sheet: https://www.thermocoupleinfo.com/pdf/type-t-thermocouple-reference-table.pdf

i. Sensor (4p)

The reference temperature is the room temperature (22 Deg C). Assume that the cold junction compensation wasn't done for the temperature that you report here.

Question 1: Is the sensor (thermocouple type T) a suitable sensor for your measurement range (justify your answer)?

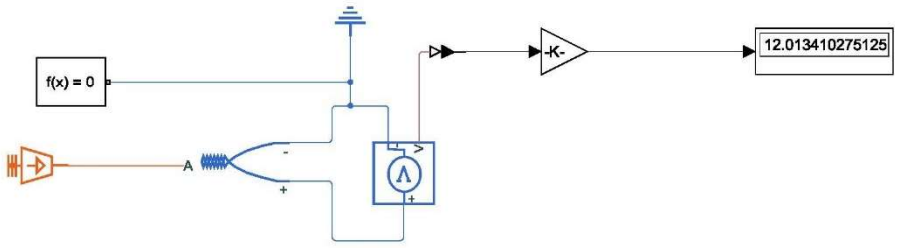
Thermocouples are among the easiest temperature sensors to use and obtain and are widely used in science and industry. They are based on the Seebeck effect that occurs in electrical conductors that experience a temperature gradient along their length. They are "simple", rugged, need no batteries, measure over very wide temperature ranges and more. (Thermocouple Temperature Sensors, how they work, color codes, recommended limits, limits of accuracy, calibration tables, vendor links and more. (archive.org))



TYPE T THERMOCOUPLE (tempsens.com)

My measurement range would be between 150°C and 250°C. Jumping by 5°C:

°C	0	1	2	3	4	5	6	7	8	9	10
150	6.704	6.754	6.805	6.855	6.905	6.956	7.006	7.057	7.107	7.158	7.209
160	7.209	7.26	7.31	7.361	7.412	7.463	7.515	7.566	7.617	7.668	7.72
170	7.72	7.771	7.823	7.874	7.926	7.977	8.029	8.081	8.133	8.185	8.237
180	8.237	8.289	8.341	8.393	8.445	8.497	8.55	8.602	8.654	8.707	8.759
190	8.759	8.812	8.865	8.917	8.97	9.023	9.076	9.129	9.182	9.235	9.288
200	9.288	9.341	9.395	9.448	9.501	9.555	9.608	9.662	9.715	9.769	9.822
210	9.822	9.876	9.93	9.984	10.038	10.092	10.146	10.2	10.254	10.308	10.362
220	10.362	10.417	10.471	10.525	10.58	10.634	10.689	10.743	10.798	10.853	10.907
230	10.907	10.962	11.017	11.072	11.127	11.182	11.237	11.292	11.347	11.403	11.458
240	11.458	11.513	11.569	11.624	11.68	11.735	11.791	11.846	11.902	11.958	12.013
250	12.013	12.069	12.125	12.181	12.237	12.293	12.349	12.405	12.461	12.518	12.574



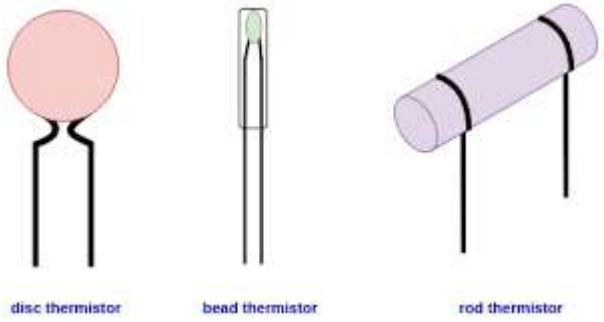
In the chosen range the thermocouple type T is suitable. These are the reasons:

- It's accurate enough (3 digits after 0).
- Its sensitivity is enough for our needs.
- It's sensitive for our environment conditions. Which is a lab, no high pressure or risk of corrosion.
- Multiple repeated measures show that the thermocouple is stable enough to use for our project.

On the other hand, we should be careful when picking the electrical components compatible with the sensor because of its small voltage signal which makes it susceptible to noise.

Question 2: Name another temperature sensor for your measurement range and state the pros and cons in relation to the thermocouple.

Another temperature sensor would be the thermistor. It is an electrical resistor whose resistance is greatly reduced by heating, used for measurement and control (Oxford dictionary):



Thermistor: Construction, Working Principle, Types and Applications | Sensors and Transducers | Teachics

When it comes to pros:

- The thermistor is highly sensible. It can detect the slightest change of temperature.
- The size of the thermistor is relatively small compared to the thermocouple.
- Its cost is lower than that of the thermistor.
- Multiple models allowing flexibility in application.

When it comes to cons:

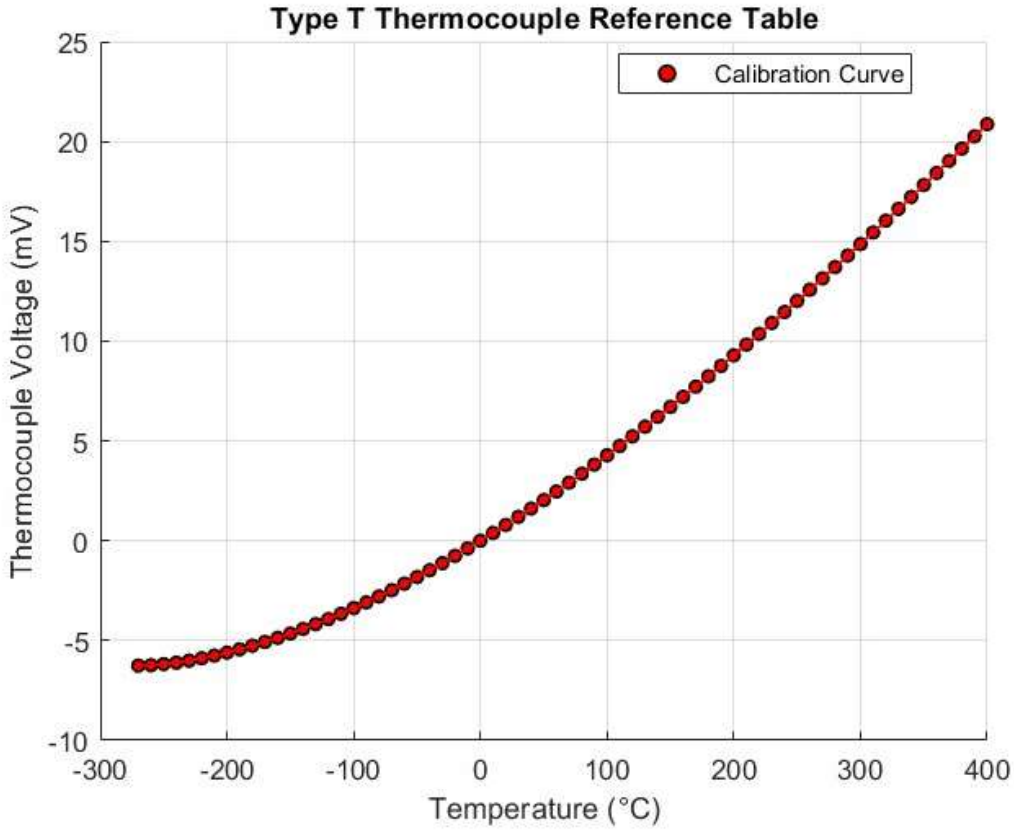
- Usually, the max temperature is around 300°C compared to the 400°C limit on the thermocouple.
- Compared to thermocouples, the thermistor is relatively unstable in the long run.
- The thermistor is fragile: It can easily break.

ii. Calibration (7 p)

The next step is to calibrate your sensor. Set a number of different temperatures (at least 10). Start at your specified min temperature and end at your specified max temperature. Make a graph (in MATLAB or Excel) showing the calibration curve and the true temperature. The calibration curve must show the temperature in (deg C) on the x-axis. The measured milli volts from the thermocouple on the y1-axis. The data from data sheet will be in y2 axis. if you don't have access to the real thermocouple, use the data sheet ONLY. And there will be Y2 and X axis only.

Add the graph to your report (clearly show which curve is the calibration curve and which is the true temperature) and explain any differences between the curves.

The thermocouple type T used in our case is in an environment where the reference temperature is 22°C. Before we start the experiment, here is a graph of a thermocouple at optimal conditions:



In my case, I chose 20 values to work on between 150°C and 250°C and compared the results between the two thermocouples at different reference temperatures:

°C	Voltage in mV (0°C Ref temperature)	Voltage in mV (22°C Ref temperature)
150	6.704	5.616
155	6.956	5.861
160	7.209	6.107
165	7.463	6.355
170	7.720	6.604
175	7.977	6.855
180	8.237	7.107
185	8.497	7.361
190	8.759	7.617
195	9.023	7.874
200	9.288	8.133
205	9.555	8.393
210	9.822	8.654
215	10.092	8.917
220	10.362	9.182
225	10.634	9.448
230	10.907	9.715

235	11.182	9.984
240	11.458	10.250
245	11.735	10.530
250	12.013	10.797

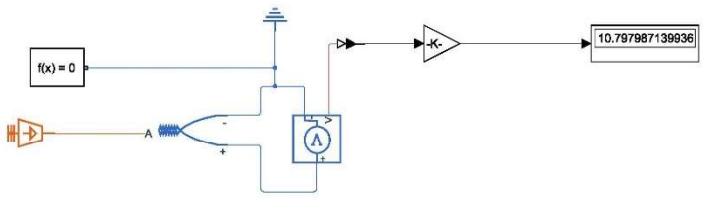
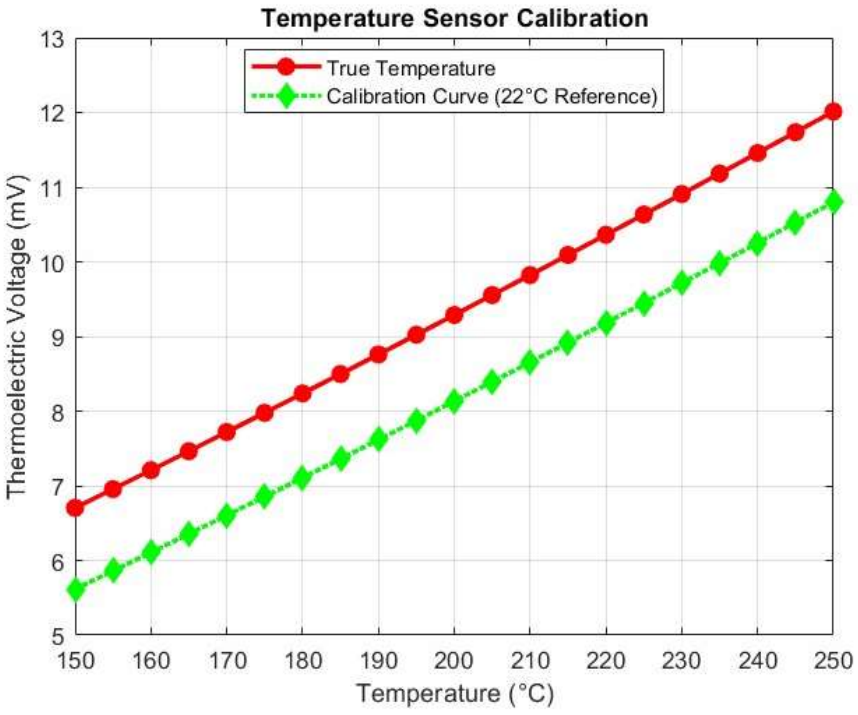


Figure 1: Simulation of thermocouple type T with 0°C as reference.

This is the graphic representation of the data given:



We don't have a real sensor to witness a difference, but in general.

Question 3: Does the sensor provide accurate temperature measurements? If not, what kind of calibration is required to compensate for the inaccuracies in the measurements?

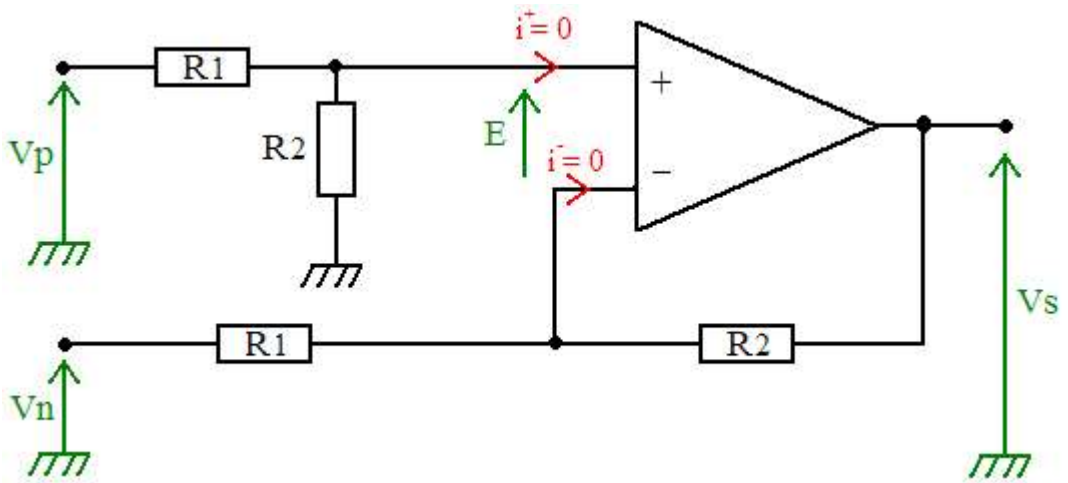
The sensor would not be accurate because the reference temperature has changed to 22°C. There are multiple kinds of calibrations to compensate the errors including amplification, Linearization, cold junction compensation (CJC), signal conditioning or filtering, analogue to digital conversion (ADC), scaling and offset adjustment, impedance matching and isolation to get rid of the noise since the thermocouple's output is small.

***Now, you must measure the temperature of the same system, however you must now perform a cold junction compensation along with signal conditioning circuit and show the results in simulation.**

To conduct a cold junction compensation, the first step we need to do display side by side the voltage of both the thermocouple type T at 22°C Ref temperature and a thermocouple which has 0°C as Ref temperature. We first need to setup an op-amplifier for subtraction purpose to measure the offset between the two thermocouples and then an addition op-amplifier to add the offset to the thermocouple:

1. Subtraction:

First, we must find the error. To do so we first did some calculation like below:



Montages de base de l'amplificateur opérationnel (elektronique.fr)

The formula is as below:

$$V_s = \frac{R_1}{R_2}(V_p - V_n)$$

Based on this formula, we obtained the variables down below:

$$R_1 = R_2 = 10\ \Omega$$

We then switch over to Simulink to add the data:

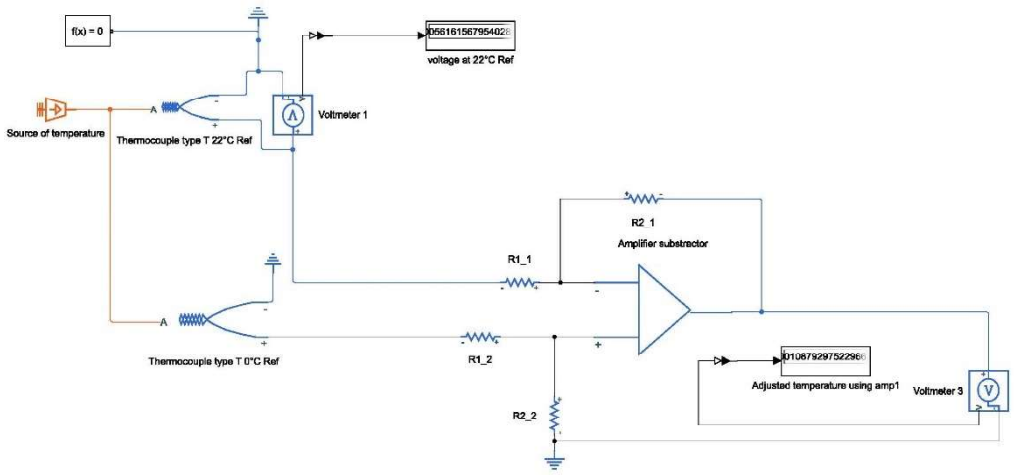
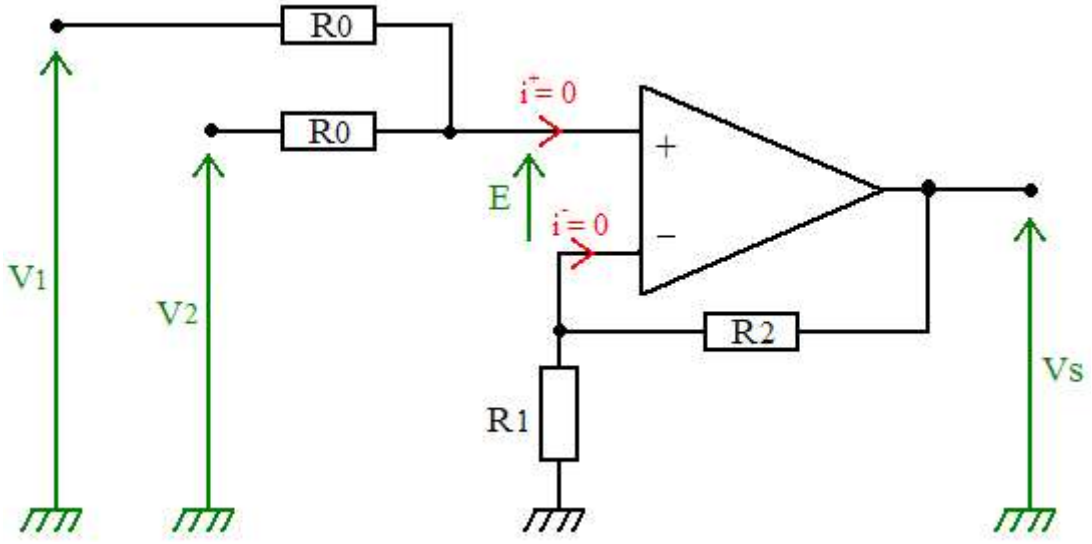


Figure 2: Calculating offset.

2. Addition:

After calculating the offset, an analysis and calculations are first conducted. The amplifier used is the non-inverter mode. It is an amplifier and a voltage adder at the same time:



Montages de base de l'amplificateur opérationnel (elektronique.fr)

In our specific context, V1 represents the thermocouple voltage, V2 stands for the constant voltage to be introduced, and R2 corresponds to Rf. The general formula applicable in this scenario is as follows:

$$V_s = \frac{R_1 + R_2}{nR_1} + \sum_{k=1}^{n} V_k$$

Based on this formula, we obtained the variables down below:

$$\begin{cases} n = 2 \\ R_0 = R_1 = R_2 = 10\ \Omega \end{cases}$$

The experiment commenced with a simulation of temperature at 22°C, followed by the addition of a constant voltage. To enhance clarity, the output was subsequently amplified to obtain the voltage in millivolts (mV):

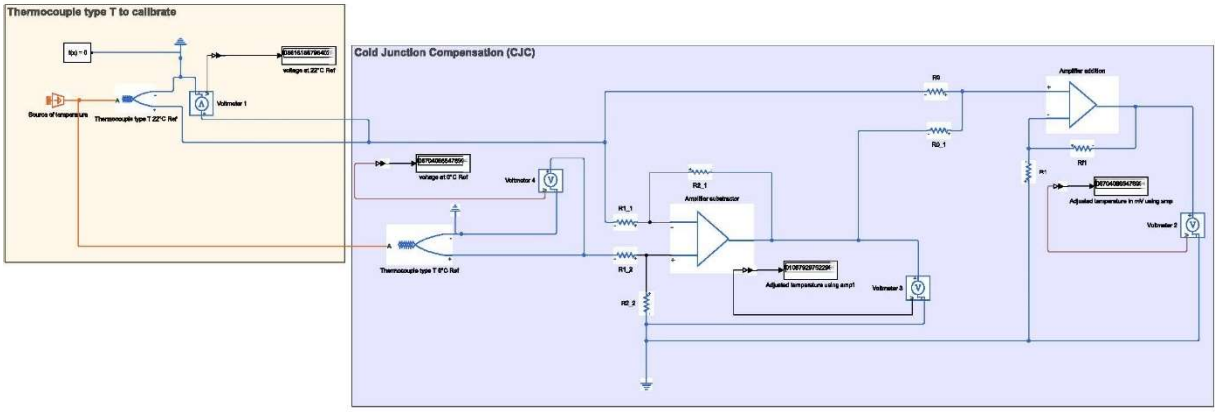


Figure 3: Simulation of thermocouple type T at 22°C Ref temperature and adjustment.

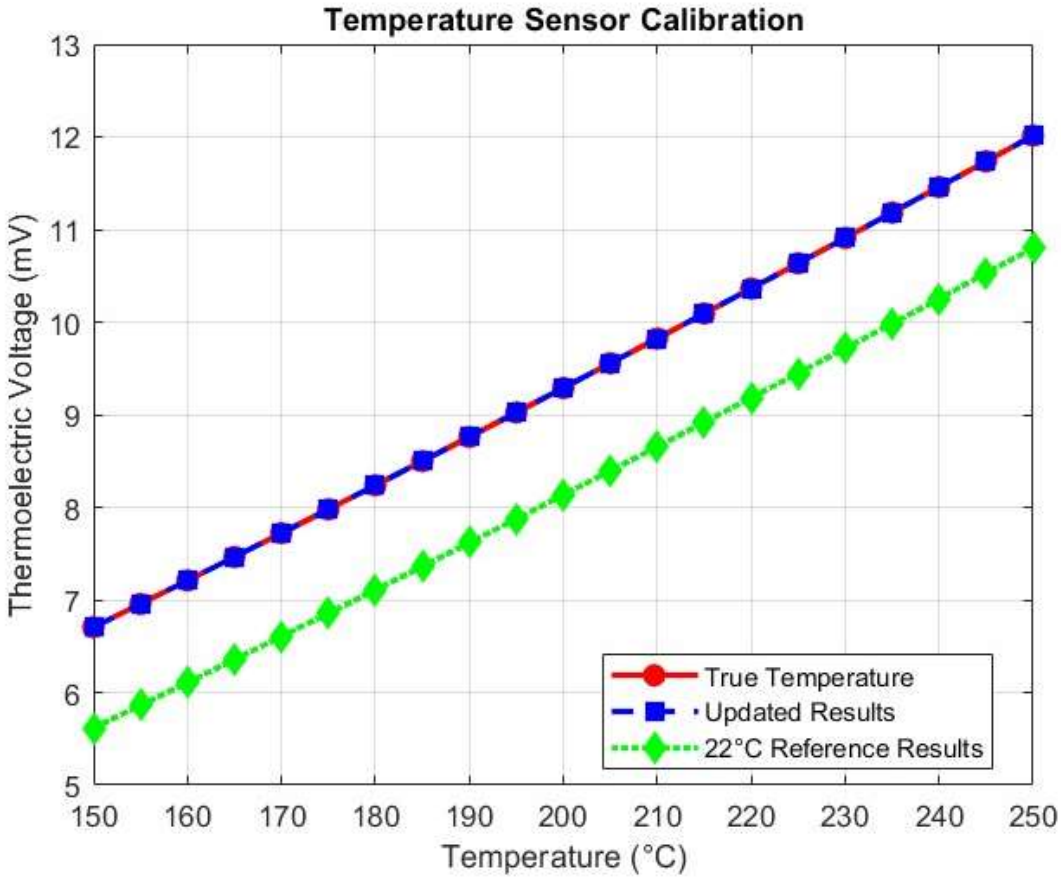
Cold Junction Compensation is required considering the difference in reference junction temperatures (0°C and 22°C room temperature), it is evident that the temperature measurements obtained are not accurately representative. To rectify this discrepancy, an attempt was made to calibrate the thermocouple by factoring in the voltage differential between 0°C and 22°C. The subsequent findings are as follows:

°C	Voltage in mV (0°C Ref temperature)	Voltage in mV (22°C Ref temperature)	Offset
150	6.704	5.616	1.088
155	6.956	5.861	1.095
160	7.209	6.107	1.102
165	7.463	6.355	1.108
170	7.720	6.604	1.116
175	7.977	6.855	1.122
180	8.237	7.107	1.130
185	8.497	7.361	1.136
190	8.759	7.617	1.142
195	9.023	7.874	1.149
200	9.288	8.133	1.155
205	9.555	8.393	1.162
210	9.822	8.654	1.168
215	10.092	8.917	1.175
220	10.362	9.182	1.180
225	10.634	9.448	1.186
230	10.907	9.715	1.192
235	11.182	9.984	1.198
240	11.458	10.250	1.208
245	11.735	10.530	1.205
250	12.013	10.797	1.216

Down below is the formula to calculate the Theoretical result voltage:

$$\begin{cases} V_S = V_{Ref(22°C)} + e \\ e = |V_{Ref(0°C)} - V_{Ref(22°C)}| \end{cases}$$

As shown from the results in the data table, there still is an error. The margin can be further seen in this graph:



It is important to note that the increasing temperature exacerbates the perceptibility of the error margin, a phenomenon that warrants closer examination in our experimental analysis.

iii. Measuring amplifier (10 p)

You will now design a measurement amplifier for your measurement system. In the next step, when the sensor signal is to be sampled and AD converted, an AD converter with the reference voltage of 5 volts will be used. Therefore, design your amplifier so that at your maximum temperature you get an output signal from the amplifier that is close to 5 volts (not over). **Design your minimum range to 1 Volt and maximum range to give 5 Volt.**

Question 4: Add your calculations for the amplifier to your report. Show how you have chosen the values of your resistors. Implement your amplifier in Simulation.

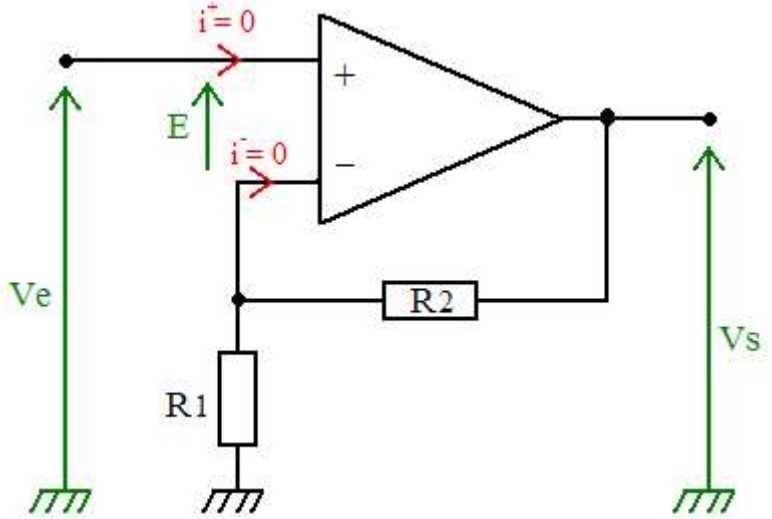
We want to transform voltage in [6.704 x 10^{-3} V 12.013 x 10^{-3} V] interval into [1V 5V]. We're going to calculate the scope and intercept value:

$$y = mx + b \Leftrightarrow \begin{cases} 1 = m \times 6.704\ x\ 10^{-3} + \quad b \\ 1 = m \times 12.013\ x\ 10^{-3} + b \end{cases}$$

$$\Leftrightarrow \begin{cases} m = \dfrac{\Delta y}{\Delta x} = 753.43755886231 \\ b = -4.05104539461 29\ V \end{cases}$$

$$\Rightarrow y = 753.43755886231x - 4.05104539461 29$$

We will now amplify our data according to the results and add the necessary components to get a value between 1V and 5V. For this we will be using an op-amplifier which performs a multiplication first to:



Montages de base de l'amplificateur opérationnel (elektronique.fr)

$$V_s = \left(1 + \frac{R_2}{R_1}\right) V_e \Leftrightarrow m = 1 + \frac{R_2}{R_1} = 753.43755886231$$

$$\Leftrightarrow \begin{cases} R_1 = 10\ \Omega \\ R_2 = 7534.3755886231\ \Omega \end{cases}$$

This is the block representing it in Simulink:

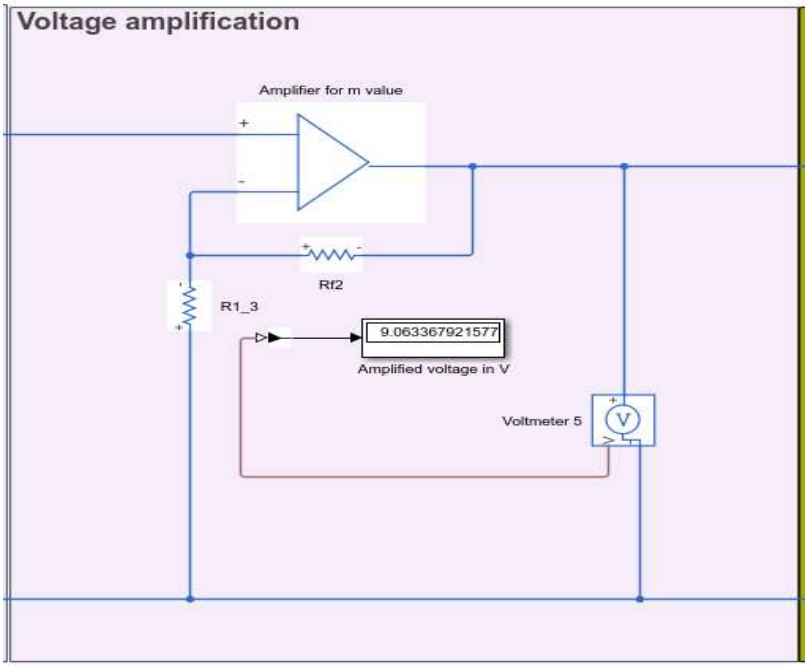
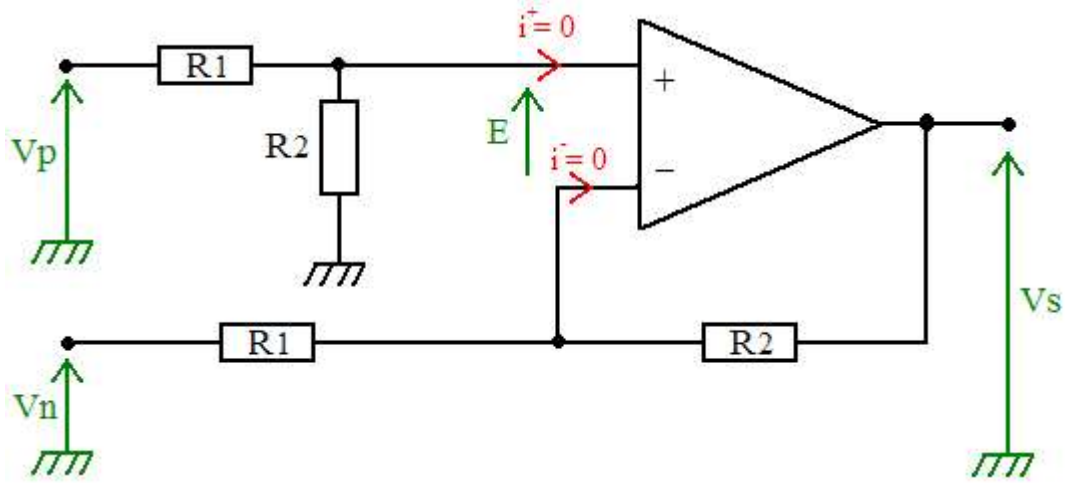


Figure 4: Voltage amplification.

Next, we add a subtractor since b value is negative:



Montages de base de l'amplificateur opérationnel (elektronique.fr)

$$V_s = (V_p - V_n) \times \frac{R_2}{R_1} \Leftrightarrow \begin{cases} V_n = b = 4.0510453946129\ V \\ \dfrac{R_2}{R_1} = 1 \Leftrightarrow R_1 = R_2 = 10\ \Omega \end{cases}$$

This is the block representing it in Simulink:

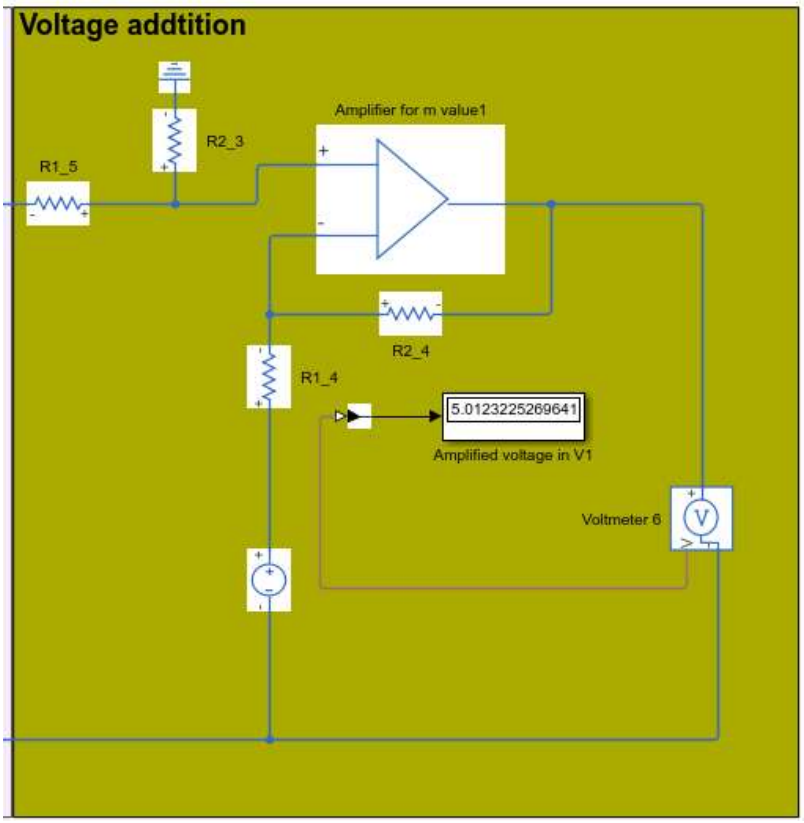


Figure 5: Voltage addition.

iv. Sampling and AD conversion (4 p)

The output signal from your measuring amplifier must now be sampled and AD-converted. The AD converter must have a reference voltage of 5 v.

Question 5: Now choose the number of bits in your AD converter (you choose yourself, justify your choice) and calculate what resolution you get with this number of bits. Add your calculation to the report.

The sensitivity of a thermocouple is a measure of how much voltage or electromotive force (EMF) it generates in response to a unit change in temperature. It's expressed in microvolts per degree Celsius (µV/°C) and is specific to the type of thermocouple being used. The sensitivity can be calculated using the thermoelectric voltage equation or the Seebeck coefficient:

$$S = \frac{E}{\Delta T}$$

Where:

S: is the Seebeck coefficient in µV/°C.

E: is the thermoelectric voltage or EMF produced by the thermocouple in µV/°C.

ΔT: The temperature difference in degrees Celsius between the two ends of the material in °C.

The type T thermocouple is made of copper (temperature) and constantan (reference). These are there Seebeck coefficient:

$$Copper\ (Cu)\ \approx 6.3\mu V/°C$$

$$Constantan\ (CuNi) \approx -44.0\ \mu V/°C$$

$$Sensitivity = |\ (-44.0) - (6.3)\ | = |\ (-50.3)\ | = 50.3\mu V/°C$$

However, it's important to note that the thermocouple type T has nodes impurities which heavily impacts the sensitivity of the sensor. Type T thermocouple sensitivity is generally assumed within the range of 40-43 µV/°C. We're going to assume that the sensitivity is equal to 43.4 µV/°C in our case.

As said above the AD converter must have a reference voltage of 5V. To determine the number of bits required for an analog-to-digital converter (ADC) to cover a specific temperature range, we can use the following formula:

$$N = log_2(\frac{V_{ref}}{V_{resolution}})$$

$$\Leftrightarrow \begin{cases} V_{ref} = 5V \\ V_{resolution} = sensitivity \times Gain\ (m) = 43.4 \times 753.43755886231 \times 10^{-6} = 32.699\ \text{mV} \end{cases}$$

Where:

$$\begin{cases} N: The\ number\ of\ bits. \\ V_{range}: Voltage\ range. \\ V_{resolution}: The\ resolution\ that\ we\ chose. \end{cases}$$

$$\Rightarrow \boldsymbol{N = log_2\left(\frac{5}{32.699 \times 10^{-3}}\right) = 7.25652938366}$$

As a result, we will choose an 8-bit AD converter.

Question 6: What will be your maximum error due to the quantization uncertainty (in degrees Celsius)?

We chose an 8-bit AD converter. Our temperature range is from 150°C to 250°C. The maximum quantization error for temperature would be:

$$MQE = \frac{1}{2} \times \frac{1}{2^n} \times (\Delta T)$$

$$= \frac{1}{2^{n+1}} \times (\Delta T)$$

$$\Leftrightarrow \begin{cases} n = 8 \\ \Delta T = 250 - 150 = 100\ °C \end{cases}$$

$$\Rightarrow MQE = \frac{1}{2^9} \times 100 = 0.1953125\ °C$$

What we can conclude with this is that it takes 0.390625 (0.1953125 x 2) °C to increment 1 bit from the ADC.

Add your calculations to the report. The calculations must show calculations from the entire chain, from physical value to a digital value after the AD converter.

Vref = 5V meaning that the input rage must be 5 volts:

- For Analog-Digital Converter [lowest value highest value] is set to [1.012 6.012] equivalent to Vref difference range and 2e1 conversion start frequency:

Digital output would then vary from 0 to 205 based on this formula:

$$Digital\ output = \frac{(Amplified\ Output\ voltage\ (V) - lowest\ value)}{V_{ref}} \times 2^{number\ of\ bits}$$

Once we calculated the ADC parameters. It's time to simulate it:

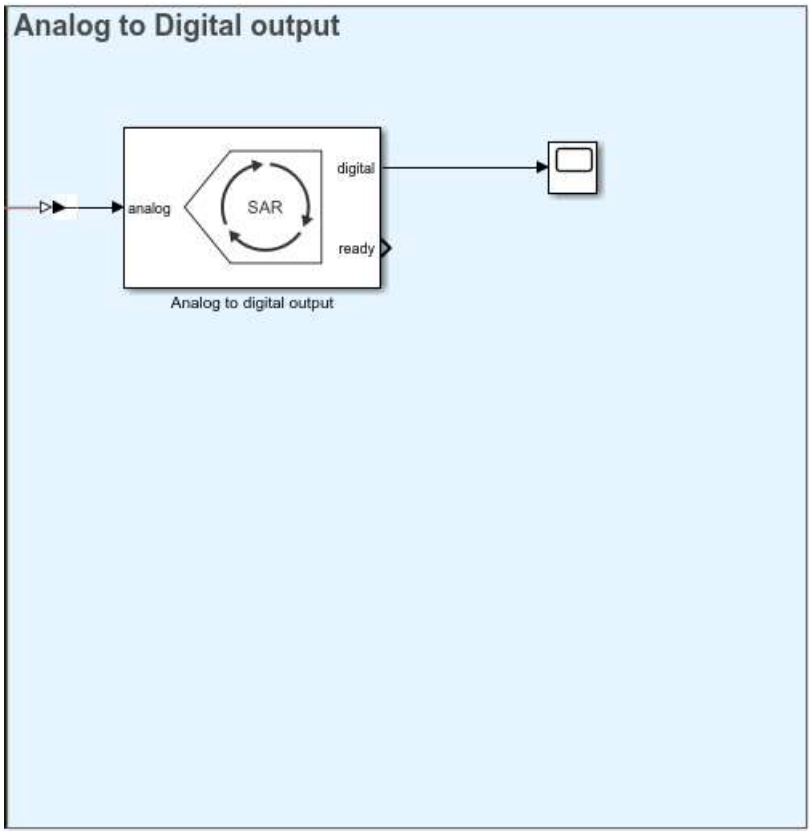


Figure 6: Analog to digital.

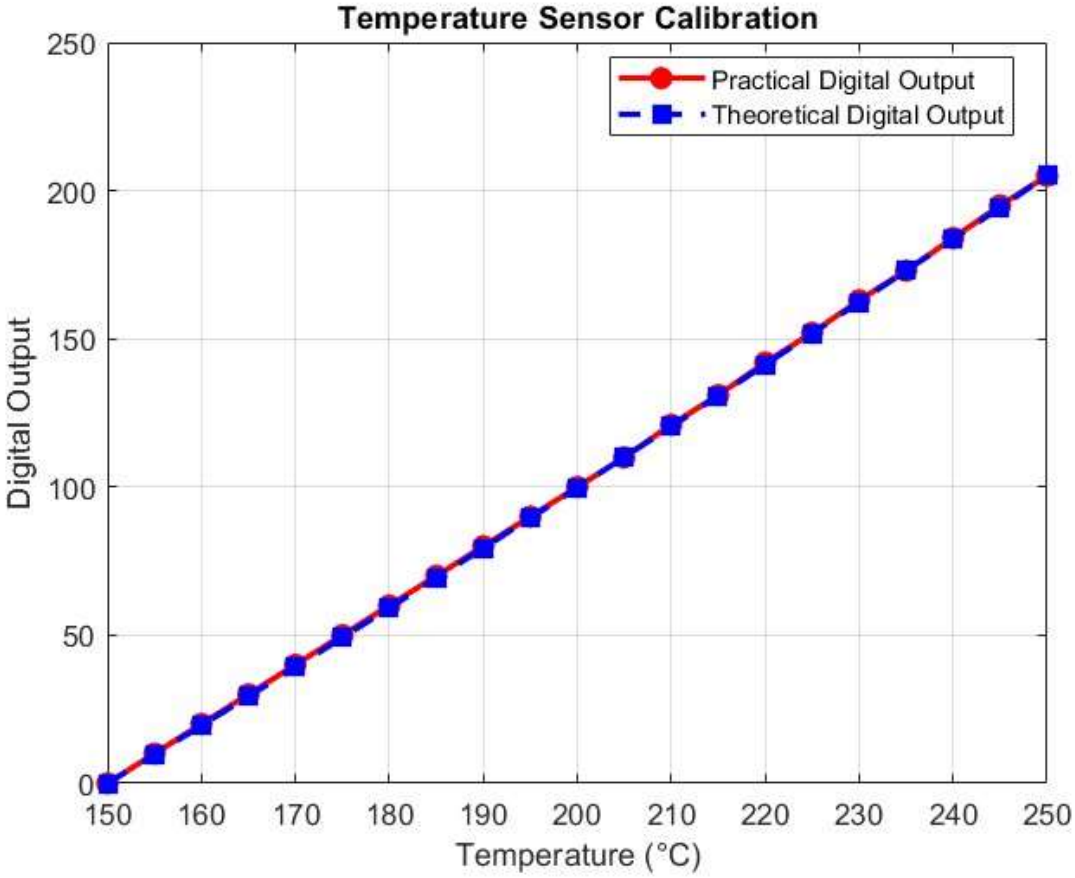
Sample Data collection:

S.No	Temperature (Deg C)	Output voltage (mV)	Compensated Output voltage (mV)	Amplified Output voltage (V)	Practical Digital output	Theoretical Digital output (not rounded)
1	150	5.616	1.088	1.007	0	0
2	155	5.861	1.095	1.197	10	9.728
3	160	6.107	1.102	1.388	20	19.5072
4	165	6.355	1.108	1.58	30	29.3376
5	170	6.604	1.116	1.773	40	39.2192
6	175	6.855	1.122	1.967	50	49.152
7	180	7.107	1.13	2.163	60	59.1872
8	185	7.361	1.136	2.36	70	69.2736
9	190	7.617	1.142	2.557	80	79.36
10	195	7.874	1.149	2.756	90	89.5488
11	200	8.133	1.155	2.956	100	99.7888
12	205	8.393	1.162	3.157	110	110.08
13	210	8.6543	1.1677	3.359	121	120.4224
14	215	8.917	1.175	3.562	131	130.816
15	220	9.182	1.18	3.767	142	141.312
16	225	9.448	1.186	3.972	152	151.808

17	230	9.715	1.192	4.178	163	162.3552
18	235	9.984	1.198	4.385	173	172.9536
19	240	10.25	1.208	4.593	184	183.6032
20	245	10.53	1.205	4.802	195	194.304
21	250	10.797	1.216	5.012	205	205.056

The error of digital output is due to the maximum error quantization.

The graph is as follows:



Conclusion:

In this part of the project, we learned about the thermocouple type T. We calibrated it, amplified its output in accordance with a given range then we transformed it into a digital output using an ADC.

Side Note: I think that having a smaller Vref in my range of measures is better. In fact, Vref = 4 V meaning a range of [1.012 5.012] on the ADC presents a well spread output across all the 8 bits of the ADC since our amplified output is between 1.007 V and 5.012 V. The results would range from 0 to 255 unlike in our cases which is from 0 to 205. On a side note, I think it is a good practice to put the amplified voltage range coming into the ADC [1.007;5.012] in the middle bits wise. Since, we only used 205 bits. We can make 150°C equivalent to 25 bits and 250°C equivalent to 225 bits.

Submission

Summarize your results, calculations and answers to questions in a Word document. The document and Simulink models are submitted via Canvas.

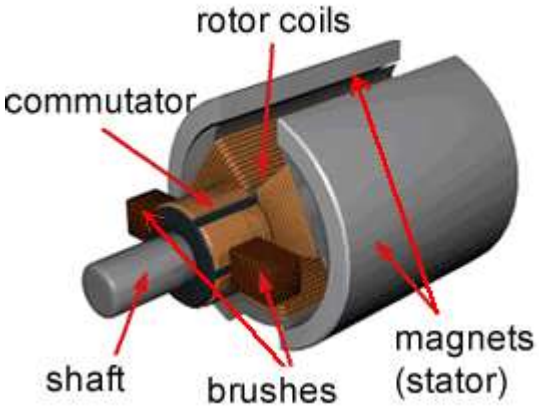
- **Part 2**

a. In this section, write a report (Maximum 1 or 2 pages) on Speed control of DC Motor using Arduino or any microcontroller. The report should contain information on the DC Motor model, PID Controller, response of P, PI, PID control. The following references shall be used but not limited to. (5 p)

Introduction:

Speed control of DC Motor in general is one of the basics of mechatronics. In this project, we will use a Raspberry Pi to control the motor using an integrated PID (Proportional-Integral-Derivative) controller. The PID controller is a feedback control system that uses the three parameters to control the output of a system based on the error between the setpoint and the actual process variable. The error in our case is in °C since we want to keep it in the spirit of temperature measurement. In this part, we will discuss the DC Motor model, PID Controller, and the response of P, PI, and PID control.

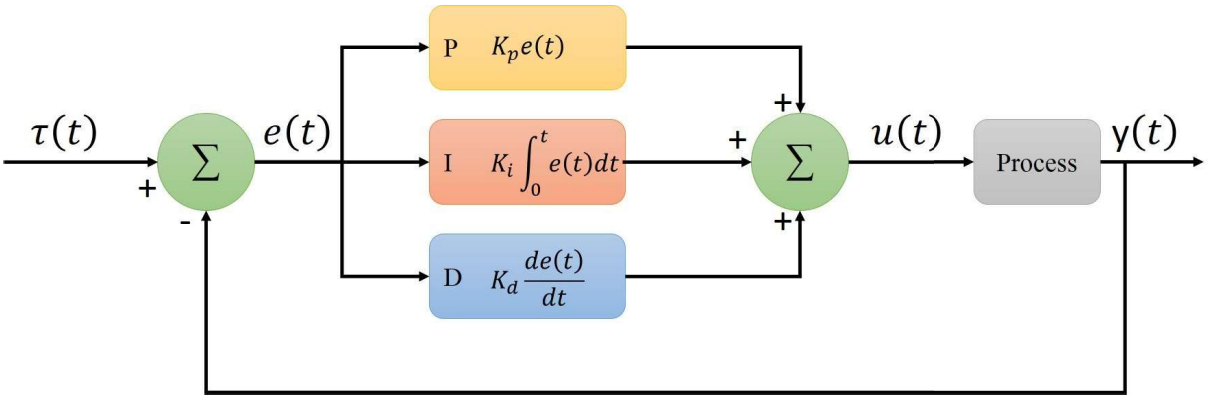
1) DC Motor Model:



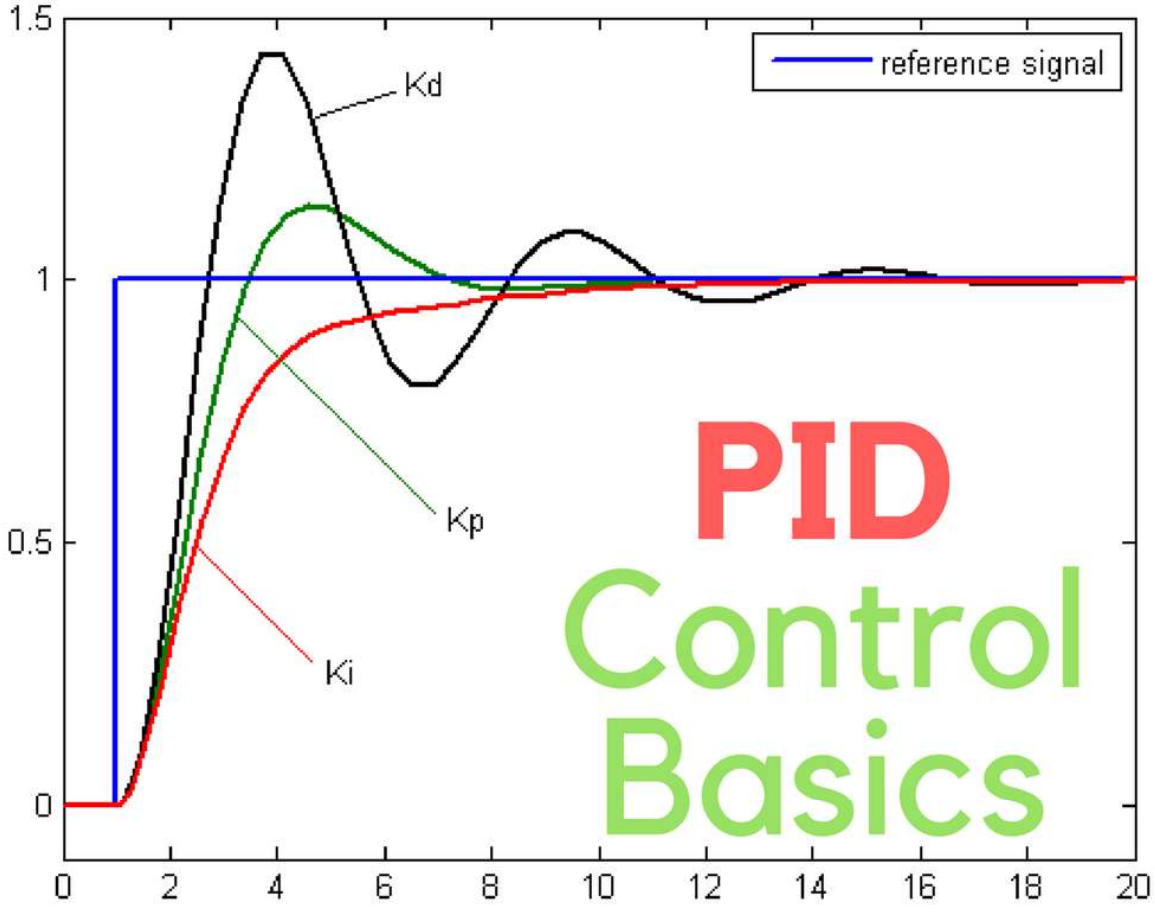
3-pole DC electric motor

A DC motor is a type of electric motor that converts electrical energy into mechanical energy. It consists of a stator and a rotor, which are connected by a commutator. The stator contains the windings that are connected to the power supply, and the rotor contains the magnetic field that is produced by the stator. When a current is passed through the windings, the magnetic field in the rotor generates a torque that causes the rotor to rotate.

2) PID Controller:



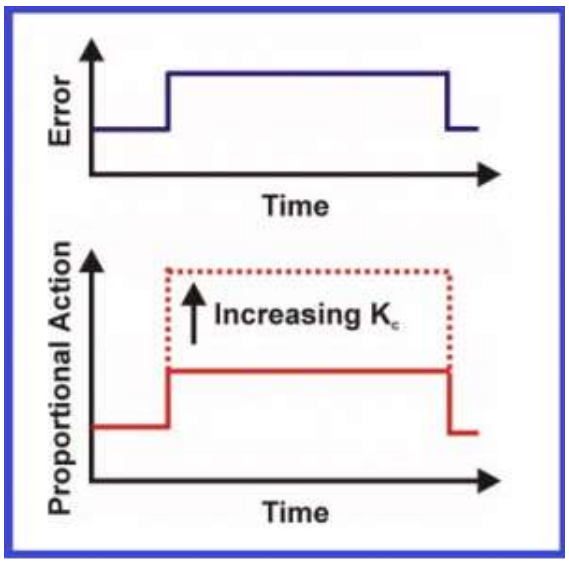
PID Controller Concept



Pid Control Graph

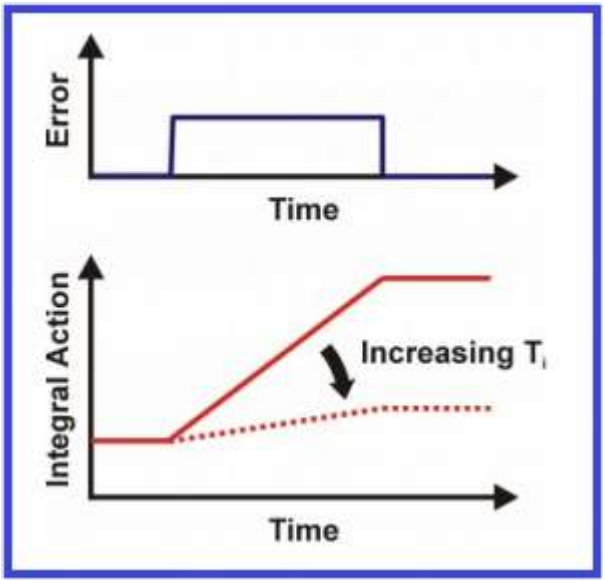
The PID controller is a feedback control system that uses three parameters to control the output of a system based on the error between the setpoint and the actual process variable. The three parameters are:

1. Proportional (P): Also known as first order. This parameter controls the proportional error (temperature – setpoint) of the system. It is the main controller that helps to reduce the error quickly.



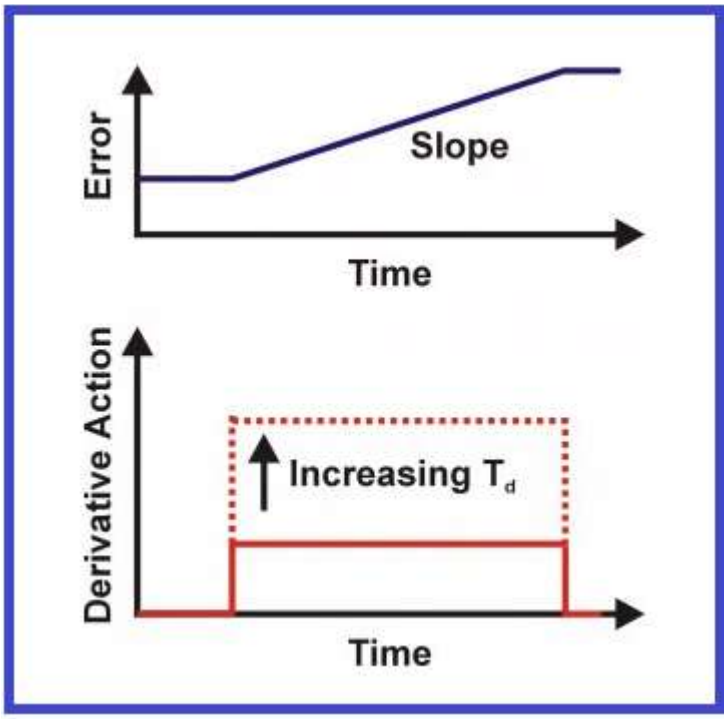
Proportional

2. Integral (I): This parameter controls the integral error of the system. It helps to reduce the steady-state error by adjusting the output over time.



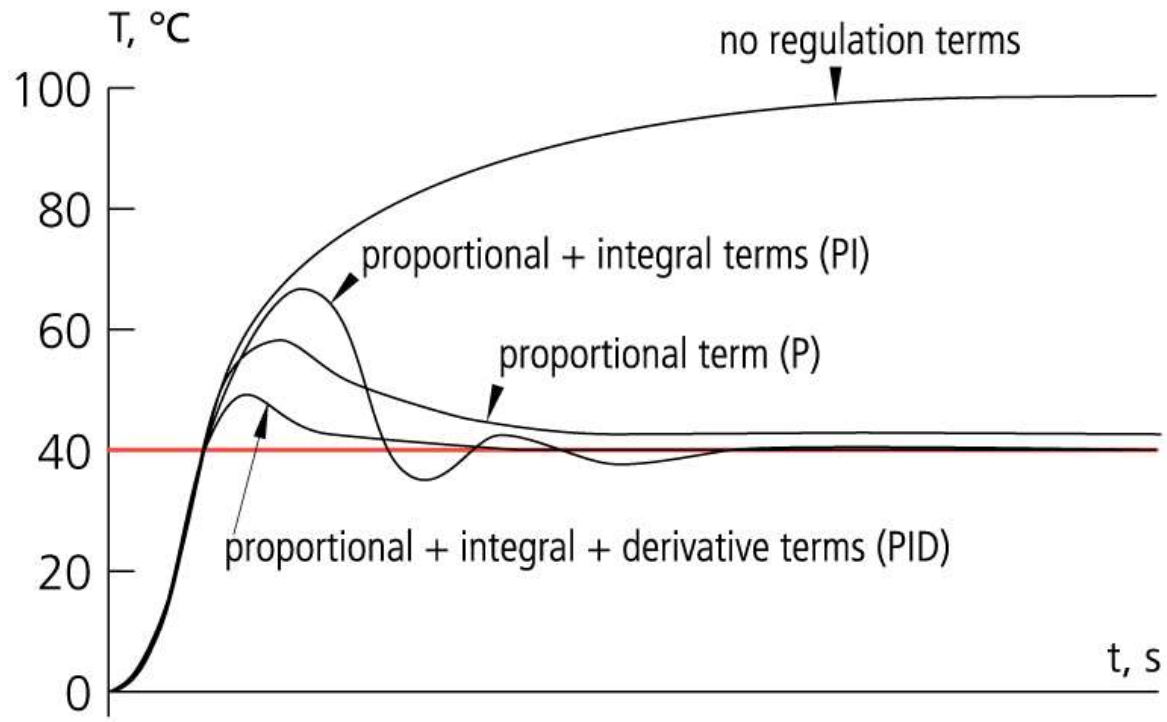
Integral

3. Derivative (D): This parameter controls the derivative error of the system. It helps to reduce the overshoot and undershoot of the system by adjusting the output based on the rate of change of the error.



Derivative

3) Response of P, PI, and PID Control:



Effect of the PID Controller

1. Proportional (P) Control: In P control, the output of the system is directly proportional to the error. As the error increases, the output increases, and as the error decreases, the output decreases. This type of control is effective for small errors, but it tends to overshoot and undershoot the setpoint. **Acceleration at the cost of stabilization time**.

2. Proportional-Integral (PI) Control: In PI control, the output of the system is proportional to the error, and it also includes an integral term that helps to reduce the steady-state error. The integral term accumulates the error over time and adjusts the output accordingly. This type of control is effective for larger errors and is less prone to overshoot and undershoot the setpoint. **More acceleration at the cost of stabilization time.**

3. Proportional-Integral-Derivative (PID) Control: In PID control, the output of the system is proportional to the error, and it also includes an integral term and a derivative term. The integral term accumulates the error over time and adjusts the output accordingly. The derivative term helps to reduce the overshoot and undershoot of the system by adjusting the output based on the rate of change of the error. **This type of control is the most effective and is recommended for most applications.**

Conclusion:

In summary, this project not only demonstrates the technical prowess of control systems but also demonstrate the significance of selecting the appropriate control strategy when aiming to maintain the desired speed of a DC motor. A comprehensive understanding of control mechanisms is important in various industries and applications, from robotics and automation to manufacturing and beyond.

b. Develop a simulated or hardware Speed control of DC motor model using a microcontroller individually without using models borrowed from other sources. (10 P)

Greenhouse Temperature Control System Using Raspberry Pi

(Including Simulation in Proteus 8)

Green House

I. Introduction:

In the world of technology and agriculture, the "Greenhouse Temperature Control System Using Raspberry Pi" project helps in the maintenance crops by automating the process. This report delves into our pursuit of precise greenhouse temperature control, combining the power of a Raspberry Pi, the insight of a BME280 sensor, and the motor speed control using a PID controller. Beyond the real-world application, we explore the advantages of simulation using Proteus 8 professional, offering a holistic view of our system's capabilities. Our project showcases the harmony between technology and sustainable agriculture, promising a future of optimal plant growth and resource/time efficiency.

II. Simulation environment:

Before introducing our solution to the greenhouse. Proper simulation is needed to estimate the components needed and figure out the correct way to set it up. After performing research in the circuit simulation market, I finally settled with Proteus 8 professional as the adequate tool for our case.

Proteus 8 professional is a circuit simulating and designing software developed by Labcenter Electronics and is widely used in the industry. Here are its main advantages and features:

- Schematic Capture.
- PCB layout.
- Simulation.

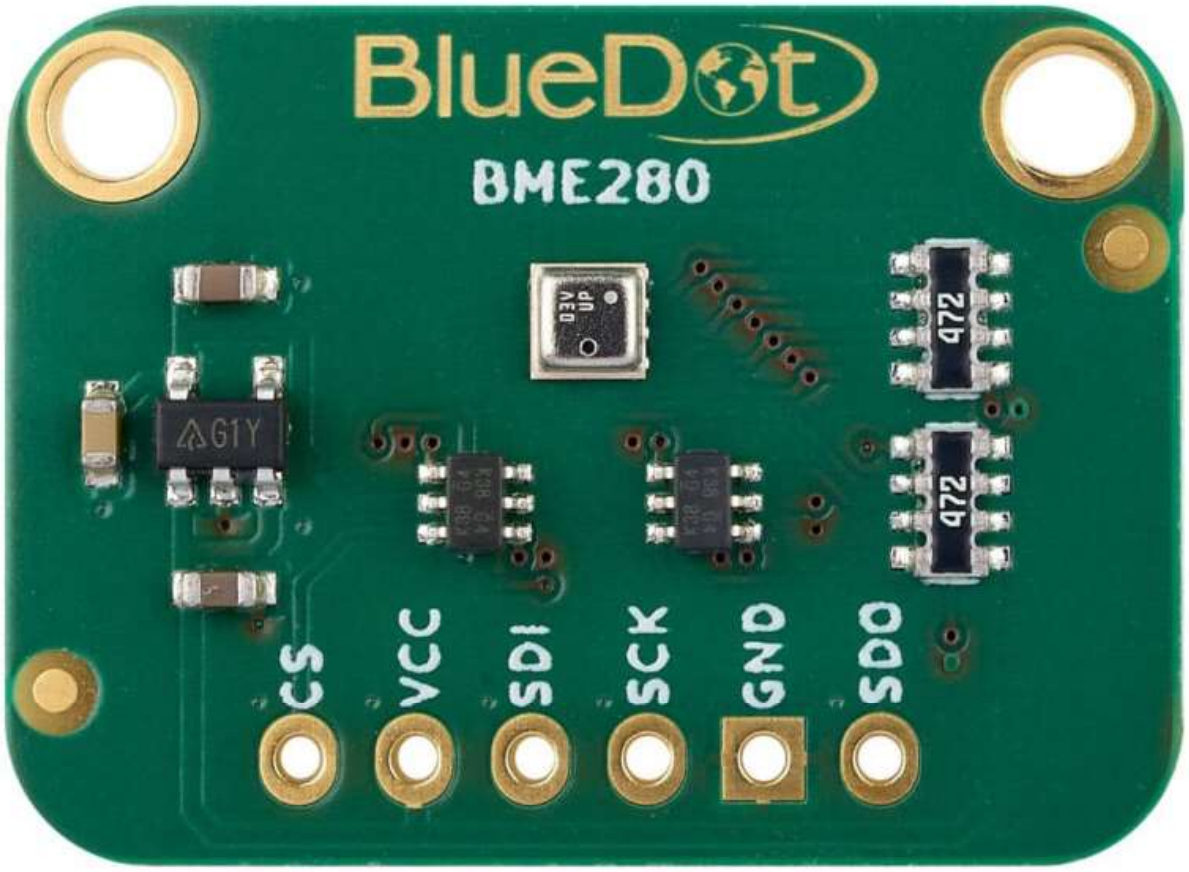
- 3D visualization.
- Microcontroller simulation.
- VSM studio which makes the compilation part a lot easier (not used here because this is not our goal. Our goal is to get used to the python language and other environments).

The main disadvantage of this software is its high price. Another problem is the limitation of libraries that we can use for raspberry Pi (Smbus Pygame Wiringpi RPi.GPIO Spidev Adafruit_GPIO Adafruit_I2C Adafruit_MCP230xx Adafruit_MCP3008 Adafruit_MotorHAT Adafruit_PCA9685 Adafruit_PureIO Automationhat Explorerhat ADS1x15 cap1xx grove_128_64_oled grove_rgb_lcd grovepi max31855 pcf8574 pcf8591 piglow sn3218).

III. Project components:

Figure 1: Project components.

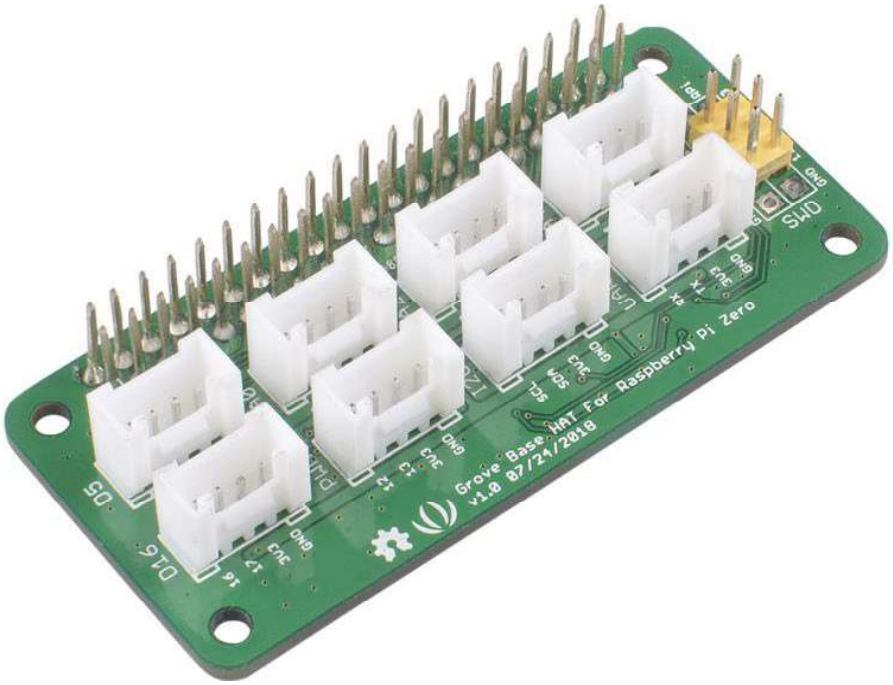
- **BME280:**



BME280

The BME280 is an integrated environmental sensor by Bosch that measures humidity, pressure and temperature.

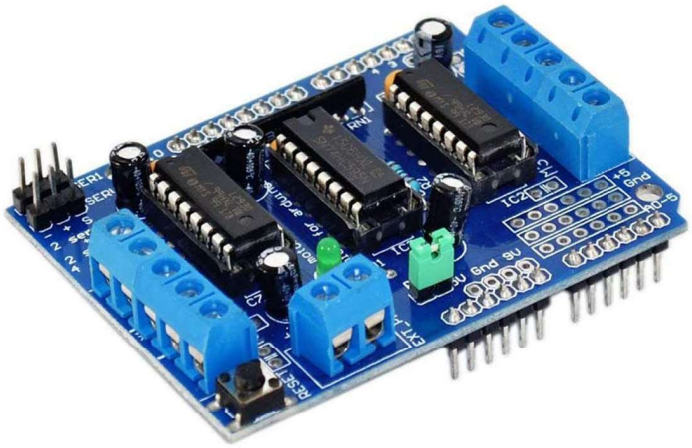
- **Grovehat (not used but prepared for future use):**



Grove Base HAT for Raspberry Pi

The Grove HAT for Raspberry Pi is an accessory board that simplifies the connection of Grove sensors and modules to a Raspberry Pi.

- **L293D DC motor driver:**



DC Motor Driver

The L293D is a 16 pin IC, with eight pins, on each side, dedicated to the controlling of a DC motor.

- **LM016L Display:**

LM016L

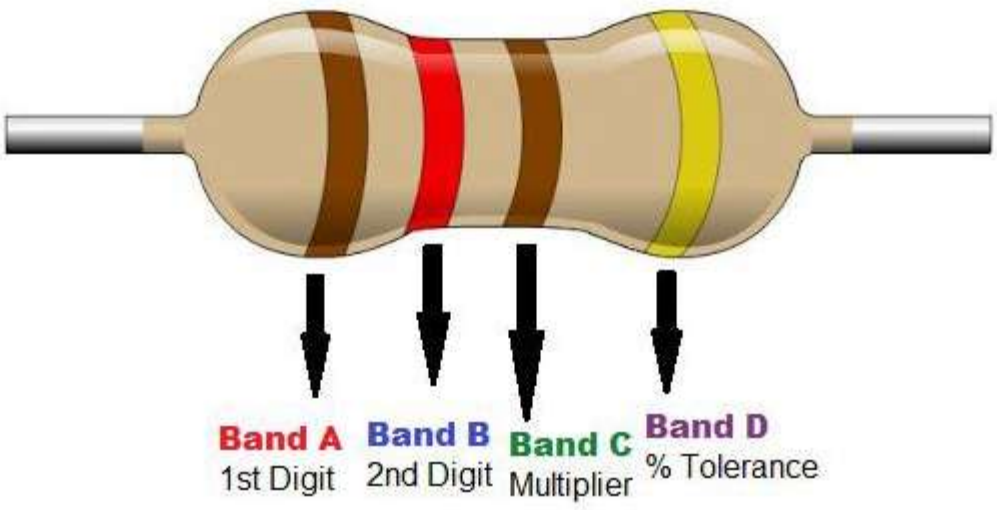
LCD1602, or 1602 character-type liquid crystal display, is a kind of dot matrix module to show letters, numbers, and characters.

- **DC Motor:**

DC Motor for a fan

A DC fan motor is an electric motor specifically designed for powering direct current (DC) fans. These motors are commonly used in a variety of applications, including greenhouses.

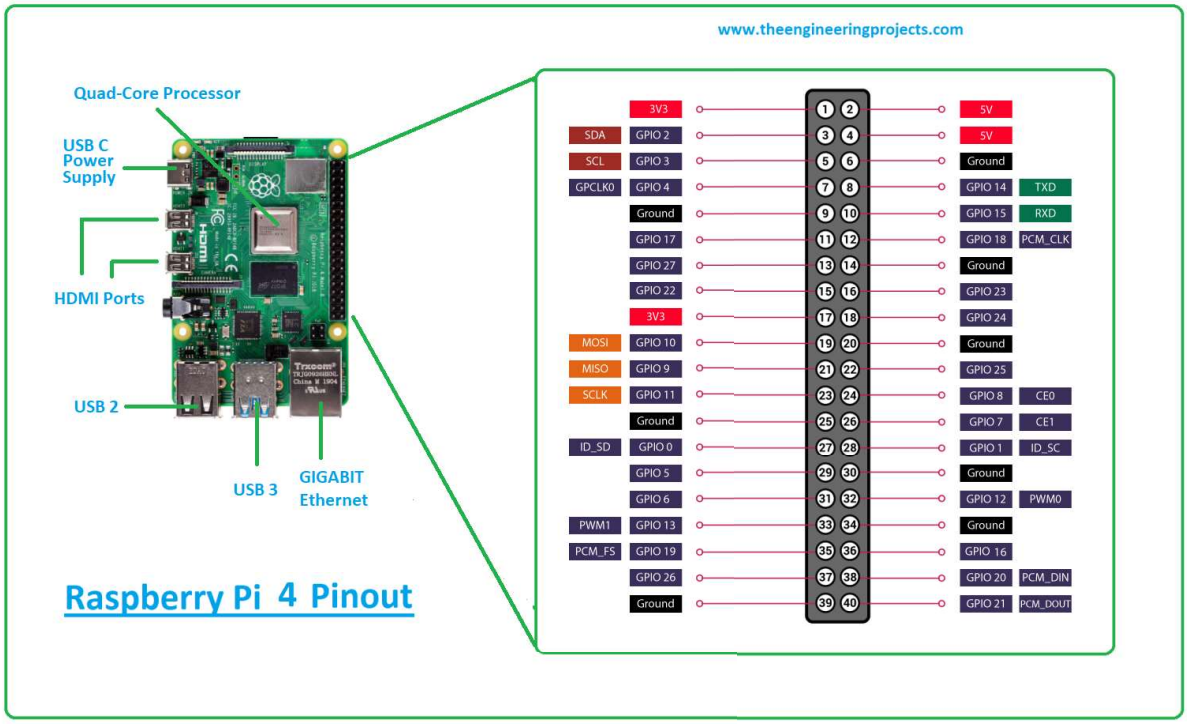
- **Pull-up:**



Pull-up

Pull-up resistors are resistors which are used to ensure that a wire is pulled to a high logical level in the absence of an input signal.[1]

- **Raspberry Pi 4:**



Raspberry Pi 4

The Raspberry Pi is a credit card sized computer that can connect to a huge variety of sensors and other modules like LCD displays, servos, and motors.

IV. Simulation:

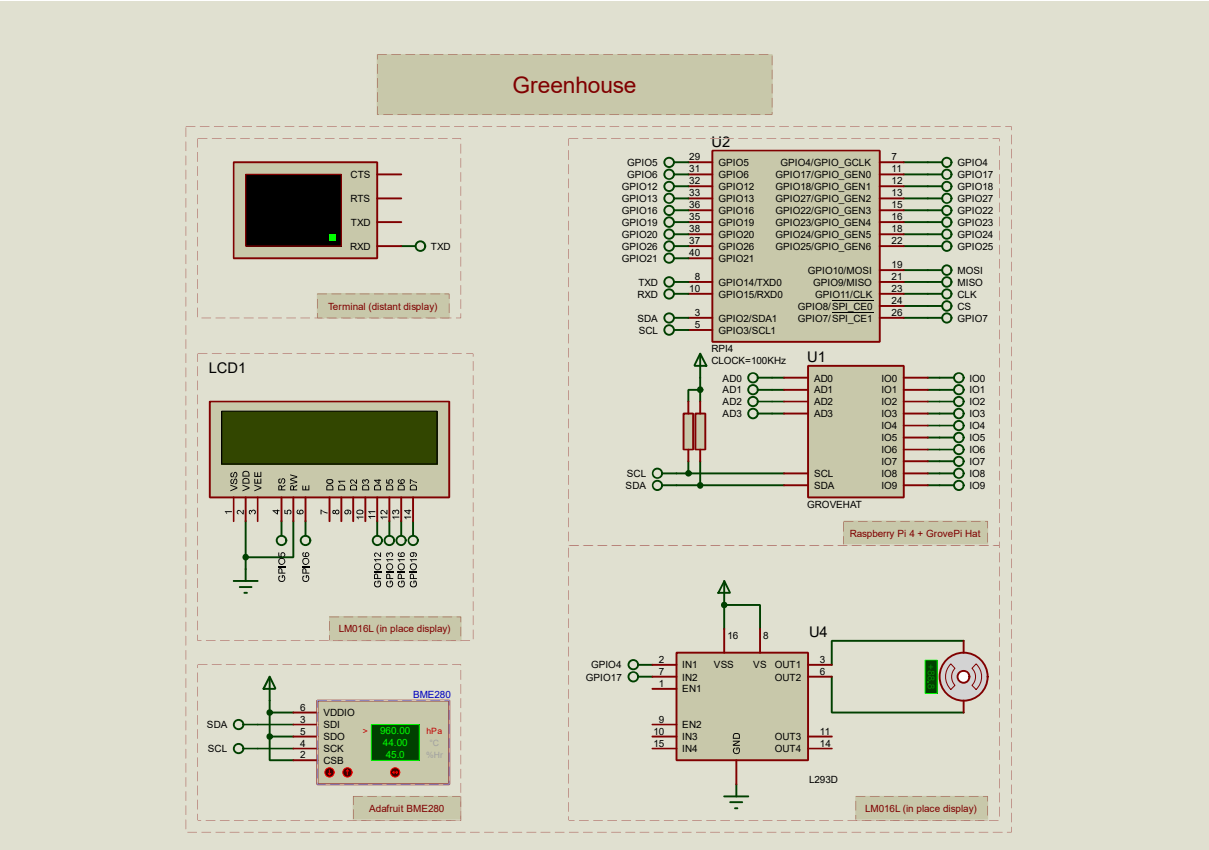


Figure 2: Project layout.

To simulate the circuit, we applied the connexions as shown above. Here is the Raspberry Pi source code to receive the data from the bme280, display the data and control the DC motor:

```
import RPi.GPIO as GPIO
import time
from adafruit_bme280 import basic as adafruit_bme280
import board
import pio
import cpu
import FileStore
import timer
import VFP
import Grove
import Ports
import InternetCommunication
import MQTT
import Generic


# Constants for the PID controller
setpoint = 30.0  # Desired temperature (adjust as needed)
Kp = 1.0  # Proportional gain
Ki = 0.1  # Integral gain
```

```
Kd = 0.01  # Derivative gain
integral = 0.0
prev_error = 0.0


# Define GPIO to LCD mapping
LCD_RS = 5
LCD_E  = 6
LCD_D4 = 12
LCD_D5 = 13
LCD_D6 = 16
LCD_D7 = 19

# Define some device constants
LCD_WIDTH = 16    # Maximum characters per line
LCD_CHR = True
LCD_CMD = False

LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line

# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005

# BME280 sensor setup
i2c = board.I2C()
bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c)



# GPIO setup
def GPIO_setup():
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)       # Use BCM GPIO numbers
    GPIO.setup(LCD_E, GPIO.OUT)  # E
    GPIO.setup(LCD_RS, GPIO.OUT) # RS
    GPIO.setup(LCD_D4, GPIO.OUT) # DB4
    GPIO.setup(LCD_D5, GPIO.OUT) # DB5
    GPIO.setup(LCD_D6, GPIO.OUT) # DB6
    GPIO.setup(LCD_D7, GPIO.OUT) # DB7
    GPIO.setup(4, GPIO.OUT)  # Motor direction control (set as output)
    GPIO.setup(17, GPIO.OUT)


def peripheral_setup():
    # Peripheral Constructors
    pio.cpu = cpu.CPU()
    pio.storage = FileStore.FileStore()
```

```
    pio.timer = timer.Timer()
    pio.server = VFP.VfpServer()
    pio.grove = Grove.Grove()
    pio.i2c = Ports.I2c()
    pio.spi = Ports.SPI()
    pio.uart = Ports.UART()
    pio.twitter = InternetCommunication.Twitter()
    pio.email = InternetCommunication.Email()
    pio.mqtt = MQTT.MQTT()
    pio.U3 = Generic.BME280()
    pio.storage.begin()
    pio.server.begin(0)
    pio.mqtt.begin("Raspberry Pi 4", "localhost", 1883, "", "")

def main():
    # Main program block
    peripheral_setup()
    GPIO_setup()
    pwm = GPIO.PWM(17, 1000)  # PWM signal to control motor speed
    pwm.start(100)  # Start with 100% duty cycle (maximum speed)
    lcd_init()
    setpoint = 30.0  # Desired temperature (adjust as needed)
    Kp = 1.0  # Proportional gain
    Ki = 0.1  # Integral gain
    Kd = 0.01  # Derivative gain
    integral = 0.0
    prev_error = 0.0
    # Motor speed constraints
    motor_speed_min = 0.1  # Minimum motor speed
    motor_speed_max = 100  # Maximum motor speed
    motor_acceleration = 10  # Maximum acceleration rate (adjust as needed)
    current_duty_cycle = 0.1  # Initialize duty cycle to 0

    while True:
        # Read the temperature
        temperature_c = bme280.temperature

        # Calculate the error
        error = setpoint - temperature_c

        # Calculate the integral term
        integral += error

        # Calculate the derivative term
        derivative = error - prev_error

        # Calculate the control output using the PID formula
        output = Kp * error + Ki * integral + Kd * derivative
```

```
        # Calculate the motor speed based on the control output
        target_duty_cycle = int(output)

        # Ensure that the target duty cycle is within the defined constraints
        target_duty_cycle = max(motor_speed_min, min(target_duty_cycle,
motor_speed_max))

        # If the current speed is below the threshold, accelerate to the
target duty cycle
        if current_duty_cycle < (100 - int(2.86 * (temperature_c -
setpoint))):
            current_duty_cycle += motor_acceleration
        else:
            current_duty_cycle = (100 - int(2.86 * (temperature_c -
setpoint)))

        # Ensure that the motor speed remains within the defined constraints
        current_duty_cycle = max(motor_speed_min, min(current_duty_cycle,
motor_speed_max))

        # Set the motor direction and PWM duty cycle
        if current_duty_cycle > 0:
            GPIO.output(4, GPIO.HIGH)  # Set direction to forward
            pwm.ChangeDutyCycle(current_duty_cycle)
        else:
            GPIO.output(4, GPIO.LOW)  # Set direction to reverse
            pwm.ChangeDutyCycle(0)  # Turn off the motor

        # Store the current error for the next iteration
        prev_error = error

        # Format the data for LCD display
        temperature_display = f"Temp: {temperature_c:.2f} C"
        speed_display = f"Speed: {(100 - current_duty_cycle) * 1.92:.2f}"

        # Print data to the virtual console
        console_data = f"Temperature: {temperature_c:.2f} C, {speed_display}"
        pio.uart.println(console_data)

        lcd_string(temperature_display, LCD_LINE_1)
        lcd_string(speed_display, LCD_LINE_2)

        time.sleep(2)

def lcd_init():
   # Initialise display
   lcd_byte(0x33,LCD_CMD) # 110011 Initialise
```

```
    lcd_byte(0x32,LCD_CMD) # 110010 Initialise
    lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
    lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
    lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size
    lcd_byte(0x01,LCD_CMD) # 000001 Clear display
    time.sleep(E_DELAY)

def lcd_byte(bits, mode):
    # Send byte to data pins
    # bits = data
    # mode = True  for character
    #        False for command

    GPIO.output(LCD_RS, mode) # RS

    # High bits
    GPIO.output(LCD_D4, False)
    GPIO.output(LCD_D5, False)
    GPIO.output(LCD_D6, False)
    GPIO.output(LCD_D7, False)
    if bits&0x10==0x10:
     GPIO.output(LCD_D4, True)
    if bits&0x20==0x20:
     GPIO.output(LCD_D5, True)
    if bits&0x40==0x40:
     GPIO.output(LCD_D6, True)
    if bits&0x80==0x80:
     GPIO.output(LCD_D7, True)

    # Toggle 'Enable' pin
    lcd_toggle_enable()

    # Low bits
    GPIO.output(LCD_D4, False)
    GPIO.output(LCD_D5, False)
    GPIO.output(LCD_D6, False)
    GPIO.output(LCD_D7, False)
    if bits&0x01==0x01:
     GPIO.output(LCD_D4, True)
    if bits&0x02==0x02:
     GPIO.output(LCD_D5, True)
    if bits&0x04==0x04:
     GPIO.output(LCD_D6, True)
    if bits&0x08==0x08:
     GPIO.output(LCD_D7, True)

    # Toggle 'Enable' pin
    lcd_toggle_enable()
```

```
def lcd_toggle_enable():
   # Toggle enable
   time.sleep(E_DELAY)
   GPIO.output(LCD_E, True)
   time.sleep(E_PULSE)
   GPIO.output(LCD_E, False)
   time.sleep(E_DELAY)

def lcd_string(message,line):
   # Send string to display

   message = message.ljust(LCD_WIDTH," ")

   lcd_byte(line, LCD_CMD)

   for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)

if __name__ == '__main__':
   try:
      main()
   except KeyboardInterrupt:
      pass
   finally:
      lcd_byte(0x01, LCD_CMD)
      GPIO.cleanup()
```

Figure 3: Raspberry Pi source code.

The Python script for a Raspberry Pi above uses a PID controller to maintain a specified temperature using a BME280 sensor. It adjusts a motor's speed based on the PID output to control the temperature. The LCD displays the current temperature and motor speed.

V. Conclusion :

In summary, our "Greenhouse Temperature Control System Using Raspberry Pi" project represents a harmonious fusion of technology and agriculture. By leveraging the capabilities of the Raspberry Pi, BME280 sensor, and PID controller, we have created a sophisticated system for maintaining greenhouse temperatures with precision, ensuring optimal conditions for plant growth. We've also utilized Proteus 8 for simulation, enabling us to plan and estimate component requirements and cost effectively by the built-in bill option. This project not only showcases the potential for technology in sustainable agriculture but also points towards a future of increased resource efficiency and improved crop yield.

References:

1. https://wokwi.com/projects/324274590349001300
2. https://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SystemModeling
3. Pull-up and Pull-down Resistors | Resistor Applications | Resistor Guide (eepower.com)

Please note that the submission is individual, collaboration is not permitted. In case of suspicion of cheating (e.g., identical answers/solutions), oral presentation is applied to those involved.

Grade

Grades are set according to:

F - 0-49 %

E - 50-59 %

D - 60-69 %

C - 70-79 %

B - 80-89 %

A – More than 90 %