VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Winter Semester 2024-25

Laboratory Assignment – 1 : Sensors &Actuators

Name- Harsh Prakash

Reg. No.- 22BCT0098

Course Code- BCSE311P

Class Number- VL2024250505942

Prof. Name - SIVANESAN S

Problem Statement:

Simulate, Demostrate and illustrate (graphically) the workings of various thermal sensors using C and python progamming tools

Scenario

An advanced industrial monitoring system utilizes several temperature sensors to monitor environmental conditions within a manufacturing facility. Given the differences in accuracy, drift rates, and response times, the readings from these sensors can show slight variations, which can impact decision-making. It is essential for the system to enhance decision-making capabilities in light of redundant or inconsistent readings while ensuring operational efficiency. The monitoring system must:

1. Systematically record temperature sensor measurements throughout a specified duration.

2. Examine sensor drift and redundancy to enhance data processing efficiency.

3. Activate cooling (Fan ON) or heating (Heater ON) in response to established temperature thresholds.

4. Present a visual depiction of trends and system performance.

Objective:

The objective of this assignment is to replicate the behavior of various temperature-sensing technologies in a real-world context and to evaluate their performance over a specified duration. Students will create C programs to produce sensor data, implement control mechanisms based on predefined thresholds, and utilize Python for the purposes of graphical representation and analysis.

Comprehensive Documentation for Thermal Sensors Simulation

1.Problem Statement

- **Clearly define the problem based on the given scenario.**
- **Explain the significance and real-world applications.**

DIGITAL ASSIGNMENT - 1

COMPREHENSIVE DOCUMENTATION

1. PROBLEM STATEMENT

DEFINATION

The objective of this assignment is to simulate, demonstrate and illustrate the behaviour of various temperature-sensing technologies in a real-world industrial monitoring scenario. This includes implementing sensor data simulation, accounting for drift and measurement errors, actuating control mechanisms based on predefined thresholds, and visualizing the result graphically.

Significance and Real-World Application

Temperature sensor are widely used in industries for monitoring and maintaining optimal environmental conditions. This project replicates the behavior of various sensors to:

* Evaluate their performance over time.
* Enhance decision-making in environments prone to data inconsistencies.
* Ensure operational efficiency in critical applications, such as manufacturing facilities.

2.Problem Analysis with Numerical Example

- **Provide a breakdown of sensor drift, accuracy errors, and control logic.**
- **Include numerical calculations illustrating sensor behavior over time.**

2. Problem Analysis with Numerical Example.

Sensor Characteristics

1. Thermoelectric Sensor: Accuracy: ± 2.5°C, Drift: 0.6 1. per year
2. Thermoresistive Sensory: Accuracy: ± 1°C, Drift: 0.6 1. per year
3. Thermo-Optical Sensor: Accuracy: 1.5°C, Drift: 0.6 1. per year
4. Thermo-Acoustic Sensor: Accuracy: 2°C, Drift: 0.6 1. per year

Example Calculation

Thermoresistive Sensor

For a thermoresistive sensor measuring a temperature of 25°C after 5 years:

* Drift factor = 1 + (0.006 * 5) = 1.03
* Resistance = RO * (1 + α * Temperature) * Drift Factor
* Using RO = 100 Ω and α = 0.00385
* Resistance = 100 * (1 + 0.00385 * 25) * 1.03

= 110.99 Ω

Thermoelectric Sensor

For a thermoelectric sensor measuring a temperature difference (ΔT) of 50°C after 5 years:

* Drift factor = $1 + (0.006 * 5) = 1.03$
* Voltage = Seebeck Coefficient * ΔT * Drift factor
* Seebeck Coefficient = 0.041 mV/°C (Given)
* Voltage = $0.041 * 50 * 1.03$
  $= 2.1115$ mV

Thermo - optical Sensor

For a thermo - optical sensor with a temperature of 100°C, emissivity of 0.98, and area of 0.0001 m^2 after 5 years:

* Drift factor = $1 + (0.006 * 5) = 1.03$
* Radiation Power = Stefan - Boltzmann Constant * Emissivity * Area * (Temperature + 273.15)^4 * Drift Factor
* Stefan - Boltzmann = 5.67×10^{-8} W/m^2K^4
* Radiation Power = $5.67 * 0.98 * 0.0001 * (100 + 273.15)^4 * 1.03$
  $= 5.85 \times 10^{-3}$ W

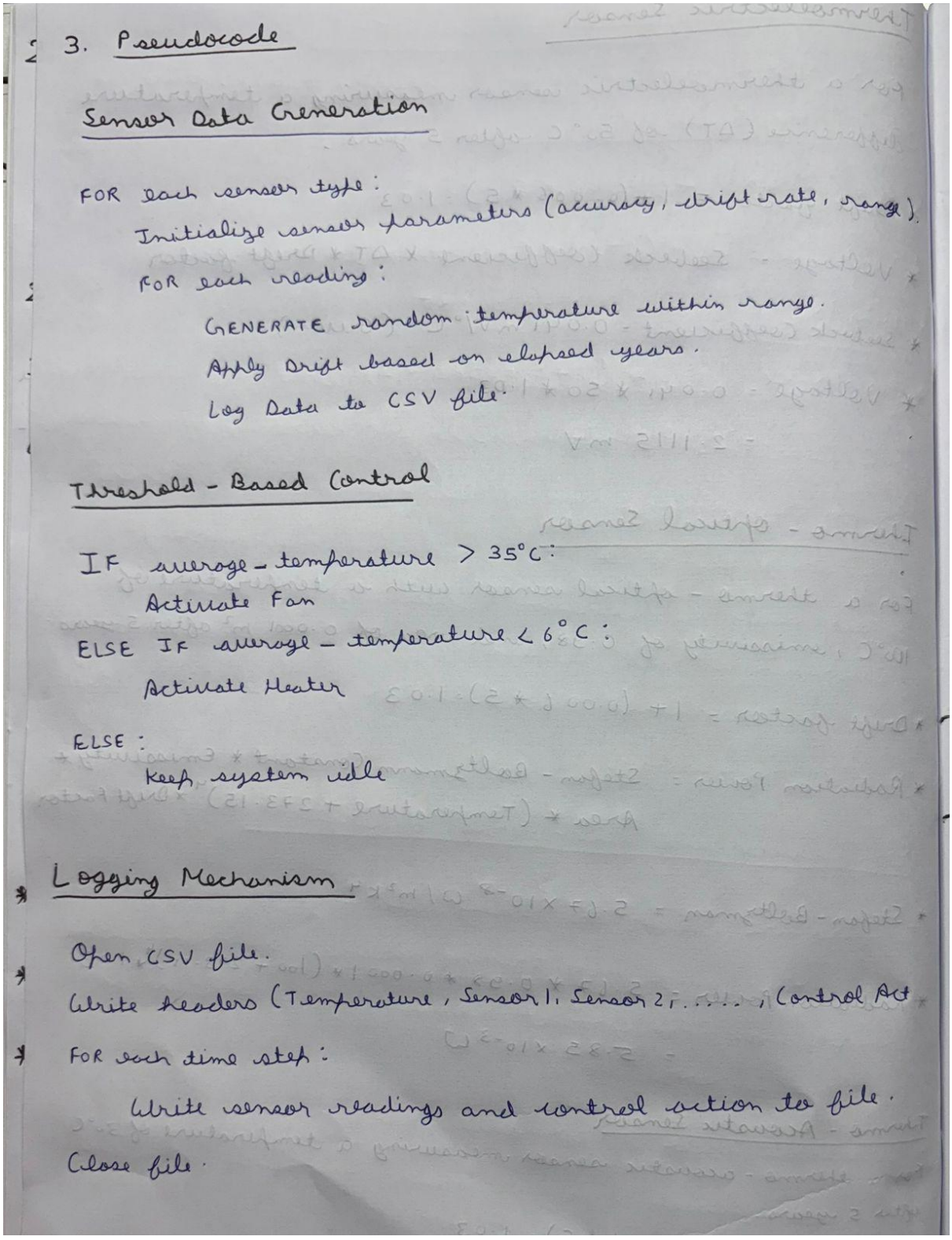
Thermo - Acoustic Sensor

For a thermo - acoustic sensor measuring a temperature of 30°C after 5 years.

* Drift factor = $1 + (0.006 * 5) = 1.03$
* Speed of Sound = Base Speed + (Temperature Coefficient * Temperature) * Drift factor
* Base speed = 331.3 m/s & Temperature Coefficient = 0.606 m/s/°C
* Speed of Sound = $(331.3 + 0.606 * 30) * 1.03$
  $= 352.08$ m/s

3.Pseudocode

• Present structured pseudocode explaining sensor data generation, threshold-based control, and logging mechanisms.



3. Pseudocode

Sensor Data Generation

```
FOR each sensor type:
    Initialize sensor parameters (accuracy, drift rate, range).
    FOR each reading:
        GENERATE random temperature within range.
        Apply drift based on elapsed years.
        Log Data to CSV file.
```

Threshold - Based Control

```
IF average_temperature > 35°C:
    Activate Fan
ELSE IF average_temperature < 6°C:
    Activate Heater
ELSE:
    Keep system idle
```

Logging Mechanism

```
Open CSV file.
Write headers (Temperature, Sensor 1, Sensor 2, ..... , Control Act
FOR each time step:
    Write sensor readings and control action to file.
Close file.
```

4.Implementation and Test Cases

- **Include well-commented C code for sensor data simulation.**

thermoresistive.h

```c
#include <math.h>
#include <stdlib.h>

#ifndef SENSOR_H
#define SENSOR_H

// RTD constants for a typical PT100 sensor
#define RO 120.0
#define ALPHA 0.0054

// Steinhart-Hart coefficients for a thermistor
#define A 1.009249522e-03
#define B 2.378405444e-04
#define C 2.019202697e-07

// Drift parameters
#define DRIFT_RATE 0.006
#define SENSOR_ERROR_THERMORESSITIVE 1.0

// Function to calculate resistance for RTD given a temperature
double cal_res_rtd(double temperature, int years) {
    double drift_factor = 1 + (DRIFT_RATE * years);
    double resistance = RO * (1 + ALPHA * temperature) * drift_factor;
    double error = ((rand() % 2001) - 1000) / 1000.0 * SENSOR_ERROR_THERMORESSITIVE;
    return resistance + error;
}

// Function to calculate resistance for a thermistor given a temperature
double cal_res_thermistor(double temperature, int years) {
    double inv_T = 1.0 / (temperature + 273.15);
    double R = (inv_T - A) / (B + C * pow((inv_T - A), 2));
    double resistance = exp(R);

    double drift_factor = 1 + (DRIFT_RATE * years);
    double error = ((rand() % 2001) - 1000) / 1000.0 * SENSOR_ERROR_THERMORESSITIVE;

    return resistance * drift_factor + error;
}

#endif
```

thermoelectric.h

```c
#include <math.h>
#include <stdlib.h>

#ifndef THERMOELECTRIC_H
#define THERMOELECTRIC_H

// Seebeck Coefficient
#define SEEBECK_COEFF 0.041

// Drift parameters
#define DRIFT_RATE 0.006
#define SENSOR_ERROR_THERMOELECTRIC 2.5

// Function to calculate thermoelectric voltage
double cal_thermoelectric_voltage(double hot_temp, double cold_temp, int years) {
    double delta_T = hot_temp - cold_temp;

    // Apply drift over time
    double drift_factor = 1 + (DRIFT_RATE * years);
    double voltage = SEEBECK_COEFF * delta_T * drift_factor;

    // Apply random noise (±2.5°C)
    double error = ((rand() % 5001) - 2500) / 1000.0;
    voltage += SEEBECK_COEFF * error;

    return voltage;
}

#endif
```

thermo_optical.h

```c
#include <math.h>
#include <stdlib.h>

#ifndef THERMO_OPTICAL_H
#define THERMO_OPTICAL_H

// Stefan-Boltzmann constant
#define STEFAN_BOLTZMANN 5.670e-8

// Default emissivity
#define DEFAULT_EMISSIVITY 0.43

// Default sensor surface area (m²)
#define DEFAULT_AREA 0.0012

// Drift parameters
#define DRIFT_RATE 0.006
#define SENSOR_ERROR_THERMOOPTICAL 1.5

// Function to calculate emitted power based on temperature
double cal_thermal_radiation(double temperature, int years, double emissivity, double area) {
    double temp_K = temperature + 273.15;
    double drift_factor = 1 + (DRIFT_RATE * years);
    double power = STEFAN_BOLTZMANN * emissivity * area * pow(temp_K, 4) * drift_factor;
    double error = ((rand() % 3001) - 1500) / 1000.0;
    temp_K += error;
    power = STEFAN_BOLTZMANN * emissivity * area * pow(temp_K, 4) * drift_factor;
    return power;
}

#endif
```

thermo_acoustic.h

```c
#include <math.h>
#include <stdlib.h>

#ifndef THERMO_ACOUSTIC_H
#define THERMO_ACOUSTIC_H

// Speed of sound constants
#define BASE_SPEED 331.3
#define TEMP_COEFF 0.646

// Drift parameters
#define DRIFT_RATE 0.006
#define SENSOR_ERROR_THERMOACOUSTIC 2.0

// Function to calculate speed of sound based on temperature
double cal_sound_speed(double temperature, int years) {
    // Apply drift over time
    double drift_factor = 1 + (DRIFT_RATE * years);
    double speed = (BASE_SPEED + TEMP_COEFF * temperature) * drift_factor;

    // Apply random noise (±2°C)
    double error = ((rand() % 4001) - 2000) / 1000.0;
    speed = (BASE_SPEED + TEMP_COEFF * (temperature + error)) * drift_factor;

    return speed; // Output in m/s
}

#endif
```

22BCT0098_main.c

22BCT0098_main.c > ...

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include "thermoresistive.h"
#include "thermo_acoustic.h"
#include "thermoelectric.h"
#include "thermo_optical.h"

#define NUM_READINGS 1200

void operation(double temperature, char *status) {
    if (temperature > 35) {
        strcpy(status, "Fan On");
    } else if (temperature < 6) {
        strcpy(status, "Heater On");
    } else {
        strcpy(status, "Stable");
    }
}

int main() {
    int choice, years;
    double temperature, sensor_value;
    char filename[50], status[20];
    FILE *logfile;

    srand(time(NULL));

    while (1) {
        printf("HARSH PRAKASH 22BCT0098\n");
        printf("Select Sensor Type:\n");
        printf("1. Thermoresistive (RTD/Thermistor)\n");
        printf("2. Thermoelectric (Thermocouple)\n");
        printf("3. Thermo-Optical (Infrared Sensor)\n");
        printf("4. Thermo-Acoustic (Ultrasonic-based)\n");
        printf("5. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        if (choice == 5) {
            printf("Exiting program...\n");
            break;
        }

        printf("Enter years of operation: ");
        scanf("%d", &years);

        switch (choice) {
            case 1:
                snprintf(filename, sizeof(filename), "thermoresistive_log.csv");
                logfile = fopen(filename, "w");
                fprintf(logfile, "Sensor ID, Temperature, Resistance, Status\n");
                for (int i = 0; i < NUM_READINGS; i++) {
                    temperature = (rand() % 6001) / 100.0;
                    sensor_value = cal_res_rtd(temperature, years);
                    operation(temperature, status);
                    fprintf(logfile, "RTD_%d, %.2f, %.6f, %s\n", i + 1, temperature, sensor_value, status);
                }
```

```
                fprintf(logfile, "RTD_%d, %.2f, %.6f, %s\n", i + 1, temperature, sensor_value, status);
            }
            fclose(logfile);
            break;
        case 2:
            snprintf(filename, sizeof(filename), "thermoelectric_log.csv");
            logfile = fopen(filename, "w");
            fprintf(logfile, "Sensor ID, Temperature, Voltage, Status\n");
            for (int i = 0; i < NUM_READINGS; i++) {
                temperature = (rand() % 6001) / 100.0;
                sensor_value = cal_thermoelectric_voltage(temperature, temperature - 10, years);
                operation(temperature, status);
                fprintf(logfile, "TC_%d, %.2f, %.6f, %s\n", i + 1, temperature, sensor_value, status);
            }
            fclose(logfile);
            break;
        case 3:
            snprintf(filename, sizeof(filename), "thermo_optical_log.csv");
            logfile = fopen(filename, "w");
            fprintf(logfile, "Sensor ID, Temperature, Power, Status\n");
            for (int i = 0; i < NUM_READINGS; i++) {
                temperature = (rand() % 6001) / 100.0;
                sensor_value = cal_thermal_radiation(temperature, years, DEFAULT_EMISSIVITY, DEFAULT_AREA);
                operation(temperature, status);
                fprintf(logfile, "IR_%d, %.2f, %.6f, %s\n", i + 1, temperature, sensor_value, status);
            }
            fclose(logfile);
            break;
        case 4:
            snprintf(filename, sizeof(filename), "thermo_acoustic_log.csv");
            logfile = fopen(filename, "w");
            fprintf(logfile, "Sensor ID, Temperature, Speed, Status\n");
            for (int i = 0; i < NUM_READINGS; i++) {
                temperature = (rand() % 6001) / 100.0;
                sensor_value = cal_sound_speed(temperature, years);
                operation(temperature, status);
                fprintf(logfile, "UA_%d, %.2f, %.6f, %s\n", i + 1, temperature, sensor_value, status);
            }
            fclose(logfile);
            break;
        default:
            printf("Invalid choice!\n");
            continue;
    }
    printf("Sensor data logged in %s\n\n", filename);
  }
  return 0;
}
```

- **Display sample output logs and describe different test scenario**

```
PS C:\Users\harsh\Documents\DA\SENSOR> gcc 22BCT0098_main.c
PS C:\Users\harsh\Documents\DA\SENSOR> .\a.exe
HARSH PRAKASH 22BCT0098
Select Sensor Type:
1. Thermoresistive (RTD/Thermistor)
2. Thermoelectric (Thermocouple)
3. Thermo-Optical (Infrared Sensor)
4. Thermo-Acoustic (Ultrasonic-based)
5. Exit
Enter choice: 1
Enter years of operation: 2
Sensor data logged in thermoresistive_log.csv

HARSH PRAKASH 22BCT0098
Select Sensor Type:
1. Thermoresistive (RTD/Thermistor)
2. Thermoelectric (Thermocouple)
3. Thermo-Optical (Infrared Sensor)
4. Thermo-Acoustic (Ultrasonic-based)
5. Exit
Enter choice: 2
Enter years of operation: 3
Sensor data logged in thermoelectric_log.csv

HARSH PRAKASH 22BCT0098
Select Sensor Type:
1. Thermoresistive (RTD/Thermistor)
2. Thermoelectric (Thermocouple)
3. Thermo-Optical (Infrared Sensor)
4. Thermo-Acoustic (Ultrasonic-based)
5. Exit
Enter choice: 3
Enter years of operation: 4
Sensor data logged in thermo_optical_log.csv

HARSH PRAKASH 22BCT0098
Select Sensor Type:
1. Thermoresistive (RTD/Thermistor)
2. Thermoelectric (Thermocouple)
3. Thermo-Optical (Infrared Sensor)
4. Thermo-Acoustic (Ultrasonic-based)
5. Exit
Enter choice: 4
Enter years of operation: 2
Sensor data logged in thermo_acoustic_log.csv
```

5. Screenshots of Output with Different Test Cases

- **Capture and annotate sensor reading logs and control action changes.**

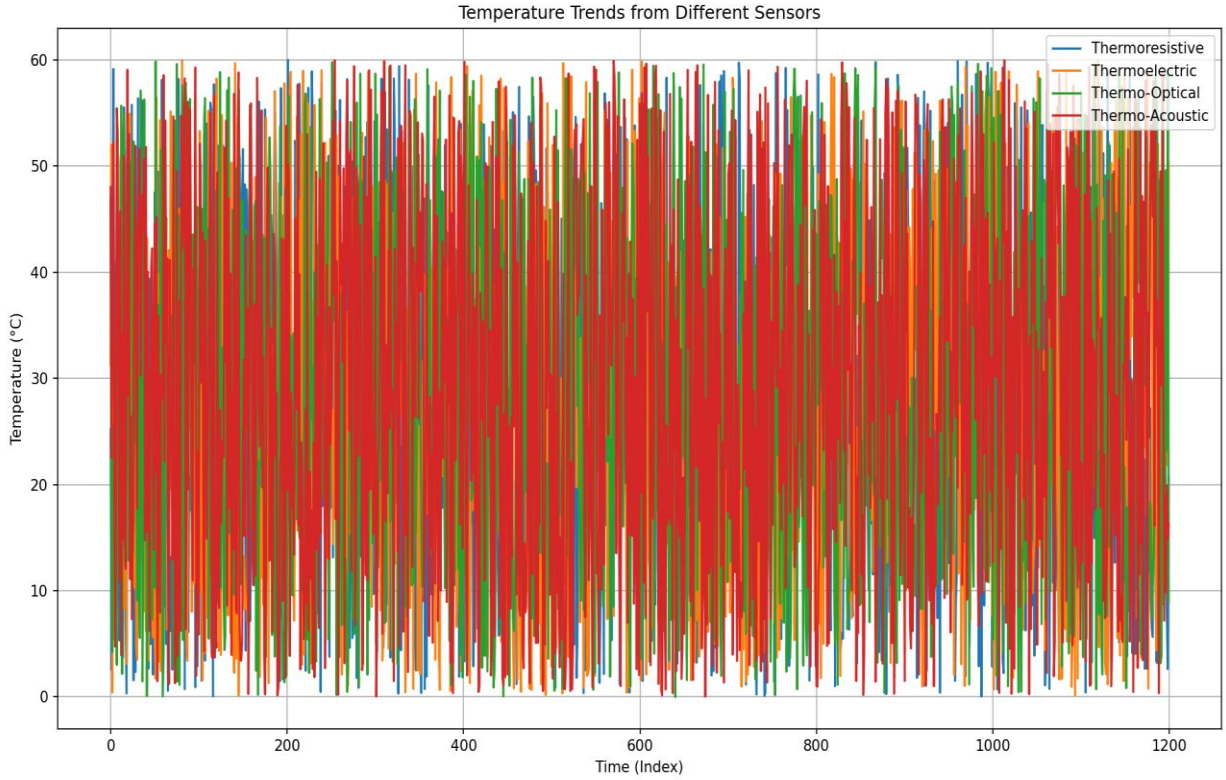
thermoresistive_log.csv

```
Sensor ID, Temperature, Resistance, Status
RTD_1, 25.18, 137.552440, Stable
RTD_2, 2.54, 122.506671, Heater On
RTD_3, 36.85, 145.771346, Fan On
RTD_4, 59.14, 160.207593, Fan On
RTD_5, 41.15, 149.367182, Fan On
RTD_6, 39.74, 147.752538, Fan On
RTD_7, 26.93, 138.349048, Stable
RTD_8, 22.63, 135.440211, Stable
RTD_9, 41.11, 147.528951, Fan On
RTD_10, 15.84, 130.837492, Stable
RTD_11, 5.31, 125.914171, Heater On
RTD_12, 13.41, 130.355956, Stable
RTD_13, 45.35, 150.379442, Fan On
RTD_14, 49.20, 153.683179, Fan On
RTD_15, 3.72, 123.597487, Heater On
RTD_16, 55.52, 157.633684, Fan On
RTD_17, 50.33, 154.118206, Fan On
RTD_18, 7.33, 125.599838, Stable
RTD_19, 0.33, 120.852406, Heater On
RTD_20, 20.45, 133.874619, Stable
RTD_21, 44.20, 149.859299, Fan On
RTD_22, 17.54, 132.456311, Stable
RTD_23, 13.86, 130.587055, Stable
RTD_24, 20.10, 133.772098, Stable
RTD_25, 55.76, 157.196070, Fan On
RTD_26, 18.33, 132.625374, Stable
RTD_27, 1.56, 122.014011, Heater On
RTD_28, 35.86, 145.421127, Fan On
RTD_29, 2.35, 122.253074, Heater On
RTD_30, 28.18, 139.242768, Stable
RTD_31, 40.03, 146.833713, Fan On
RTD_32, 56.38, 159.080651, Fan On
RTD_33, 33.08, 142.665070, Stable
RTD_34, 5.56, 125.723115, Heater On
RTD_35, 32.60, 142.030298, Stable
RTD_36, 56.18, 157.636496, Fan On
RTD_37, 15.78, 131.125145, Stable
RTD_38, 42.39, 149.305345, Fan On
RTD_39, 9.30, 127.122717, Stable
RTD_40, 44.82, 150.698880, Fan On
RTD_41, 22.13, 136.103323, Stable
RTD_42, 15.52, 132.258644, Stable
RTD_43, 26.65, 138.087430, Stable
RTD_44, 2.67, 123.905922, Heater On
RTD_45, 39.37, 147.724901, Fan On
RTD_46, 4.98, 125.523764, Heater On
RTD_47, 18.91, 134.533724, Stable
RTD_48, 12.03, 128.947985, Stable
RTD_49, 2.10, 122.640130, Heater On
RTD_50, 27.32, 139.544800, Stable
RTD_51, 47.51, 152.840918, Fan On
RTD_52, 34.73, 143.600100, Stable
RTD_53, 38.32, 146.497336, Fan On
RTD_54, 32.91, 143.797588, Stable
RTD_55, 49.45, 154.578123, Fan On
RTD_56, 14.65, 131.656118, Stable
RTD_57, 10.66, 129.292572, Stable
RTD_58, 16.46, 131.897073, Stable
```

thermoelectric_log.csv

```
Sensor ID, Temperature, Voltage, Status
TC_1, 31.24, 0.512049, Stable
TC_2, 51.98, 0.420783, Fan On
TC_3, 0.43, 0.373428, Heater On
TC_4, 41.18, 0.433534, Fan On
TC_5, 5.46, 0.389541, Heater On
TC_6, 16.44, 0.485645, Stable
TC_7, 7.74, 0.479454, Stable
TC_8, 15.82, 0.321932, Stable
TC_9, 44.70, 0.470557, Fan On
TC_10, 41.79, 0.367155, Fan On
TC_11, 19.34, 0.447146, Stable
TC_12, 50.44, 0.407212, Fan On
TC_13, 51.33, 0.515370, Fan On
TC_14, 17.37, 0.392575, Stable
TC_15, 43.38, 0.319226, Fan On
TC_16, 8.27, 0.393026, Stable
TC_17, 28.75, 0.335995, Stable
TC_18, 37.13, 0.513115, Fan On
TC_19, 3.11, 0.337020, Heater On
TC_20, 19.18, 0.321727, Stable
TC_21, 19.83, 0.381915, Stable
TC_22, 34.83, 0.332141, Stable
TC_23, 54.91, 0.499093, Fan On
TC_24, 25.88, 0.494870, Stable
TC_25, 15.79, 0.400283, Stable
TC_26, 21.34, 0.326442, Stable
TC_27, 22.28, 0.351944, Stable
TC_28, 10.29, 0.445219, Stable
TC_29, 50.80, 0.519716, Fan On
TC_30, 4.74, 0.413813, Heater On
TC_31, 3.49, 0.476256, Heater On
TC_32, 18.78, 0.353256, Stable
TC_33, 21.38, 0.359488, Stable
TC_34, 14.17, 0.414592, Stable
TC_35, 49.39, 0.428368, Fan On
TC_36, 13.30, 0.489458, Stable
TC_37, 20.08, 0.519675, Stable
TC_38, 14.26, 0.369205, Stable
TC_39, 8.06, 0.357069, Stable
TC_40, 43.30, 0.420578, Fan On
TC_41, 6.22, 0.428778, Stable
TC_42, 43.24, 0.380439, Fan On
TC_43, 4.68, 0.398069, Heater On
TC_44, 3.46, 0.379209, Heater On
TC_45, 13.90, 0.444973, Stable
TC_46, 24.53, 0.419389, Stable
TC_47, 8.07, 0.463136, Stable
TC_48, 35.25, 0.395732, Fan On
TC_49, 5.62, 0.449975, Heater On
TC_50, 19.18, 0.326032, Stable
TC_51, 4.74, 0.331649, Heater On
TC_52, 1.71, 0.320661, Heater On
TC_53, 56.45, 0.440545, Fan On
TC_54, 26.94, 0.517092, Stable
TC_55, 39.80, 0.326073, Fan On
TC_56, 3.31, 0.495239, Heater On
TC_57, 54.30, 0.391632, Fan On
```

• Show graphs illustrating temperature variations and threshold-based actions.



---

6. Results and Analysis

• Discuss observed trends from the generated data.

• Compare different sensor types based on accuracy and drift characteristics.

• Analyze system efficiency in handling redundant sensor readings.

6. Result and Analysis

Observed Trends

- Thermoelectric Sensor : Higher error margins but consistent drift.
- Thermoresistive Sensor : Most accurate with minimal drift.
- Thermo-Optical Sensor : Drift affects reliability over extended periods.
- Thermo-Acoustic Sensor : Moderate Accuracy but effective in specific scenarios.

COMPARISON OF SENSOR TYPES

Sensor type	Accuracy (°C)	Drift Rate (°C/year)	NOTES
Thermoelectric	± 2.5	0.6	High error, simple design
Thermoresistive	± 1	0.6	Best overall performance
Thermo-Optical	± 1.5	0.6	Sensitive to emissivity
Thermo-Acoustic	± 2	0.6	Best for sound-based setup

System Efficiency

* Redundant readings are averaged to enhance reliability
* Control mechanisms effectively mitigate temperature deviations.