HARDWARE PROJECT EE-1030
DIGITAL THERMOMETER

KARTIK LAHOTI - EE 25 BTECH11032
SUBHODEEP CHAKRABORTHY - EE 25 BTECH11OSS

31st October 2025

Contents

OBJECTIVE

To design, implement and validate a digital thermometer, using a PT-100 Resistance Temperature Detector (RTD) for measurement, an Arduino microcontroller for processing and a 16x2 LCD for output.

This experiment uses the least squares approach to estimate a model for ambient temperature versus voltage.

THEORY

Resistance of a PT-100 varies with temperature as given by the Callendar-Van Dusen Equation:

$$R(T) = R_0(1 + A \times T + B \times T^2) \tag{1}$$

Values of A and B can be found experimentally. When placed in a voltage divider circuit, the voltage across the PT-100 can be measured and used to infer temperature as per the above relationship.

APPARATUS

1. Arduino UNO
2. PT-100 RTD
3. Lab-grade Thermometer
4. Electric Kettle
5. Jumper cables
6. Breadboard
7. Android Phone
8. Resistors of at most 5% tolerance (130Ω used)
9. Push Button
10. Potentiometer
11. 16x2 LCD
12. JST XH 2-pin wire connector
13. Arduino Cable
14. USB-A to USB-C convertor

CIRCUIT REPRESENTATION

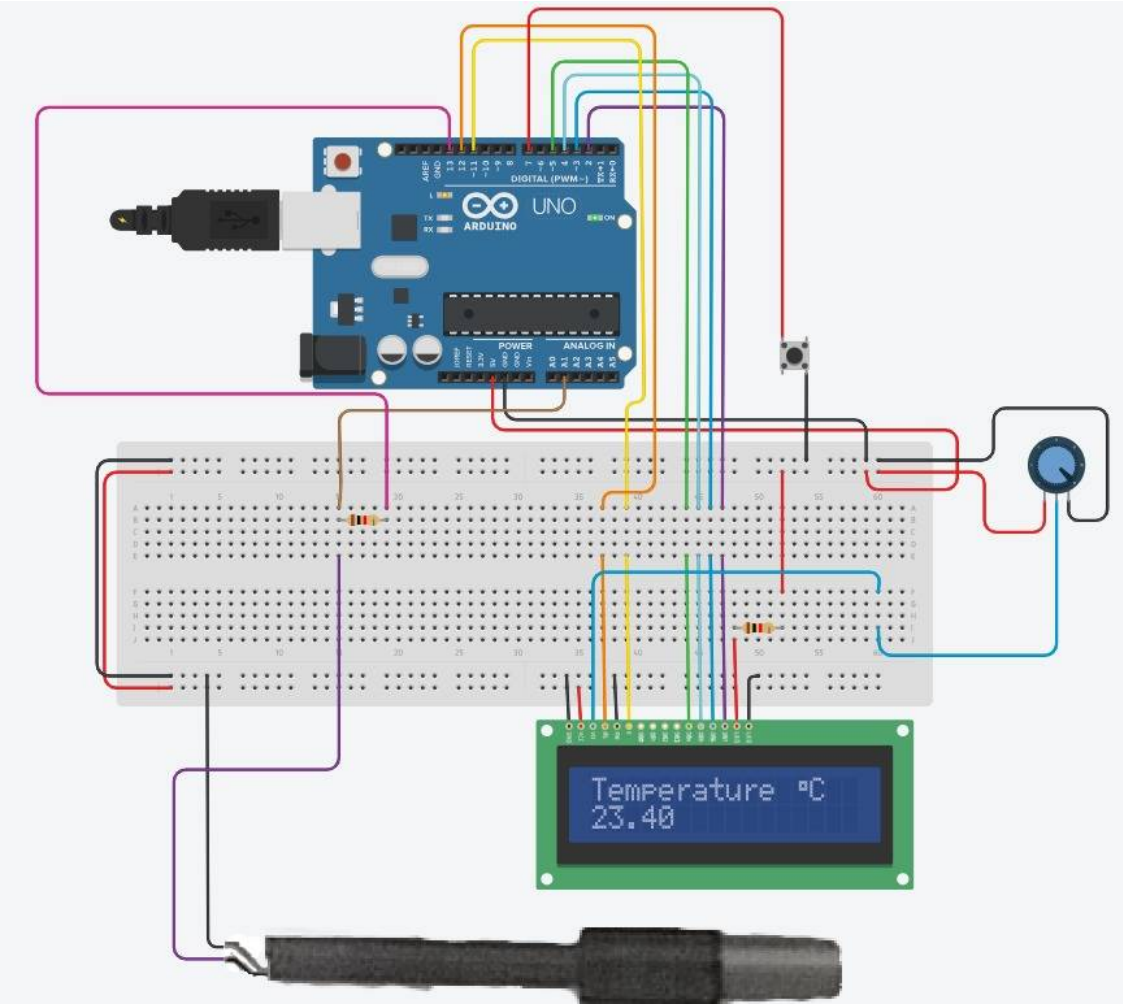

Figure 1: Circuit Schematic

1 PROCEDURE

1.1 Circuit Assembly

Make connections as per the circuit schematic. Note that while the push button in the schematic has 4 terminals, same connections are applicable for 2-terminal push-buttons. Ensure that a resistor of appropriate denomination is connected to the LED backlight of the LCD.

To choose the value of the resistor in series with the PT-100, our aim was to maximize the sensitivity of the variation of voltage across the PT-100 with temperature. Taking V as the voltage source, R as value of resistance of PT-100, x as resistance in series, we have voltage across PT-100 as:

$$V_R = V\frac{R}{x+R} \tag{2}$$

Sensitivity $(S) = \frac{dV_R}{dR}$. Using the quotient rule:

$$S = \frac{d}{dR}\left[V\frac{R}{x+R}\right] = V\left[\frac{(1)(x+R) - R(1)}{(x+R)^2}\right] = \frac{Vx}{(x+R)^2} \tag{3}$$

To maximise sensitivity with respect to the series resistor x, we set $\frac{dS}{dx} = 0$:

$$\frac{dS}{dx} = \frac{d}{dx}\left[V\frac{x}{(x+R)^2}\right] = V\left[\frac{(1)(x+R)^2 - x(2(x+R))}{(x+R)^4}\right] = 0 \tag{4}$$

$$V\left[\frac{(x+R) - 2x}{(x+R)^3}\right] = 0 \tag{5}$$

$$V\frac{R-x}{(x+R)^3} = 0 \tag{6}$$

$$\Rightarrow x = R \tag{7}$$

Since R varies with temperature, we take value of x as the median of R's expected range, i.e.; $x = 130\Omega$.

1.2 Platformio Setup

In order to work with the Arduino on an Android phone, we need to setup platformio and Arduino Droid on Termux.

1.2.1 Platformio

1. Login to `proot-distro`.
2. Download and run the platformio installer.

```
wget -O get-platformio.py https://raw.githubusercontent.com/platformio/platformio-core-installer/master/
    get-platformio.py
python3 get-platformio.py
```

3. Follow on-screen steps to install pio to PATH, if necessary.
4. Create a folder `PT100` in `/sdcard` path and initialize the project.

```
mkdir PT100
pio project init --board uno --ide vscode --project-dir PT100
```

5. Add code in `src/main.cpp` in project folder.

```
nvim PT100/src/main.cpp
```

6. Compile file.

```
pio run
```

1.2.2 Arduino Droid

1. Install the Apkpure app on device.
2. Install Arduino Droid version 6.3.0 from the Apkpure app (newer versions do not work reliably on recent Android versions).
3. In Arduino Droid, click : > Actions > Upload > Upload precompiled.
4. Navigate to project folder and upload `.pio/build/uno/firmware.hex` with Arduino connected.

1.3 Taking Readings

Dip both PT-100 and thermometer into kettle. Power on the devices and take readings of voltage-temperature mappings for different temperatures using the kettle.

1.4 Training of Model

Using the collected data, apply least squares as per the Python code.

1.5 Validation of Results

Take another set of readings and map for temperature versus temperature. Use the Python code to generate error plot.

1.6 Error Analysis

This implementation results in the following measures of error:

1. Mean: 1.358°C
2. Median: 1.595°C
3. Mode: 1.81°C

1.7 Observations

LCD could display expected values. Potentiometer could adjust contrast. Push button switched between $^{\circ}$C and $^{\circ}$F reliably. Button could also switch to display voltage reading on LCD correctly.

2 CALIBRATION MODEL

The collected data consists of values of voltage V across the PT-100 for temperature T over a large range of known temperatures. In this data, we know that maximum error is present in the values of V, and the goal is to find a predictive model for T in terms of V.

Given this and the hardware-level computational constraints of the Arduino, the best modelling method that balances accuracy as well as computational complexity is the least squares method.

2.1 Least Squares Method

An empirical model for the data can be formed using least squares. The basic physical relation is given by the Callendar-Van Dusen equation:

$$R(T) = A + BT + CT^2 \tag{8}$$

The above expression which gives the resistance of pt-100 sensor at a temperature T, can be used to find voltage across pt-100 in the voltage divider circuit used.

$$V_{Pt-100} = \frac{R(T) \times V_s}{R_s + R(T)} \tag{9}$$

where V_s (5V) is source voltage, R_s (130Ω) is resistance in series.

Analysing the graph of V_{PT-100} in the relevant temperature range, we see that it approximates to a parabola.

$$V_{PT-100} = \alpha + \beta T + \gamma T^2 \tag{10}$$

Since the greater amount of errors is in the measurements of V, normally applying least squares on Equation (10) would be preferred. However, since the intention is to build a predictive model for T, the better choice is applying least squares on equation (11):

$$T(V) = \eta_0 + \eta_1 V + \eta_2 V^2 \tag{11}$$

where V is voltage across pt-100. Equation (10) and Equation (11) are equivalent in our operating range since a quadratic $y = pu^2 + qu + r$ can be approximated by a quadratic for a small region as $u = ly^2 + my + n$.

2.2 In Matrix Form

The system $\mathbf{T} = \mathbf{V}\eta$ is expressed as:

$$\begin{pmatrix} T1 \\ T2 \\ \vdots \\ T_n \end{pmatrix} = \begin{pmatrix} 1 & V_1 & V_1^2 \\ 1 & V_2 & V_2^2 \\ \vdots & \vdots & \vdots \\ 1 & V_n & V_n^2 \end{pmatrix} \begin{pmatrix} \eta_0 \\ \eta_1 \\ \eta_2 \end{pmatrix} \tag{12}$$

Goal: To find best-fit η for the equation. Since $\mathbf{V}$ is a tall matrix for our data, this system of equation is overdetermined and usually has no exact solution.

We try to find η that minimizes $\|\mathbf{V}\eta - \mathbf{T}\|$, i.e., the vector in Col($\mathbf{V}$) closest to $\mathbf{T}$ in Euclidean distance. Using the normal equations for least squares, we estimate $\hat{\eta}$:

$$\eta = \left(\mathbf{V}^\top \mathbf{V}\right)^{-1} \mathbf{V}^\top \mathbf{T} \tag{13}$$

After running the obtained data through the python code, the trained model is obtained giving best-fit η as:

$$\eta = \begin{pmatrix} -367.48044857 \\ 37.67235012 \\ 54.77146835 \end{pmatrix} \tag{14}$$

This least square solution is obtained from the formula on Python Code Line 30. When the least square solution was found using `np.linalg.lstsq` (line 33) and using SVD (lines 37-38), the $\hat{\eta}$ vector obtained was equal:

$$\eta = \begin{pmatrix} -367.48044858 \\ 37.67235009 \\ 54.77146835 \end{pmatrix} \tag{15}$$

Even though the `linalg` module of numpy has a `lstsq` function, internally it computes the solution using Singular Value Decomposition (SVD) for improved numerical stability and accuracy.

3 Training Data

Table 1: Training Data

Temperature ($^{\circ}$C)	Voltage Reading (V)
27.5	2.3558
33.8	2.3851
37.9	2.3998
41.5	2.4145
47.9	2.4389
50.6	2.4487
52.6	2.4536
57.6	2.4633
62.2	2.4780
63.8	2.4780
66.1	2.4878
68.5	2.4927
73.3	2.5122
74.9	2.5171
77.7	2.5269
81.1	2.5367
84.3	2.5513
89.5	2.5660
91.7	2.5709
96.7	2.5904
44.8	2.4218
59.3	2.4682
64.9	2.4829
70.9	2.5024
87.4	2.5611

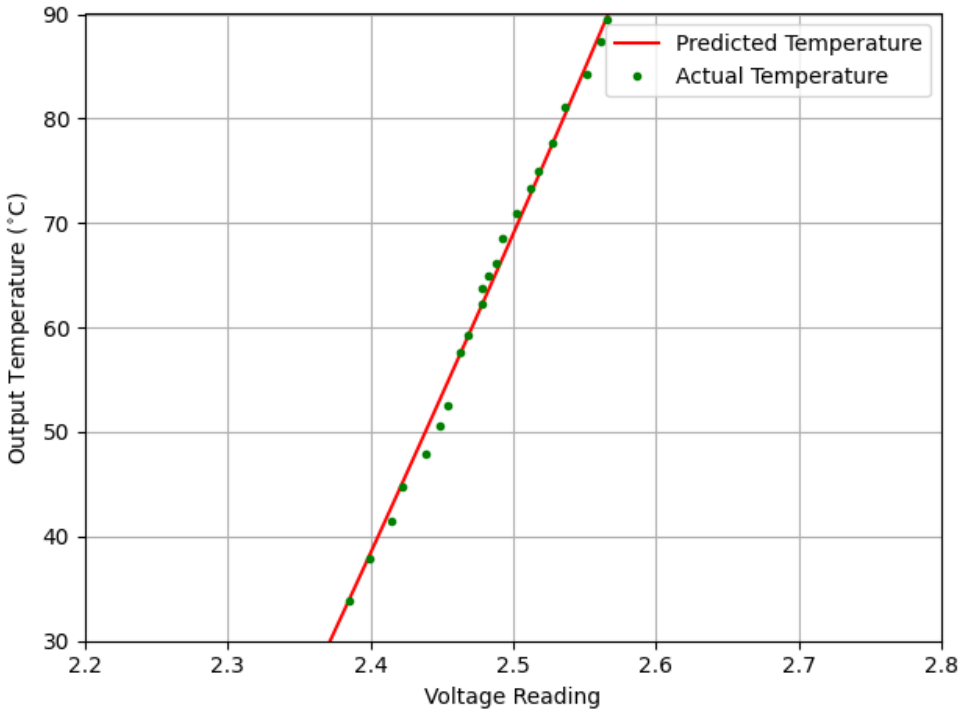

Figure 2: Training Data Plot

4 Validation Data

Table 2: Validation Data

Predicted Temp (°C)	True Temp (°C)	Voltage Reading (V)
36.89	36.1	2.3949
42.78	42.5	2.4145
51.68	50.3	2.4438
59.18	57.1	2.4682
63.71	61.9	2.4829
71.31	69.5	2.5073

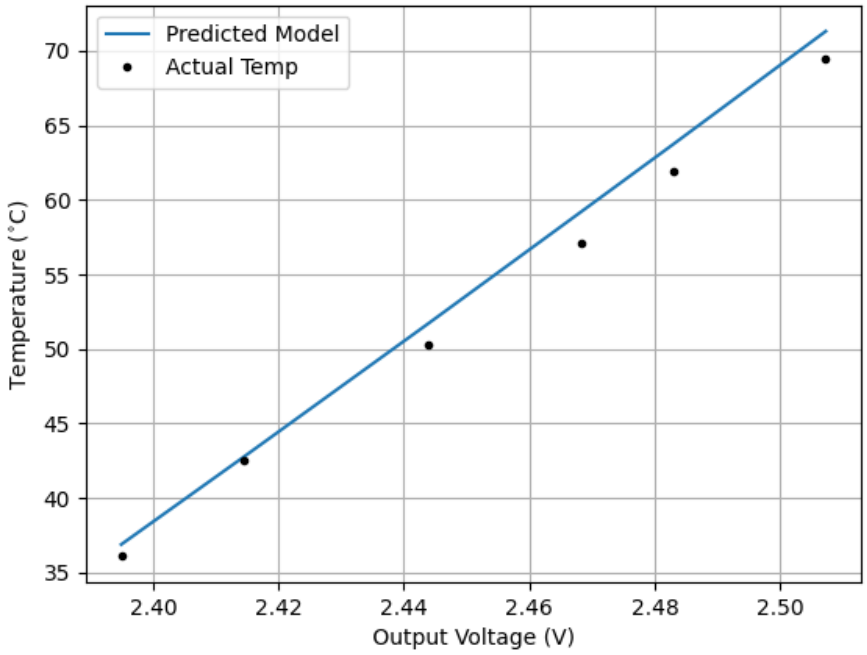

Figure 3: Validation Data Plot

5 PYTHON CODE

```python
import numpy as np
import numpy.linalg as LA
import sys
from pathlib import Path
import matplotlib.pyplot as plt

# --- Load Training Data ---
data = {}
# Assumes 'training_data.txt' is in the same folder
with open(Path(__file__).resolve().parent / "training_data.txt") as fp:
    for line in fp:
        # Assuming format: "Temperature Voltage"
        parts = line.split()
        if len(parts) >= 2:
            data[parts[0]] = parts[1]

try:
    n = int(sys.argv[1]) # Polynomial degree
except IndexError:
    n = 2 # Default to quadratic

T = np.empty((len(data), 1))
V = np.empty((len(data), n + 1))
i = 0
for temp, volt in data.items():
    T[i] = float(temp)
    volts = []
    volt_val = float(volt)
    for j in range(n + 1):
        if j == 0:
            volts.append(1) # Column for eta_0
        else:
            volts.append(volts[-1] * volt_val)
    V[i] = volts
    i += 1

# --- Least Squares Solutions ---
# 1. Using Normal Equation
lst_squares = LA.inv(V.T @ V) @ V.T @ T
print("--- Solution from Normal Equation ---")
print(lst_squares)

# 2. Using numpy.linalg.lstsq (preferred)
lstsq_sol = np.linalg.lstsq(V, T, rcond=None)[0]
print("\n--- Solution from np.linalg.lstsq ---")
print(lstsq_sol)

# 3. Using SVD
U, S, VT = np.linalg.svd(V, full_matrices=False)
n_svd = VT.T @ np.linalg.inv(np.diag(S)) @ U.T @ T
print("\n--- Solution from SVD ---")
print(n_svd)

# --- Plot Code (Training) ---
plt.figure(1)
plt.plot(V[:, [1]], V @ lst_squares, color="red", label="Predicted Temperature")
plt.plot(V[:, [1]], T, 'k.', label="Actual Temperature")
plt.grid()
plt.xlim(2.2, 2.8)
plt.ylim(30, 90)
plt.ylabel('Output Temperature ($^{\degree}$C)')
plt.xlabel('Voltage Reading (V)')
plt.tight_layout()
plt.legend(loc='best')
plt.savefig(Path(__file__).resolve().parent / "train.png")
```

```
plt.close('all')

# --- Validation ---
valid_data = {}
# Assumes 'validation_data.txt' is in the same folder
# Format: "Predicted True Voltage"
with open(Path(__file__).resolve().parent / "validation_data.txt") as fp:
    for line in fp:
        parts = line.split()
        if len(parts) >= 3:
            # Key: True Temp, Value: [Voltage, Predicted Temp]
            valid_data[parts[1]] = [parts[2], parts[0]]

T_thermo = np.empty((len(valid_data), 1)) # True Temp
Vv = np.empty((len(valid_data), n + 1))
T_pt = np.empty((len(valid_data), 1))     # Predicted Temp (from model)
i = 0
for temp_true, volt_data in valid_data.items():
    T_thermo[i] = float(temp_true)
    T_pt[i] = float(volt_data[1]) # Predicted Temp from file
    volts = []
    volt_val = float(volt_data[0]) # Voltage from file
    for j in range(n + 1):
        if j == 0:
            volts.append(1)
        else:
            volts.append(volts[-1] * volt_val)
    Vv[i] = volts
    i += 1

# --- Plot Code (Validation) ---
plt.figure(2)
# Note: T_pt is the *predicted* temp from the file.
# Vv @ lst_squares would be a *new* prediction.
# The plot from the doc plots T_pt vs V, and T_thermo vs V.
plt.plot(Vv[:, [1]], T_pt, 'r-', label="Predicted Model")
plt.plot(Vv[:, [1]], T_thermo, 'k.', label="Actual Temp")
plt.xlabel('Output Voltage (V)')
plt.ylabel('Temperature ($^{\degree}$C)')
plt.grid()
plt.legend(loc='best')
plt.savefig(Path(__file__).resolve().parent / "valid.png")
```

Listing 1: Python Code for Model Training and Validation

6 ARDUINO PROGRAM

```
#include <LiquidCrystal.h>

// Initialize the library with the numbers of the interface pins
// LiquidCrystal lcd(RS, E, D4, D5, D6, D7);
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

int status = 1; // Controls the display mode
int buttonPin = 7; // Push button on digital pin 7

// Custom char for degree symbol
byte degreeSymbol[8] = {
    0b00110,
    0b01001,
    0b01001,
    0b00110,
    0b00000,
    0b00000,
    0b00000,
    0b00000
};

void setup() {
    lcd.begin(16, 2);
    lcd.clear();
    Serial.begin(9600);

    // Using digital pin 13 as a stable 5V source (as per doc)
    pinMode(13, OUTPUT);
    digitalWrite(13, HIGH);

    // Button pin setup
    pinMode(buttonPin, INPUT_PULLUP); // Use internal pull-up resistor

    // Create the custom degree symbol
    lcd.createChar(0, degreeSymbol);
}

void loop() {
    float volt, temp, tempf;

    // These are the coefficients (eta_0, eta_1, eta_2) from Python
    float a0 = -367.48044857;
    float a1 = 37.67235012;
    float a2 = 54.77146835;

    // Read analog pin A1 (as per doc)
    volt = (5.0 * analogRead(A1) / 1023.0);

    // Check for button press
    if (digitalRead(buttonPin) == LOW) { // Button is pressed
        status++;
        if (status > 2) {
            status = 0; // Cycle through modes 0, 1, 2
        }
        delay(250); // Simple debounce
    }

    // Calculate temperature from voltage using the model
    temp = a0 + (a1 * volt) + (a2 * volt * volt);

    // Convert to Fahrenheit
    tempf = temp * 1.8 + 32.0;

    // Clear the LCD for new data
    lcd.clear();
```

```
66
67      switch (status) {
68          case 1: // Display Temperature in Celsius
69              lcd.setCursor(0, 0);
70              lcd.print("Temperature ");
71              lcd.write((byte)0); // Print degree symbol
72              lcd.print("C");
73              lcd.setCursor(0, 1);
74              lcd.print(temp);
75              Serial.println(temp);
76              break;
77
78          case 2: // Display Temperature in Fahrenheit
79              lcd.setCursor(0, 0);
80              lcd.print("Temperature ");
81              lcd.write((byte)0); // Print degree symbol
82              lcd.print("F");
83              lcd.setCursor(0, 1);
84              lcd.print(tempf);
85              Serial.println(tempf);
86              break;
87
88          default: // case 0: Display Voltage Reading
89              lcd.setCursor(0, 0);
90              lcd.print("Voltage Reading");
91              lcd.setCursor(0, 1);
92              lcd.print(volt, 4); // Print voltage with 4 decimal places
93              Serial.println(volt, 4);
94              break;
95      }
96
97      delay(200); // Refresh rate
98  }
```

Listing 2: Arduino Sketch (main.cpp)

ARDUINO FUNCTIONS

1. Initialize LCD and analog input pin.
2. Read analog voltage: $V = \text{analogRead(pin)} \times 5.0/1023.0$
3. Compute temperature (In Celsius) using: $T_C = \eta_0 + \eta_1 V + \eta_2 V^2$
4. Compute temperature (In Fahrenheit) using: $T_F = T_C \times 1.8 + 32.0$
5. LCD can switch modes and display temperature both in Celsius and in Fahrenheit.
6. Separate debugging mode showing current voltage reading across the sensor.
7. Stateful push button logic to remember and switch between modes of operation.

Learning Outcomes

1. Understanding PT-100 sensor characteristics and errors in sensor instrumentation.
2. Designing and implementing a complete embedded system which interfaces an analog sensor with an Arduino microcontroller.
3. Implementing least squares regression for different complexities and modeling approaches to solve for the best-fit polynomial model.
4. Applying the industry-standard Callendar-Van Dusen equation to demonstrate fitting a physical model to a curve.
5. Applied error minimisation for common issues:
    - Separated sensor and LCD wiring to reduce confusion.
    - Found high amounts of noise in Arduino 5V-pin, after testing decided to use the much more stable output of a digital pin as sensor power source.
6. Implemented advanced functionality like user-controlled mode switching.

CONCLUSION

This project successfully implemented a temperature sensor using an Android, Arduino microcontroller, PT-100 sensor, LCD and the least squares approach.