1 Introduction

From ancient data tracking to modern databases, data collection and analysis have been crucial in advancing our societies and human knowledge throughout history. Environmental data, especially, has become crucial across many fields and applications. This includes climate monitoring, industrial automation, smart agriculture, and home automation. For example, real-time temperature and humidity data, play a big role in weather forecasting, HVAC system optimization, and even in Microbiology where stable conditions in incubators or laboratories must be maintained. Data acquisition, processing, and analysis can improve decision-making, and help optimize and automate systems. MicroPython provides a method to program microcontrollers easily for such a task, and can therefore be used for real-time data collection. Python also offers powerful data analysis libraries, allowing necessary analysis and processing of collected data depending on the application.

This project focuses on the collection and analysis of environmental data (temperature and humidity) using the MicroPython-enabled development board, STM32 Nucelo F401RE. The goal of this project is to design a system that reads temperature and humidity values from a DHT11 temperature and humidity sensor, and transmits them to a PC for further processing. The data must then be preprocessed, analyzed, and visualized using Python libraries like Pandas, NumPy, and Matplotlib. This project uses MicroPython for data collection and Python for analysis, demonstrating how embedded systems and data science collaborate in practical applications.

2 Hardware Setup

The main hardware component used for this project was a MicroPython-enabled STM32 Nucleo development board, the Nucelo F401RE from STMicroelectronics. As well as, to measure and collect the temperature and humidity readings, the combined Temperature and Humidity Sensor DHT11 was used.

To set up the hardware for the project, these components are connected as shown in Figures 2 and 3. The Nucelo board is powered via a USB connected from a PC, which also serves as the communication interface for logging the sensor readings, debugging, and analysis. The DHT11 sensor has 3 pins to be connected as shown in Figure 1:

- VCC (Power): this pin is connected to the 5V pin on the nucleo board.
- DATA: the data pin on the sensor is connected to a GPIO pin (PA1) on the board.
- GND (Ground): connected to the GND pin of the board.

Figure 1: DHT11 Sensor [1]

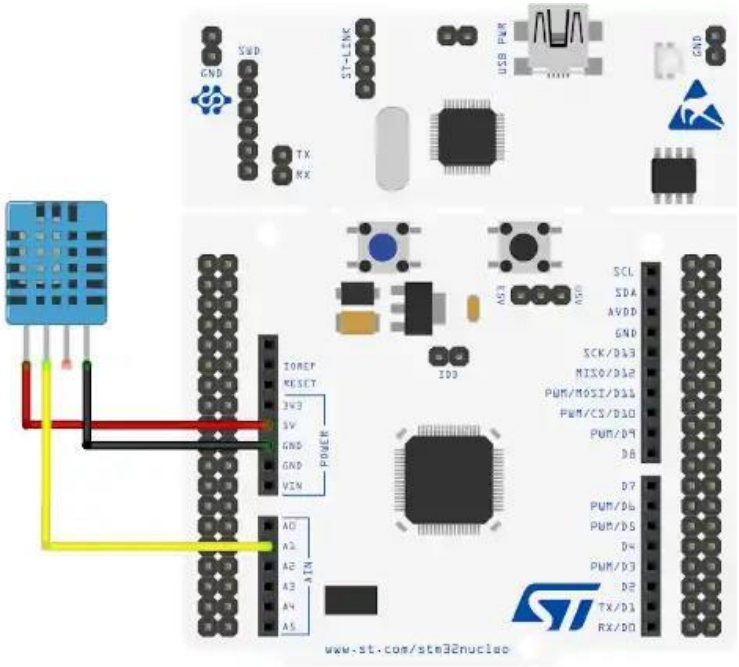


Figure 2: Circuit Connections [2]

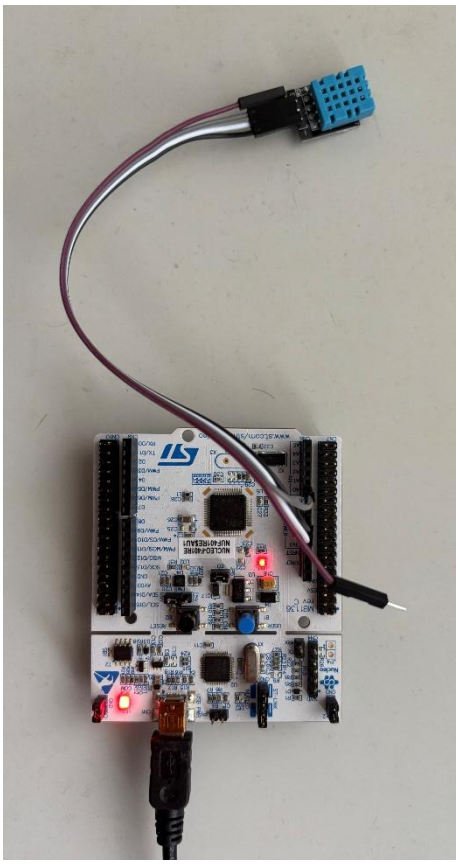


Figure 3: Circuit Connections

3 Data Collection

The STM32 Nucelo board is programmed by flashing the **main.py** source code onto it. This programs the board to read temperature and humidity values from the connected DHT11 sensor every 10 seconds. The code also displays (prints) the readings on the board's console, and sends the data to the PC via serial connection (UART) for further processing. Those values were displayed during this step on the PC, using Thonny. This is achieved in the code by the following steps:

- `sensor = dht.DHT11(machine.Pin('PA1', machine.Pin.IN))`
  An instance of the DHT11 sensor is created. The `dht.DHT11()` method is used to interface with the DHT11 sensor, which is connected to the pin 'PA1' on the board. The sensor is initialized in input mode "`IN`" to ensure that the nucleo board can receive the readings from the sensor.

- `uart = machine.UART(2, baudrate=115200)`
  UART on port 2 is configured with a baud rate of 115200, allowing the board to send the temperature and humidity data over a serial connection (USB-UART interface), to the PC using the `uart.write()` function.

- Using a `while True:` loop, the code keeps running, ensuring continuous data readings from the sensor every 10 seconds. This 10 second interval is introduced using a time delay `time.sleep(10)`

- `sensor.measure()` triggers the sensor to take a measurements, `sensor.temperature()` and `sensor.humidity()` retrieve the temperature (in Celsius) and humidity (in percentage), respectively.

4 Data Transfer

At the this point the STM32 Nucelo board is collecting the temperature and humidity data, but its only printing it to the PC via UART, and can't be used for analysis just yet. Therefore, the next step is to be able to transfer this recorded data and create a dataset for storage, and analysis. This is done in this project by running the **serial_read_to_csv.py** source code on the PC. This code continuously listens to data coming from the STM32 board's UART serial port and extracts the incoming temperature and humidity data. The program then saves this data into a CSV file on the PC, to be used later for analysis and processing. This is achieved in the code by the following:

- `ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)`
  First, the serial connection is set up using the `serial.Serial()` function. This is used to configure the serial port to communicate with the Nucleo board. `SERIAL_PORT` and `BAUD_RATE` variables define the configuration settings for the serial communication between the PC and the STM32 Nucleo board. Here `SERIAL_PORT` is defined as `COM6`, which is the serial port that the board is connected to in this case. The baud rate here is set to 115200, to match the baud rate set on the Nucleo board in the main.py source code. These parameters are important to ensure successful serial communication.

- ```
  with open("sensor_data.csv", "w", newline="") as csvfile:
      csv_writer = csv.writer(csvfile)
      csv_writer.writerow(["timestamp", "temperature",
  "humidity"])
  ```
  The code sets up the CSV file where the received data will be stored using `open() as csvfile`. The file `sensor_data.csv` is opened in write mode "`w`" and a CSV writer is created. Here a header row is also written into the CSV file with the column names: "timestamp", "temperature", and "humidity". This ensures the data is well documented and stored for further analysis.

- In an infinite while loop it continuously listens for data from the serial port. `ser.readline().decode("utf-8").strip()` reads a line from the serial port
  ```

and the data is decoded from bytes to a string and cleaned of extra spaces/newlines.

- The data is split using `line.split(",")`, extracting temperature and humidity values. This ensures the data is structured correctly before storage.

Using **main.py** running on the STM32 Nucelo board, and running **serial_read_to_csv.py** on the PC, temperature and humidity data were recorded **every 10 seconds** for **42 hours**. This data was saved in the **sensor_dara.csv** file, which created a dataset that be used for processing and further analysis.

5 Preprocessing

The preprocessing of the dataset is carried out using the Pandas library and its methods and functions. First, the CSV file is read and its contents are loaded into a Pandas DataFrame df using `df= pd.read_csv(file_path)`. Here the path of the CSV file is stored in `file_path`. Before preprocessing the data frame looks like this:

	timestamp	temperature	humidity
0	795666791	18	10
1	795666801	18	10
2	795666811	18	10
3	795666821	18	10
4	795666831	18	10
...	...	...	...
14698	795811726	20	34
14699	795811736	20	34
14700	795811746	20	34
14701	795811756	20	34
14702	795811766	20	34

14703 rows × 3 columns

So after 42 hours of 10 second interval readings, the dataset includes **14703 temperature and humidity readings**.

- **Adjusting Time stamps:**
  The first preprocessing step was to adjust the time stamps using **Pandas** to convert the time stamps into proper date and time values. This is a very important, to know at what time and data each temperature was recorded. This is crucial for the analysis of environmental data because temperatures and humidities are different throughout the day. The code adds a column of the new adjusted time stamps, which shows the date and time that each reading was recorded at. The following data frame is printed after adjusting time stamps:

	timestamp	temperature	humidity	datetime
0	795666791	18	10	2025-03-18 22:37:32+00:00
1	795666801	18	10	2025-03-18 22:37:42+00:00
2	795666811	18	10	2025-03-18 22:37:52+00:00
3	795666821	18	10	2025-03-18 22:38:02+00:00
4	795666831	18	10	2025-03-18 22:38:12+00:00
...	...	...	...	...
14698	795811726	20	34	2025-03-20 14:53:07+00:00
14699	795811736	20	34	2025-03-20 14:53:17+00:00
14700	795811746	20	34	2025-03-20 14:53:27+00:00
14701	795811756	20	34	2025-03-20 14:53:37+00:00
14702	795811766	20	34	2025-03-20 14:53:47+00:00

[14703 rows x 4 columns]

- **Removing duplicates:**
  The duplicate temperature and humidity readings from the dataset were removed, while keeping only the first occurrence. This was done using the `drop_duplicates()` which removes duplicate rows based on the specified columns where were “temperature” and “humidity” here. If two or more rows have the same values for both, it keeps only the first occurrence using `keep="first"`. This step is important to avoid an overly long dataset and simplify the data. This is especially important here since the data is being recorded every 10 seconds and environmental parameters don’t change quickly, therefore the dataset contains many duplicates. Keeping duplicates here would be pointless, by removing them one could directly see the next different temperature and humidity values and look at the time stamp to know when the value change happened. This can be seen in the data frame printed:

	timestamp	temperature	humidity	datetime
0	795666791	18	10	2025-03-18 22:37:32+00:00
22	795667009	17	10	2025-03-18 22:41:10+00:00
99	795667770	19	10	2025-03-18 22:53:51+00:00
206	795668826	20	10	2025-03-18 23:11:27+00:00
571	795672429	21	10	2025-03-19 00:11:30+00:00
...	...	...	...	...
14340	795808191	20	27	2025-03-20 13:54:12+00:00
14474	795809510	20	31	2025-03-20 14:16:11+00:00
14524	795810003	20	32	2025-03-20 14:24:24+00:00
14541	795810170	20	33	2025-03-20 14:27:11+00:00
14594	795810692	20	34	2025-03-20 14:35:53+00:00

[106 rows x 4 columns]

- **Calculating basic statistics:**
  Basic statistics of the dataset such as min, max, mean, and standard deviations are calculated using `df.describe()` from Pandas. The following basic statistics calculations were found:

	timestamp	temperature	humidity
count	1.470300e+04	14703.000000	14703.000000
mean	7.957393e+08	21.103108	19.913351
std	4.186785e+04	3.689851	10.241804
min	7.956668e+08	15.000000	10.000000
25%	7.957030e+08	18.000000	10.000000
50%	7.957392e+08	21.000000	10.000000
75%	7.957756e+08	23.000000	30.000000
max	7.958118e+08	30.000000	42.000000

From these calculations it is shown that during the 42 hour period where the data was being recorded:

least temperature recorded : 15 °C

highest temperature recorded: 30 °C

mean temperature: 21 °C

least humidity recorded : 10 %

highest humidity recorded: 42 %

mean humidity: 20 %

6 Further Analysis

To capture the temperature variations and trends over time, multiple regression models were employed for analysis. The Linear Regression Model provides a baseline estimation of the temperature and humidity over time, providing a simple linear relationship between time and temperature. Polynomial Regression using degree 5, captures non-linear temperature variations with a higher accuracy. After generating polynomial predictions, smooth polynomial predictions were generated to create a more visually interpretable and accurate representation of the temperature and humidity trends over time

```
X_smooth = np.linspace(X.min(), X.max(), 500).reshape(-1, 1)

y_poly_pred = poly_model.predict(X_smooth)
```

The Least Squares Solution using the Normal Equation and Singular Value Decomposition (SVD) are both methods to find the best-fit line for the data. The Normal Equation directly calculates the best parameters for the regression by minimizing the error, which works well for smaller datasets. On the other hand, Singular Value Decomposition (SVD) breaks down the data into smaller parts and is more stable, especially when the data is complex or large. SVD avoids the need to directly invert large matrices, making it a better option for more complex problems. Both methods help in finding the best parameters for the regression, with the normal equation being simpler for smaller data and SVD being better for larger or more difficult datasets.

Overall, the polynomial regression proved to be the most effective in modeling the temperature trends, while the linear model, though simpler, still provided valuable insights.

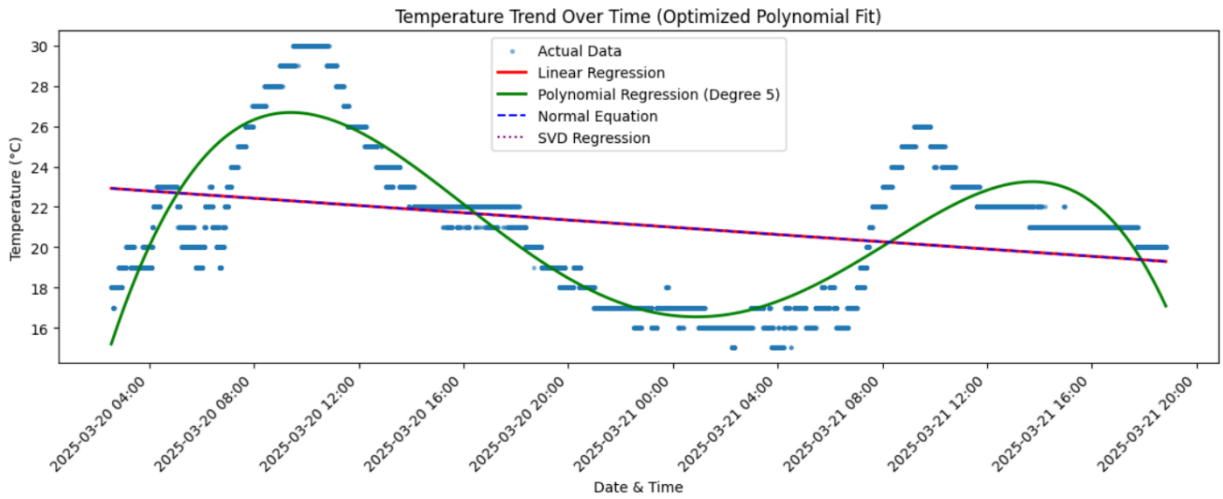
Optimal Heating and Cooling Threshold Calculation

Here Newtons's method is used to calculate the optimal threshold value for heating/cooling that minimizes the deviation between the threshold and the temperature data. There are 4 input parameters used by this function: `temps` (temperatures), `initial_treshold`, `max_iter` (maximum iteration), and `tol` (tolerance). The method iteratively adjusts the threshold based on the calculated gradient and Hessian until the deviation is minimized within a specified tolerance. The result is the threshold that best fits the temperature data. Once the threshold is determined, the `optimal_threshold()` function evaluates its effectiveness in minimizing the "cost" associated with heating and cooling adjustments, using `heating_threshold` and `cooling_threshold`. It computes the squared deviations when temperatures fall outside desired ranges, ensuring the best balance between comfort and efficiency.

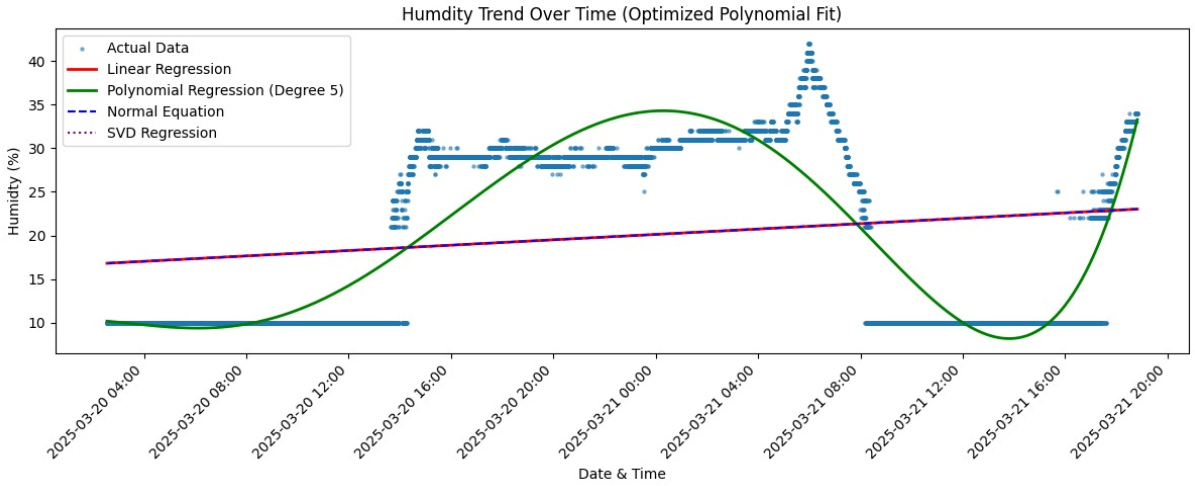
7 Visualization

To illustrate the analyzed trends, plots were generated using Matplotlib showcasing:

- **Temperature Trends Over Time:**
  - Scatter plots of actual temperature values.
  - Regression model predictions overlaid for comparison.



- **Humidity Trends Over Time:**
    - Actual humidity values with regression predictions.
    - Normal equation and SVD-based estimations.



Visual representations created by Matplotlib provide a clear understanding of seasonal variations and can assist in predictive modeling for climate control applications.