Coursera: Embedded Systems interface QT-Python

Natalia Baeza

Agust 2025

1 Introduction

The purpose of this document is to explain in detail the performance of a prototype temperature and humidity interface for an embedded system, based on the instructions from a Coursera module assignment. The name of the course is *Rapid Prototyping of Embedded Interface Designs*"

For this project, the module suggested using fake data to simulate sensor readings. However, I decided to use real sensors, which means some objectives have slightly changed due to component limitations. Additionally, prerequisites such as installation steps will be skipped in this document, but you can find all the details in the code.

Please note: this project is just for fun (and for the Coursera assignment), so don't be too critical. Could things be improved? Yes, of course—but it is what it is.

Now that the disclaimers are out of the way—and since the code can be found on my personal GitHub repo repository—let's enjoy the process!

2 Objectives

2.1 Coursera Requirements

- Your main Python program should start the Qt UI and be able to respond to events from it and send data to it.
- Assume the data from the pseudo-sensor is humidity and temperature.
- Provide a Qt button that allows your main Python program to read a single humidity/temperature value and display it on the UI.
- Use a simple SQL database to store incoming humidity/temperature pairs along with a timestamp.
- Provide a Qt button that will read 10 values from the pseudo-sensor with a one-second delay between each reading.
- Provide a Qt button that will calculate and display on the UI all temperature/humidity pairs read so far—including minimum, maximum, and average values for each type of reading.
- Your UI should include two fields to set temperature and humidity alarm values. If any temperature or humidity reading from the pseudo-sensor exceeds these alarm values, indicate an alarm on the UI. These alarm fields should have default values at startup.
- Include a Qt button on your UI that will close the UI and terminate the main Python program.
- Optional:
  - Add a button on the UI that toggles all displayed temperature values between Fahrenheit and Celsius.
  - For additional practice, when executing step 6 above, generate line graphs of temperature and humidity and display them on the UI.

2.2 Requirements Adjustment

Some modifications were made to the original objectives due to the real sensor's behavior. For example, the speed of data collection from the sensors is longer than one second, so a one-second delay is not possible. Besides, in real-life conditions, the temperature does not change quickly, so collecting 10 samples every second would not show significant variation. The microcontroller used in this project has been programmed to send data every 10 seconds.

The button for single data acquisition was replaced by automatic data display. However, there are still two other buttons on the UI.

The number of samples can be configured in the code (in a future version, this might be changed to a textbox or a similar feature).

Alarm thresholds currently work only when some data is plotted (spoiler). This may change in future versions.

3 Components

Let's illustrate the system and describe it!

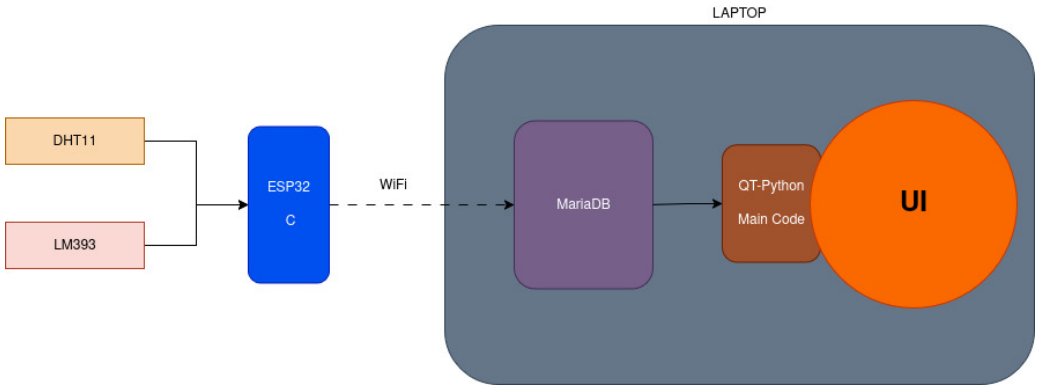


Figure 1: General system diagram

Everything related to the microcontroller will be omitted to save space in this document. Nevertheless, it is important to note that all messages are sent through Wi-Fi to **MariaDB** running on the laptop. Each message contains temperature (in Celsius), humidity percentage (both from DHT11), and light intensity from LM393.

3.1 Database: MariaDB

On the laptop, there are two main components: MariaDB and the Qt-Python code. The database was chosen for its simplicity and good integration with Python.

After installing MariaDB on the computer, a new database named **sensor_data** was created. Inside this database, a table called **readings** was defined with the following structure:

Field	Type	Extra
id	int(11)	
temperature	decimal(5,2)	
humidity	decimal(5,2)	
timestamp	datetime	MUL current_timestamp()
device_id	varchar(50)	
light_level	decimal(5,2)	
light_raw	int(11)	

Table 1: DESCRIBE readings - command

The timestamp value is set when the message arrives, taking the current machine clock. The microcontroller sends data to MariaDB, which stores it.

3.2 Qt-Python Code

For the implementation, a class named **DBManager** was created.

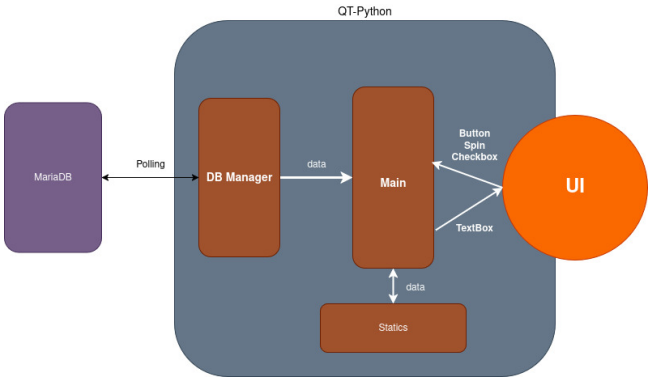


Figure 2: Part of the code

This class is responsible for connecting to the database and checking for new data using **QTimer** from Qt-Python every 8 seconds (configurable). The information is sent to the main code using **pyqtSignal** from QtCore. This class runs on a separate **QThread** to avoid blocking the main thread, which handles UI interactions. subsectionQt-Python: Main (UI) The main code initializes the UI and updates its components, as shown in Figure 3.

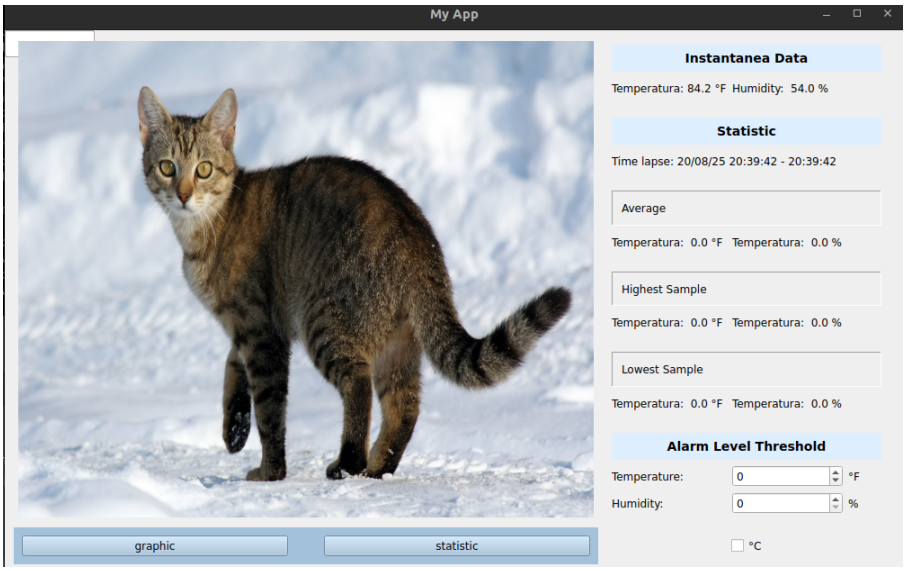


Figure 3: UI initialization

When the program starts, it sets the window size and displays a cat image while no graph is shown. Several **QTextBox** widgets display information such as instantaneous temperature and humidity, titles, and units. A **QCheckBox** allows changing the temperature unit, **QSpinBox** sets parameter thresholds, and **QPushButton** elements start data collection and plotting from the database.

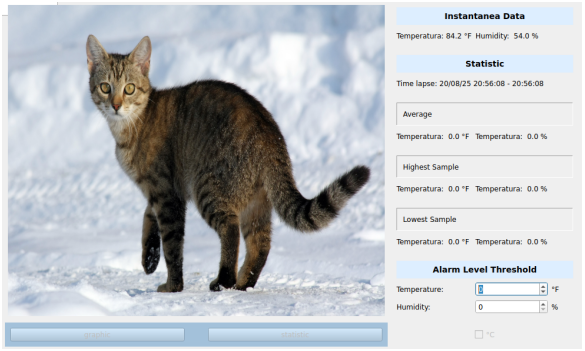
4 Functionality

Once the program is running, the user can see the instantaneous temperature and humidity in the upper-right corner without any interaction. Every time new data arrives in the database, the UI is updated to show the new values.

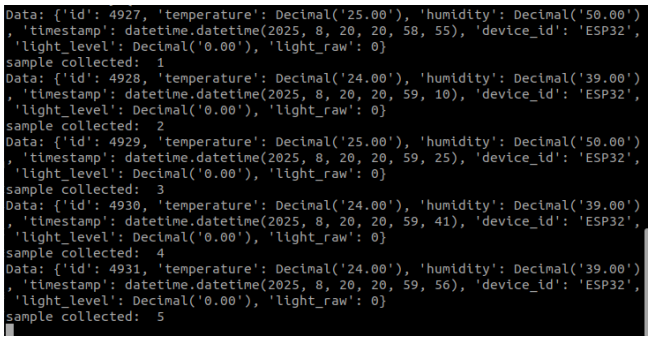
Figure 4: Current Data

4.1 Statistics

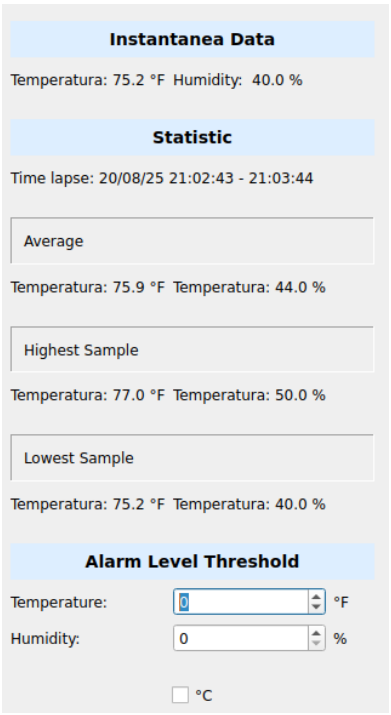
This section displays the time lapse between the first and last collected data samples, and calculates the average, maximum, and minimum values when the user presses the **Statistics** button.



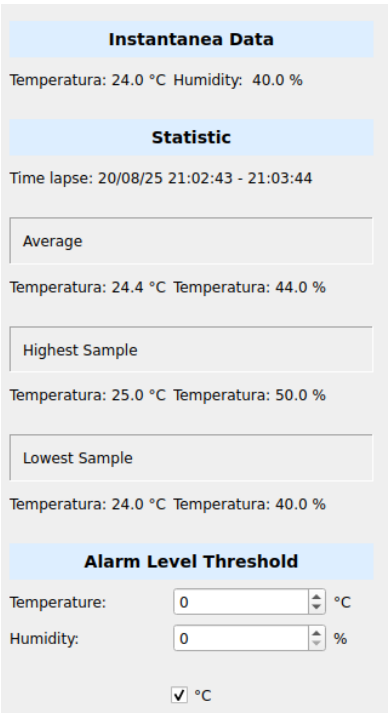
(a) UI after pressing a button



(b) Data retrieved from DB



(c) Statistics calculation displayed



(d) Unit change applied

Figure 5: Statistics functionality and unit change

When the button is pressed, both it and the other button are disabled until the process finishes. Once completed, all components return to their normal state.

Figure 5 also shows the **unit change** feature when the user checks the **checkbox** located at the bottom.

4.2 Plotting

When the user presses the **Statistics** button, its behavior is similar to the other button. This means that other components, such as the **QCheckBox** and the **QSpinBox** threshold selector, are disabled until the process is finished. During this time, the plot grid is displayed (Figure 6).

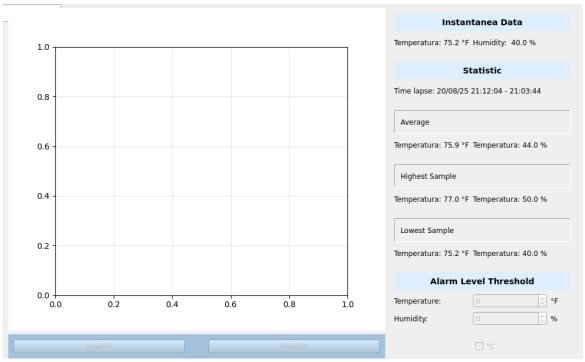
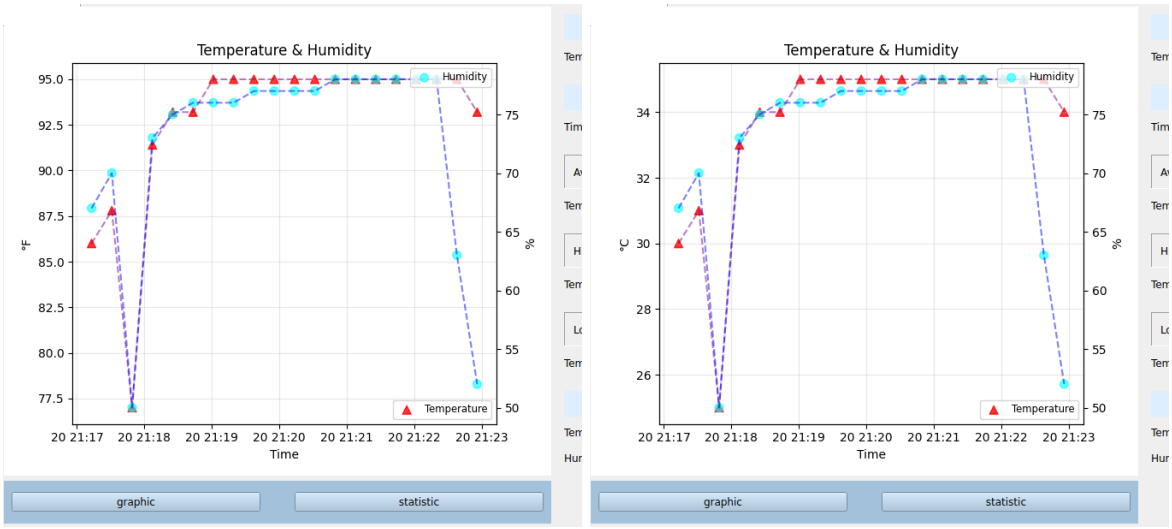


Figure 6: Plot grid displayed while waiting

For this feature, the number of samples was set to 20, and the temperature was intentionally modified by touching the sensor with different objects. The results can be seen in Figure 7, which also demonstrates the unit change when the **QCheckBox** is selected.



(a) UI after pressing the button

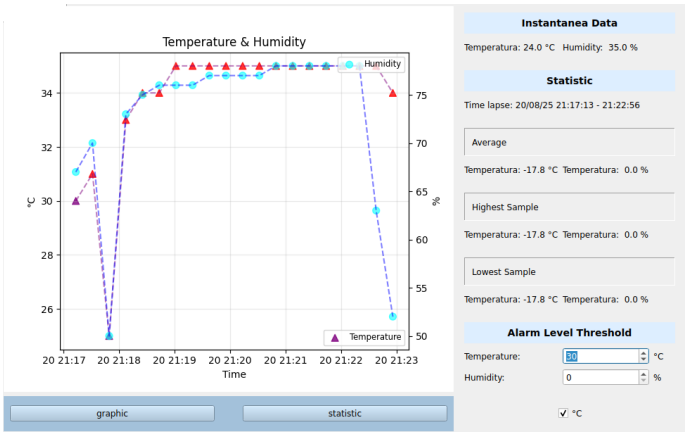
(b) Data retrieved from the database

Figure 7: Plotting with both units displayed

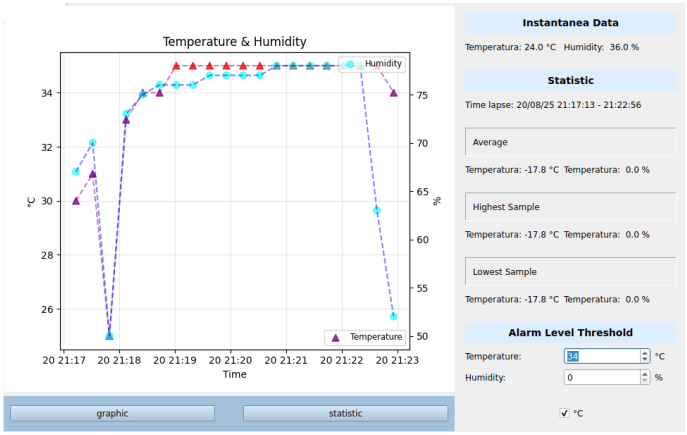
4.3 Threshold

The **QSpinBox** used for setting the threshold only works when a plot is displayed; otherwise, nothing happens.

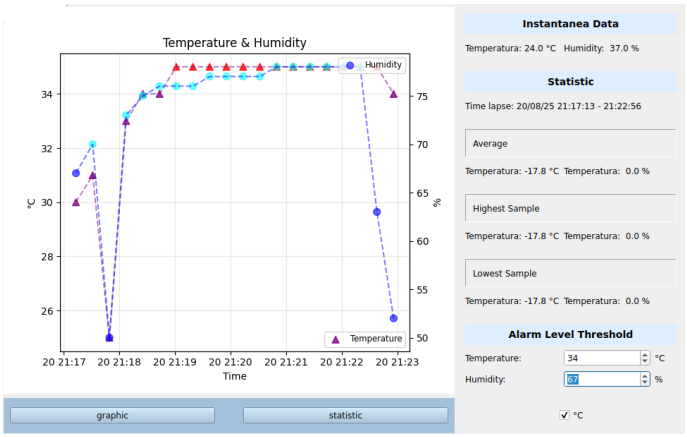
When the plot is generated, the **QSpinBox** allows the user to observe how the markers of each parameter change color depending on whether the data exceeds the configured threshold: temperature markers change from purple to red, and humidity markers from blue to cyan, as shown in Figure 8.



(a) Temperature threshold



(b) Temperature threshold equals 34 Celsius



(c) Humidity threshold applied

Figure 8: Threshold behaviour applied to plotted data

5 Future Adjustments

- Calculate statistics each time the graph is plotted.
- Add an **input box** to change the number of samples dynamically.
- Implement **real-time** plotting: each new data point will be plotted as it arrives.
- Provide separate plots for different parameters.
- Parameterize all configurations for better flexibility (code).