Assignment 3 - Smart Temperature Monitoring System

A.A 2024/2025

Elisa Yan

April 22, 2025

Contents

Chapter 1

Introduction

The third assignment for the embedded system requires the development of an intelligent temperature monitoring system called "**Smart Temperature Monitoring System**" for a closed environment with window control capabilities. The system consists of interconnected subsystems that interact with each other to monitor temperature, control window openings, and provide operator interfaces.

Chapter 2

System Architecture

The smart temperature monitoring system is architecturally organized into four interconnected subsystems: the temperature control, the window regulator, the control unit, and the dashboard. Each has a specific task within the integrated system.

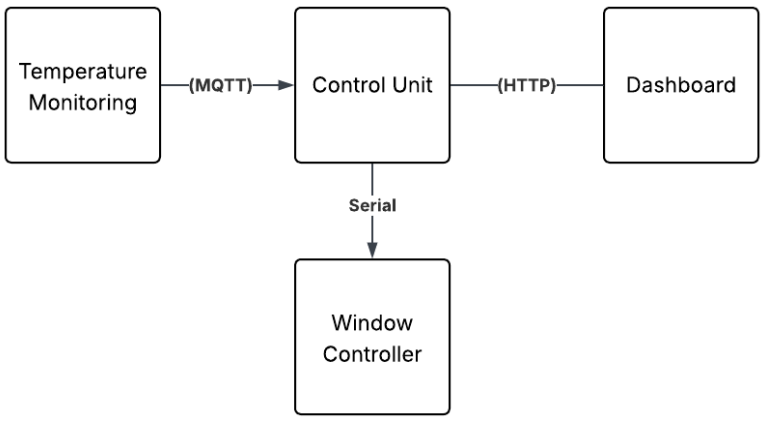

Figure 2.1: System architecture

There are various types of communication between the subsystems. Serial communication is used between the window controller and the control unit. MQTT is used to communicate the temperature monitoring system and the control unit. Finally, HTTP is used to connect the dashboard to the control unit.

The following sections discuss each subsystem in detail to better understand its functioning and interactions.

2.1 Temperature Monitoring subsystem

The Temperature Monitoring Subsystem performs real-time environmental sensing using an ESP32-based hardware platform equipped with a TMP36 temperature sensor. As illustrated in Figure 2.2, the FSM architecture comprises six distinct states, each managing a critical phase of the monitoring process:

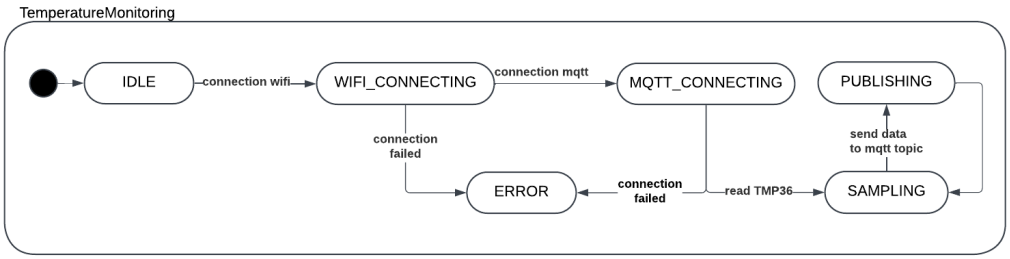

Figure 2.2: Temperature Monitoring FSM

- IDLE, the system enters an initial phase dedicated to preparing the hardware and initiating the WiFi connection process,
- WIFI_CONNECTING, this part is responsible for making sure the ESP32 successfully connects to the internet, and if it doesn't, it keeps trying until it does,
- MQTT_CONNECTING, establishes communication, allowing the ESP32 to send temperature data to another device or system that is listening to a specific topic. Only when this connection is confirmed will the system begin to collect and send data,
- SAMPLING, gets temperature data from the sensor at configurable intervals; after receiving a new temperature value the switches to publishing state,
- PUBLISHING, sends the data to the MQTT topic that was previously set up, if all goes well, it goes back to sampling and repeats the process,
- ERROR, if something goes wrong, for example, if the ESP32 can't connect to the WiFi or the MQTT broker, it will enter this state, give visual feedback via LED, and try to recover by reconnecting after some time.

2.2 Window Controller subsystem

The Window Controller Subsystem is designed to manage the percentage of window openings based on whether the system is in automatic or manual mode. This allows users to control the window locally and remotely, providing a more versatile and user-friendly experience.

At the core of the system is a central controller class, called "WindowControllerPlant", responsible for managing the current operating state. The system operates based on three main states:

- INIT
- AUTOMATIC_MODE
- MANUAL_MODE

In the INIT state, the system processes some setup routines, such as turning on the motor and displaying a welcome screen on the LCD. Then decides which mode to switch to based on the user.

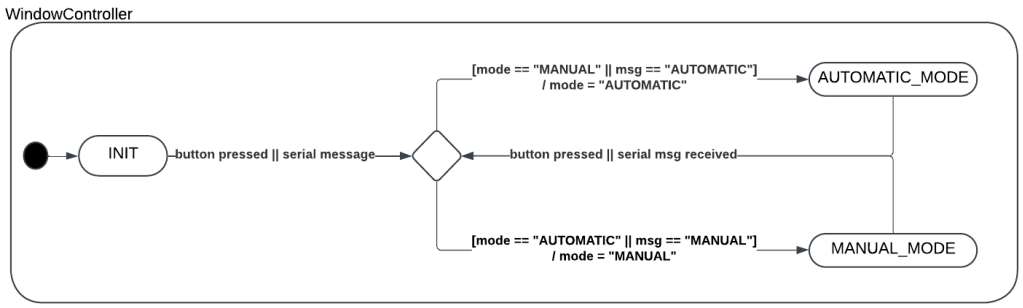

Figure 2.3: Window Controller FSM

Automatic and manual modes are controlled by separate, concurrently running tasks, which helps keep their functionalities isolated and well-structured.

2.2.1 AutomaticTask

In automatic mode the system listens for messages sent over the serial connection. The system then updates the LCD and the window position, respectively. If a request to change the mode is detected, either through a received message or by pressing the button, the system will exit automatic control and switch to manual mode.

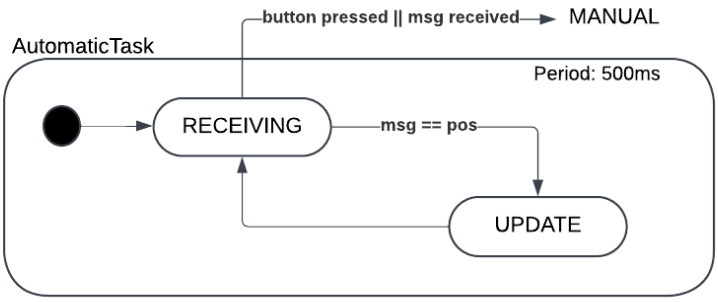

Figure 2.4: Automatic Task FSM

2.2.2 ManualTask

In manual mode, the behaviour is different. Here, the user controls the window's position directly using a potentiometer. The system reads the potentiometer's value, converts it into a window opening percentage, and adjusts the servomotor accordingly. Again, if there's any input suggesting a mode change, like a button click or a message from the dashboard, the system switches back to automatic.

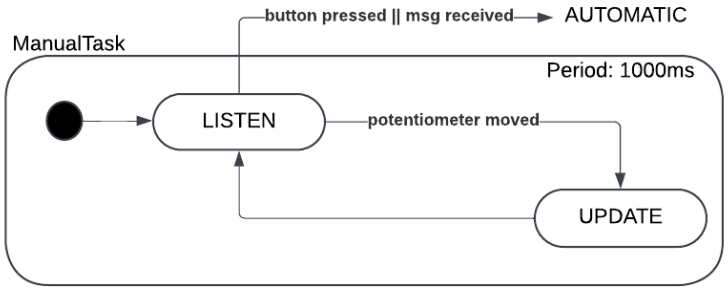

Figure 2.5: Manual Task FSM

2.3 Control Unit subsystem

The control unit subsystem serves as the backend for the other subsystems. Its main jobs are: handling temperature monitoring, making decisions based on those readings, and keeping everything in sync, from the physical window to the web dashboard.

If the system is connected to an MQTT broker and listens to a topic where temperature data is being published, then every time a new temperature value comes in, the system will check it and decide what state the environment is in.

There are four possible states: NORMAL, HOT, TOO_HOT, and ALARM. If the temperature exceeds a first threshold (T1), the system enters the HOT state. If it continues to rise and exceeds a second threshold (T2), it enters the TOO_HOT state. If it remains in this state for longer than a predefined time (DT), the system enters ALARM mode and waits for a manual reset from the dashboard.

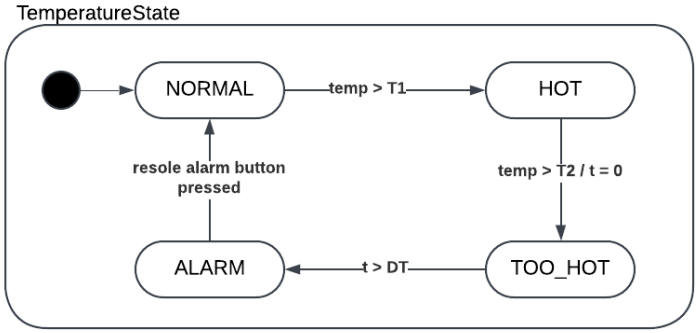

Figure 2.6: Temperature State

In AUTOMATIC mode, the system works out how much to open the window based on the temperature. For example, if it is warm but not too hot, the window ma y be half open. If it is too hot, the window is open all the way.

In MANUAL mode, however, the position of the window comes directly from the Arduino or the dashboard, depending on which was last updated.

If the user changes something on the Arduino, this subsystem will synchronize with the dashboard via HTTP; otherwise, if something is changed on the dashboard, this will update the Arduino via the serial line. This way, the system avoids conflicts and stays consistent.

2.4 Dashboard subsystem

The dashboard is the frontend interface to the system, allowing users to monitor and interact in real time. While the backend handles the logic and coordination between components, the role of the dashboard focuses on presentation and user interaction.

In particular, temperature data is visualized using a dynamic line graph powered by the *Chart.js* library. This chart is continuously updated, giving the user instant feedback on temperature trends over time. In addition to the graph, key temperature metrics such as the current average, minimum and maximum values are displayed, along with the system status and the current window position. In *Manual* mode, users control the window position via a slider, while *Automatic* mode handles it autonomously.

The dashboard also includes a *Resolve Alarm* button, which becomes active when the system enters an alarm state. When pressed, the button sends a request to the backend in an attempt to resolve the alarm condition. If successful, a temporary alert is shown to notify the user.

An overview of the dashboard interface is shown in Figure 2.7, which illustrates the main interactive elements available to the user.

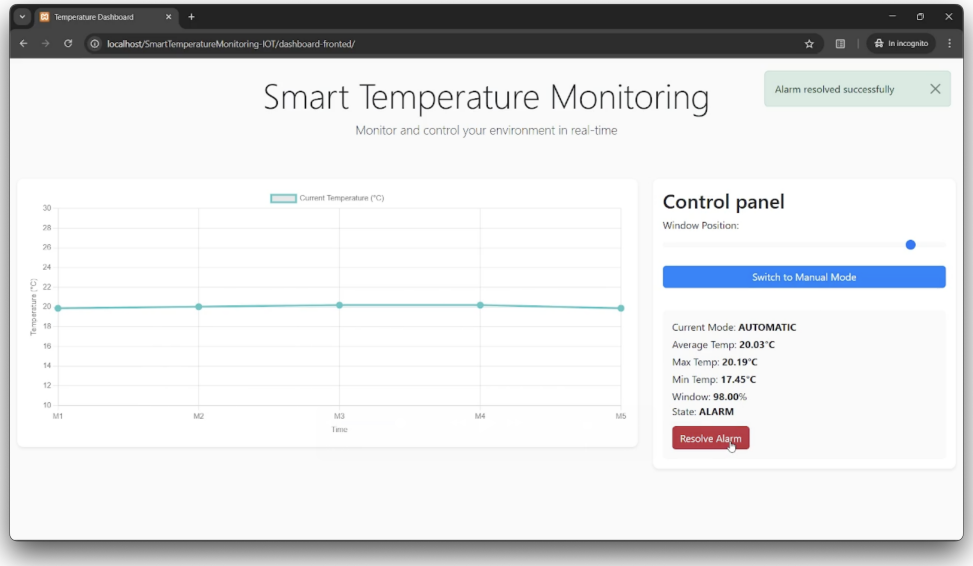

Figure 2.7: Dashboard interface

Chapter 3

Breadboard Scheme

3.1 Temperature Monitoring (ESP32)

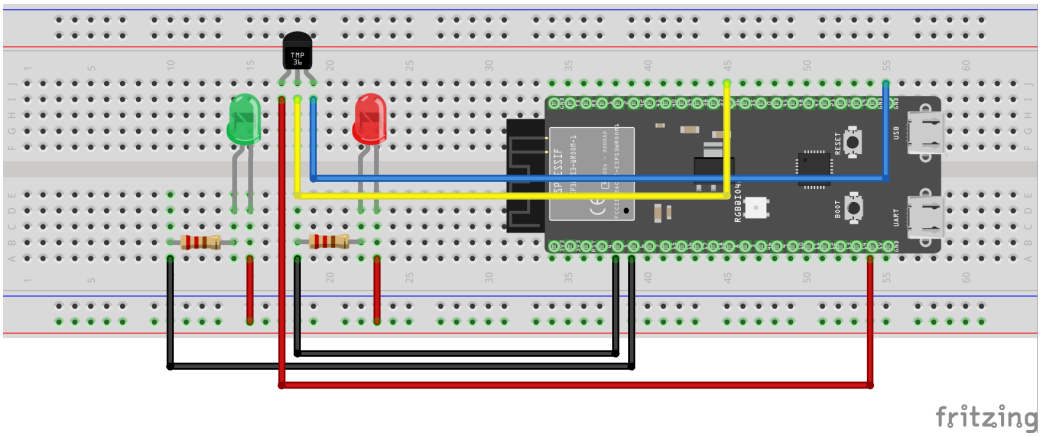

Figure 3.1: ESP breadboard scheme

3.2 Window Controller (Arduino)

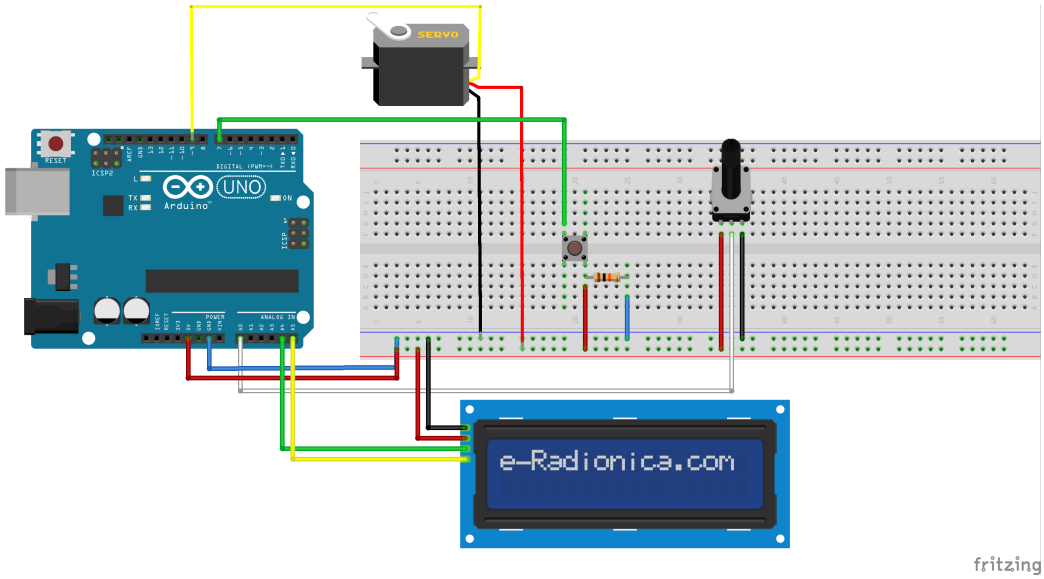

Figure 3.2: Arduino breadboard scheme

Chapter 4

System demo

Follow the link to view a video of the system: Smart Temperature Monitoring