Ministry of Higher Education

and Scientific Research

National Institute of Applied Sciences and Technology

End of Year Project Report

Design and Implementation of a Smart Egg Incubator with IOT Integration

Prepared by :

Ghada Daghmoura

Selim Boujneh

Ghaith Ben Brahim

Yassmine Bessaad

Supervisor : Pr. Abedrrahmen Ben Chaaben

Examiner : Pr. Sahbi Bellamine

2024/2025

Acknowledgements

We would like to express our deepest gratitude to Mr. Abderrahmen Ben Chaabene for his invaluable guidance, continuous support, and insightful advice throughout the duration of this project. His expertise, encouragement, and availability have been instrumental to our progress and have greatly enriched our learning experience.

We also extend our sincere thanks to all the faculty members and administrative staff of the Physics Department, whose support and assistance created an environment conducive to research and innovation. Special thanks go to the technical staff for their help with the resources and laboratory facilities, without which the practical aspects of this work would not have been possible.

We are equally grateful to our colleagues and classmates for their collaboration, constructive discussions, and moral support, which contributed to creating a positive and motivating atmosphere throughout the project.

Finally, we express our heartfelt appreciation to our families and friends for their unwavering support, patience, and encouragement during this challenging but rewarding journey. Their understanding and belief in us was a constant source of strength and motivation.

To all those who contributed, directly or indirectly, to the success of this work we extend our sincerest thanks.

Contents

List of Figures

List of Acronyms

- **ESP32**: Espressif System-on-Chip
- **IoT**: Internet of Things
- **DHT11/DHT22**: Digital Humidity and Temperature sensor
- **PID**: Proportional–Integral–Derivative
- **LCD**: Liquid Crystal Display
- **GPIO**: General Purpose Input/Output
- **I2C**: Inter-Integrated Circuit
- **UART**: Universal Asynchronous Receiver/Transmitter
- **STM32**: STMicroelectronics 32-bit microcontroller
- **RH**: Relative Humidity
- **HAL**: Hardware Abstraction Layer
- **RTOS**: Real-Time Operating System
- **HTTP**: Hypertext Transfer Protocol
- **Wi-Fi**: Wireless Fidelity
- **LED**: Light Emitting Diode
- **IDE**: Integrated Development Environment

General Introduction

Egg incubation is a fundamental process that impacts food security, sustainable agriculture, biodiversity conservation, and educational development across the globe. In many rural and developing regions, poultry farming is a critical source of income and nutrition, offering a relatively low-cost means to produce high-quality protein. However, successful egg incubation requires maintaining precise environmental conditions, particularly temperature, humidity, and ventilation, which are often difficult to control manually.

While commercial incubators exist, they are often expensive, inaccessible to small-scale farmers, and lack adaptability or openness for educational use. Manual methods, on the other hand, are prone to human error, leading to low hatch rates, resource waste, and limited scalability. In a world increasingly focused on efficiency, sustainability, and technological equity, there is a growing need for intelligent, and connected incubation systems.

This project addresses that need by developing a smart egg incubator that integrates embedded hardware, wireless IoT communication, and a mobile application. The goal is to offer a reliable, automated, and user-friendly system capable of monitoring and adjusting incubation parameters in real-time. Designed with accessibility in mind, it targets local farmers, biology educators, and students, providing both a practical tool and a learning platform.

Beyond technical achievement, the system serves a broader vision: enabling food production independence, enhancing STEM education, and supporting small-scale innovation. The solution bridges the gap between modern smart technologies and grassroots agricultural practices empowering users and improving hatch outcomes.

The report is structured as follows:

- **Chapter 1**: Introduction and Problem Definition.
- **Chapter 2**: Smart Egg Incubator Hardware and Software Design.
- **Chapter 3**: Realization, Results and Analysys

Chapter 1

Context and Problem Definition

1.1 Problem Statement

Manual egg incubation methods present several challenges that hinder efficiency and reliability. Successful incubation requires precise control of environmental parameters, which is difficult to maintain manually and can significantly impact hatch rates.

The main challenges of manual and existing low-cost incubation methods include:

- **Labor-Intensive Monitoring**: Maintaining stable temperature and humidity requires constant supervision and frequent manual adjustments, making the process time-consuming and vulnerable to human error.
- **Environmental Fluctuations**: Inconsistent temperature and humidity levels can lead to increased embryonic mortality and developmental anomalies.
- **Manual Egg Turning**: Essential for preventing the embryo from adhering to the shell, manual egg turning is tedious and often performed inconsistently, affecting hatch success.
- **Accessibility Issues**: While commercial incubators provide automated solutions, their high cost makes them inaccessible to small-scale farmers, educational institutions, and individual users.

- **Technical Limitations of Low-Cost Systems**: Affordable DIY alternatives often lack precision, reliability, and essential features such as remote monitoring and data logging, which are necessary for process optimization and early fault detection.

There is, therefore, a clear need for an affordable, intelligent egg incubator that:

- Automates critical incubation tasks,
- Ensures stable environmental control,
- Enables real-time monitoring and data collection.

Such a system would provide a practical and accessible alternative, bridging the gap between manual methods and costly commercial systems.

1.2 State of the Art

Artificial egg incubation, the practice of hatching eggs outside natural brooding, requires precise environmental control. Modern incubation technology builds upon centuries of practice, evolving from basic heat management to sophisticated systems [1]. Success hinges on managing key parameters: **Temperature** (critical for poikilothermic embryos, typically 99–100°F or 37.2–37.8°C in forced-draft systems, with stability being crucial) [1, 2], **Humidity** (managed to control water loss, often around 60% RH rising to 70–75% for hatching) [1, 3], **Rotation** (essential to prevent adhesion and ensure uniform development, automated in most modern systems) [1, 4], and **Ventilation** (supplying oxygen and removing CO_2) [5]. Incubation times vary by species (e.g., 21 days for chickens) [1].

Solutions range from large-scale **Commercial Incubators**, offering high reliability and automation but at significant cost and with proprietary controls [5, 6], to **Open-Source/DIY Projects**. DIY approaches, often using platforms like ESP32 or Arduino, offer cost savings and high customization, allowing integration with IoT platforms (like InfluxDB and Grafana for dashboards) for remote monitoring and control [7, 8]. However, achieving commercial-level reliability requires careful design and testing [6, 7].

The core technologies include **Sensors** (DHT22 for temperature/humidity, CO_2 sensors, IR thermometers, encoders for turning feedback) [5,8], **Actuators** (heaters, fans, humidifiers, motors controlled via relays or solid-state relays) [5,7], **Microcontrollers** (ESP32 favored for integrated Wi-Fi/Bluetooth capabilities) [5, 8], and advanced **Control Methods** (PID controllers or fuzzy logic for maintaining system stability) [8, 9].

Modern systems leverage **IoT Architectures** (Device-Cloud-Control model) for remote access, data logging, alerts, and enhanced fault tolerance using cloud monitoring and redundant systems [5].

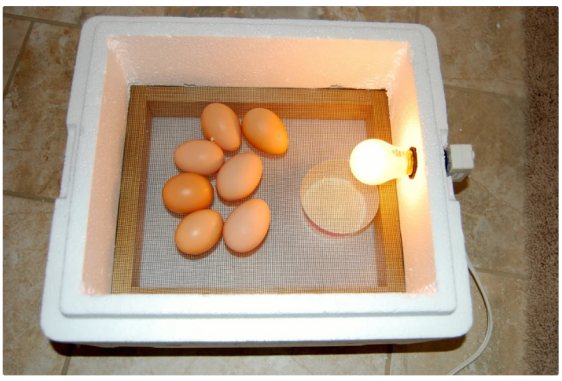

Figure 1.1: Smart egg incubator prototype featuring basic environmental control (adapted from Mujcic et al., 2021).

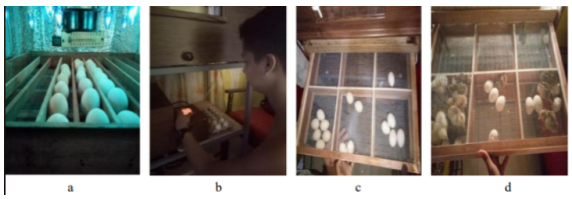

Figure 1.2: Advanced IoT-based incubator design with real-time monitoring (adapted from Petkov et al., 2023).

1.3 Motivation and Interest

The development of a smart, connected egg incubator addresses both educational and practical needs.

Educational Value From an academic perspective, the project serves as a multidisciplinary learning opportunity, combining:

Embedded systems design and microcontroller programming (STM32, ESP32).

Sensor integration and real-time data acquisition.

Actuator control for environmental management.

IoT implementation for cloud-based data storage and visualization (InfluxDB, Grafana).

Mobile application development for real-time user monitoring (Flutter framework).

This practical exposure to modern technologies enhances students' engineering skills and prepares them for complex system design challenges.

Practical Value For small-scale poultry farmers and educational institutions:

Automation reduces manual labor and improves hatch rates.

Remote monitoring enables early detection of faults and corrective action.

Data logging facilitates better control and optimization of incubation conditions.

The proposed system aims to offer a cost-effective, reliable, and user-friendly solution that fills the gap between manual processes and expensive commercial incubators.

1.4 Limitations

While the proposed system provides numerous advantages, several limitations are acknowledged:

- **Budget Constraints**: The use of low-cost sensors and actuators may limit measurement precision and long-term system durability.
- **Sensor Accuracy**: Devices such as the DHT22 exhibit inherent limitations in accuracy and reliability, requiring periodic calibration to maintain acceptable performance.
- **Hardware Robustness**: DIY assemblies may lack the mechanical stability and quality control standards found in commercial incubators.
- **Algorithm Complexity**: Implementing and tuning advanced control algorithms, such as PID controllers or fuzzy logic systems, can be time-consuming and require specialized expertise.

- **Limited Testing Scope**: Due to project time constraints, long-term validation across multiple incubation cycles and different egg species could not be fully performed.
- **Network Dependency**: The IoT functionalities rely on stable Wi-Fi connectivity and cloud services, which may not always be available in rural or remote areas.
- **Scalability Constraints**: The current prototype is optimized for small-scale use; scaling the system to handle larger egg batches would necessitate significant redesign of heating, ventilation, and control systems.

These limitations highlight areas for future improvement and serve as important considerations for subsequent iterations of the system.

Chapter 2

Smart Egg Incubator Hardware and Software Design

2.1 System Description

The egg incubator is a custom-built system designed to maintain ideal environmental conditions for the artificial incubation of eggs. It is housed within a wooden box with a top-opening roof, combining controlled heating, ventilation, and humidity regulation to ensure optimal conditions for embryo development. The egg incubator is designed to maintain optimal environmental conditions for successful artificial incubation: primarily temperature and humidity. Heating is provided by two light bulbs that turn on and off automatically to keep the temperature within the required range. These bulbs also heat water in containers placed below them, which increases humidity through evaporation. To ensure even distribution of warm, moist air, an internal fan continuously circulates the air inside the box. A sliding vent on the roof, controlled by a stepper motor using a screw-nut mechanism, opens or closes gradually in response to humidity readings or trends. This system provides smooth and precise humidity control by allowing excess moisture to escape or by sealing the box to retain it. All components work together to create and maintain a stable environment that supports embryo development and successful hatching. For the sake of simplicity, the egg-turning process in this experimental prototype is manual

and not automated.

2.1.1 Main Components and Their Roles

Wooden Box

The incubator is built using a wooden box that serves as the main structure and insulating enclosure. Its compact, closed design helps maintain a stable internal environment for temperature and humidity. The box features a top-opening roof for easy access and management of the internal components and eggs.

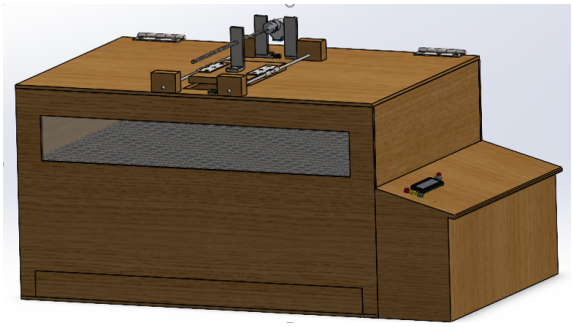

Figure 2.3: Wooden Box - Main incubator structure with roof vent mechanism and control panel

Bottom Drawer with Water Containers

At the bottom of the incubator, there is a sliding drawer containing several separate water containers that generate humidity through evaporation. This setup gives the user flexibility to adjust humidity by choosing how many containers to fill, more containers mean a greater total surface area for evaporation. The drawer can be easily pulled out and refilled without disturbing the eggs, and the water in each container is heated by the light bulbs above to raise the humidity inside the box.

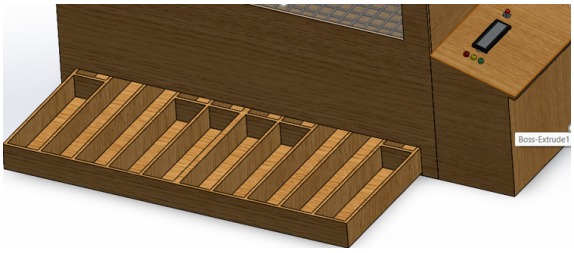

Figure 2.4: Bottom Drawer with Water Containers - Sliding drawer with compartmentalized water containers for humidity control

Light Bulbs

Two light bulbs are mounted inside the box above the water containers, acting as the primary heat source. They serve a dual purpose: maintaining the temperature needed for incubation and promoting evaporation of the water below to help regulate humidity levels. Their operation is controlled automatically to keep the temperature within the desired range.

Metallic Fence (Grill)

A metallic grill is positioned horizontally above the light bulbs and water containers, providing a stable surface to hold the eggs. This setup allows warm, humid air to rise and circulate evenly around the eggs, while keeping them safely suspended above the heating elements and water.

Internal Fan

An electric fan is installed on the inner right side of the box to ensure proper air circulation. By constantly mixing the air, the fan distributes heat and humidity evenly throughout the incubator, preventing hot or cold spots and maintaining consistent conditions for all the eggs.

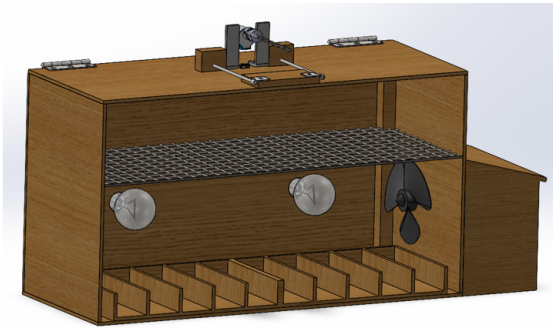

Figure 2.5: Internal Fan - Cutaway view showing internal components including fan, metallic grill, light bulbs, and water containers

Roof Vent with Stepper Motor

To control excess humidity, the roof features a small opening covered by a sliding door operated by a stepper motor using a screw-nut mechanism. The system

automatically opens or closes the vent based on current humidity readings or trends, allowing smooth and gradual adjustments to maintain the desired humidity range.

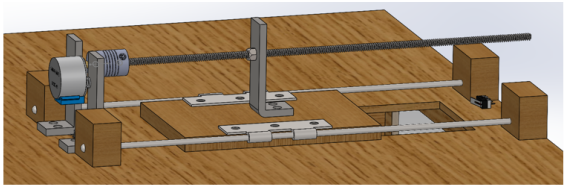

Figure 2.6: Roof Vent with Stepper Motor - Detailed view of the screw-nut mechanism for vent control

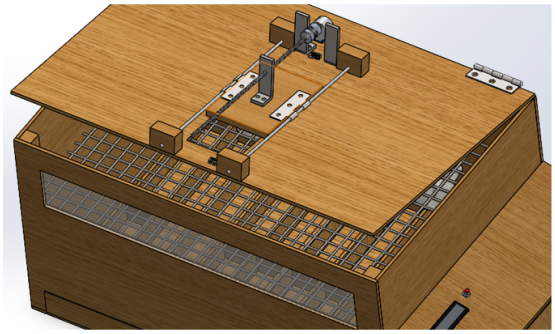

Figure 2.7: Roof Vent with Stepper Motor 2 - Top view showing vent integration into roof structure

2.2 Hardware Components Description

2.2.1 Overview

The smart incubator uses an STM32 microcontroller with sensors and actuators to regulate internal conditions. An ESP32 module ensures remote monitoring via a dashboard.

2.2.2 Microcontrollers

NUCLEO-F401RE

- **Type:** 32-bit Microcontroller (ARM Cortex-M4) development board
- **Function:** Acts as the main controller, managing sensor readings, actuator control, LCD updates, and multitasking via FreeRTOS.

- **Description:** Based on STM32F401RE ARM Cortex-M4, this Nucleo board offers multiple I/Os and supports programming via ST-LINK.

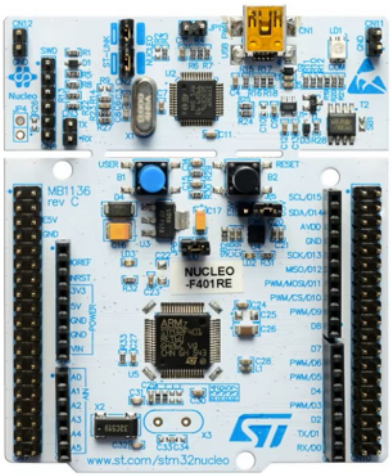

Figure 2.8: STM32 Development Board NUCLEO-F401RE

ESP32-WROOM-32D Module

- **Type:** Microcontroller with Wi-Fi & Bluetooth
- **Function:** Handles wireless communication; receives system data from STM32 via UART and responsible for transmitting this data to a remote server.
- **Description:** A dual-core Xtensa-based microcontroller from Espressif. It is commonly used in IoT applications thanks to built-in Wi-Fi, Bluetooth, and power-saving features.

Figure 2.9: ESP32-WROOM-32D Module

2.2.3 Sensors

DHT11

The DHT11 sensor measures temperature and humidity using two internal components:

A thermistor for temperature: which changes resistance as the temperature changes.

A capacitive sensor for humidity: which detects changes in moisture in the air. The sensor reads these values, converts them into digital data, and sends the information through a single data wire to the microcontroller (like the STM32). It updates the readings about once per second.

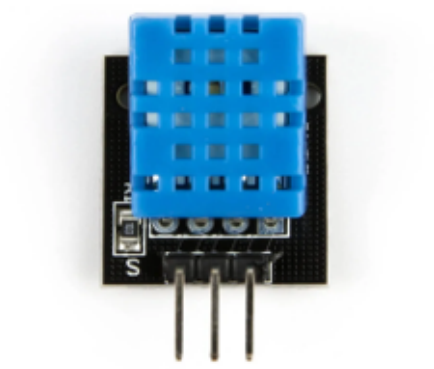

Figure 2.10: DHT11 Sensor

Limit Switches (x2)

- **Type:** Digital Mechanical Sensor
- **Function:** Detect the end limits of the stepper motor's motion, providing safe boundaries for mechanical movement.
- **Description:** Simple NO (Normally Open) switches that close the circuit when pressed. Commonly used in CNC and robotic systems for position feedback.

Figure 2.11: Limit Switches

2.2.4 Actuators

Stepper Motor (28BYJ-48 + ULN2003 driver)

- **Type:** Electromechanical Actuator
- **Function:** Rotates incrementally in response to STM32 signals to control movement (e.g., opening/closing a door).
- **Description:** The motor is driven via the ULN2003 driver. It is precise and ideal for step-based control. Two limit switches define its range of motion.

Figure 2.12: Stepper Motor 28BYJ-48 + ULN2003 driver

Relay Modules (x2)

- **Type:** Switching Actuator

- **Function:** One relay toggles the fan and the other the light bulb, based on sensor readings or control logic.
- **Description:** Electromagnetic relays that allow low-voltage GPIO pins to control high-voltage AC loads. Includes an optocoupler for isolation and a driver transistor for current gain.

Figure 2.13: Relay Module

2.2.5 Indicators & Outputs

LEDs (x3)

- **Type:** Visual Indicators
- **Function:** Provide visual feedback for fan status, bulb status, and overall system operation.
- **Description:** Standard 5mm LEDs (Red, Green, or Yellow). Connected to GPIO pins via current-limiting resistors. Turned ON/OFF by the STM32.

Figure 2.14: LEDs

16x2 LCD display

- **Type:** Text-based display module
- **Function:** Shows real-time sensor data (temperature, humidity), actuator states (fan/bulb), and system status.
- **Description:** This LCD is used to provide real-time information and status updates to the user. This type of display is common for embedded systems due to its simplicity and effectiveness in presenting textual data. The LCD is interfaced with the STM32 via an I2C driver, PCF8574 which simplifies the wiring to the LCD by converting parallel data to serial I2C communication.

Figure 2.15: 16x2 LCD Display and I2C Driver

2.2.6 Electrical Wiring

The STM32 NUCLEO-F401RE board acts as the system's central controller, connected to various sensors and actuators. All components share a common ground, and power is distributed through 3.3V or 5V lines depending on the module.

- The DHT11 sensor connects to a GPIO pin with a pull-up resistor.
- The I2C LCD connects via SDA and SCL lines.
- The ESP32 is connected over UART.
- The stepper motor is driven by a ULN2003 board via 4 GPIOs.
- Limit switches are used to detect motor end positions.
- Relays (for fan and bulb) are controlled by GPIOs.
- LEDs are connected to GPIOs with resistors.

All wiring ensures safe operation and stable signal transmission. A schematic diagram is provided below.

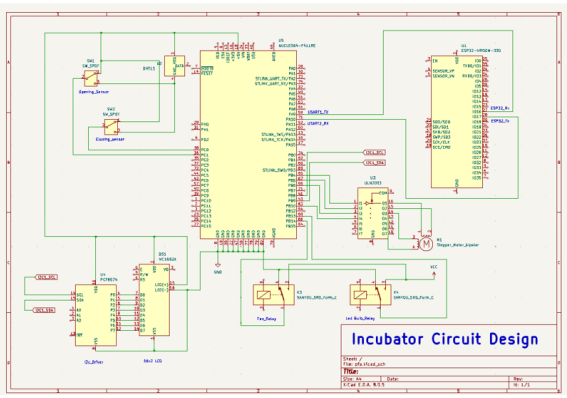

Figure 2.16: Electrical Wiring Diagram Of The Smart Incubator

2.3 Embedded Software

2.3.1 Introduction

This project involves designing embedded software for environmental control using an STM32 microcontroller running FreeRTOS to manage multiple concurrent tasks.

An ESP32 handles wireless communication via Wi-Fi. The system's main objectives are reliable sensor readings (temperature and humidity), precise motor control, user interface updates, and efficient wireless data transmission.

FreeRTOS was selected for its real-time task management, inter-task communication, and memory efficiency, ensuring a responsive and well-organized embedded system.

2.3.2 FreeRTOS Overview

The benefits of using FreeRTOS in embedded systems include improved system responsiveness, modularity, easier debugging, and efficient resource utilization.

FreeRTOS is an open-source real-time operating system designed for embedded systems. It manages concurrent tasks efficiently with priority-based scheduling. Main features include:

- **Tasks:** Independent units of execution running concurrently.
- **Queues:** Safe communication channels between tasks.
- **Semaphores:** Tools for synchronizing tasks and protecting shared resources.

FreeRTOS improves system responsiveness, modularity, debugging, and resource use in embedded applications.

2.3.3 STM32 Embedded Firmware

STM32CubeMX Project Configuration

- **Clock Configuration:** The system clock was configured to ensure optimal performance and timing for all peripherals and FreeRTOS operations.
- **Peripheral Initialization:** Key peripherals were initialized through STM32CubeMX, including:
- **GPIO:** Configured for controlling LEDs (Red, Yellow, Green), fan and bulb relays, and for reading input from limit switches for the stepper motor.
- **UART:** Two UART interfaces were set up. One (huart2) for debugging and logging messages to a console, and another (huart1) specifically for communication with the ESP32 module.

- **I2C:** An I2C interface (hi2c1) was configured for communication with I2C-compatible sensors or display modules. The `scan_i2c_bus()` function in the code confirms its use for device detection.

- **Timers:** A timer (htim1) was initialized, likely for generating precise timing signals required for the stepper motor control.

FreeRTOS Middleware Enablement and Configuration: FreeRTOS was enabled as a middleware component in STM32CubeMX. This allowed for the configuration of the RTOS kernel parameters, such as the tick rate, heap size, and the initial setup of tasks, queues, and semaphores directly within the graphical interface.

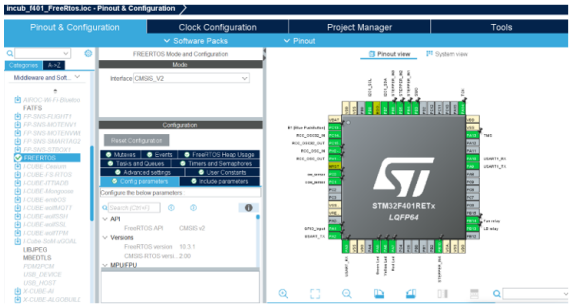

Figure 2.17: STM32CubeMX Graphical Configuration

Peripheral Roles in the System

Each configured peripheral plays a vital role in the overall system functionality:

- **UART for STM32-ESP32 Communication:** (huart1) is dedicated to establishing a reliable serial communication link between the STM32 and ESP32.

- **I2C for LCD or Sensors:** The I2C bus (hi2c1) is used for interfacing with I2C devices. Based on the code, it's used for an LCD (`lcd_init()`, `UpdateDisplay()`) and potentially other I2C sensors.

- **GPIO for Relays, LEDs, Limit Switches:** GPIO pins are extensively used for direct control and sensing. This includes driving relays for the fan and bulb, illuminating status LEDs, and monitoring the physical limits of the stepper motor via limit switches.

- **Timers for Motor Control:** The configured timer (htim1) is crucial for generating the precise pulse sequences required to control the stepper motor, ensuring accurate and smooth movement.

HAL vs Low-level Drivers: Rationale for Choice

The project primarily utilizes the STM32Cube HAL (Hardware Abstraction Layer) drivers. The HAL provides a high-level, user-friendly API that simplifies interaction with the STM32 peripherals. This choice accelerates development by abstracting complex register-level programming, making the code more readable and portable across different STM32 microcontrollers.

While low-level drivers offer finer control and potentially higher performance, the HAL was deemed sufficient and more efficient for the rapid development and maintenance of this class project, especially when combined with the FreeRTOS middleware.

2.3.4 FreeRTOS Task Design

The application is structured around several FreeRTOS tasks, each responsible for a specific function, enabling concurrent execution and modularity. The main.c file clearly defines these tasks and their attributes.

List of Main Tasks with Responsibilities

- **defaultTask:** This is a low-priority task, often serving as an idle task or for general system maintenance when no other higher-priority tasks are ready to run.

- **SensorCommTask:** Responsible for reading data from sensors, specifically the DHT11 for humidity and temperature. It likely includes data filtering and processing before making the data available to other tasks.

- **StepperTask:** Manages the stepper motor, including controlling its movement (opening/closing vents or similar mechanisms) and monitoring its limit switches to prevent over-rotation.

- **DisplayTask:** Handles updating the connected LCD, presenting real-time sensor data, system status, and other relevant information to the user.
- **FanTask:** Controls the operation of the fan based on environmental conditions (e.g., humidity levels) to regulate air circulation.
- **BulbTask:** Manages the light bulb, potentially for illumination or as a heating element, based on temperature conditions.
- **LEDTask:** Controls the Red, Yellow, and Green LEDs to provide visual status feedback, indicating whether environmental parameters are within acceptable ranges.

Task Priorities and Scheduling Considerations

FreeRTOS assigns priorities to tasks, influencing their execution order. Higher-priority tasks preempt lower-priority tasks. The provided code defines the following priorities:

- **SensorCommTask:** osPriorityHigh - This indicates that sensor data acquisition is critical and needs to be processed promptly.
- **StepperTask:** osPriorityNormal - These tasks have standard priority, allowing them to run when higher-priority tasks are blocked or complete.
- **FanTask, BulbTask, DisplayTask:** osPriorityLow - These tasks are less time-critical and will execute when resources are available.
- **LEDTask:** osPriorityBelowNormal - This task has the lowest priority, indicating that LED updates are not as critical as other system functions.

This priority assignment ensures that critical operations like sensor communication are handled swiftly, while less urgent tasks like display updates or LED indications are processed in the background, maintaining system responsiveness.

Task Stack Sizes and Memory Allocation

Each FreeRTOS task requires its own stack for local variables, function call contexts, and interrupt handling. The main.c file specifies the following stack sizes (in words, where 1 word = 4 bytes on a 32-bit system):

- **defaultTask:** 128 × 4 bytes
- **SensorCommTask:** 512 × 4 bytes
- **StepperTask:** 512 × 4 bytes
- **FanTask:** 256 × 4 bytes
- **BulbTask:** 256 × 4 bytes
- **LEDTask:** 256 × 4 bytes
- **DisplayTask:** 256 × 4 bytes

The larger stack sizes for SensorCommTask and StepperTask suggest that these tasks might involve more complex computations, deeper function call hierarchies, or handle larger data buffers compared to the other tasks. Proper stack sizing is crucial to prevent stack overflows, which can lead to unpredictable system behavior.

The task priorities and stack sizes are reflected in the FreeRTOS task declarations and creation within the code. Below is an example of how the SensorCommTask is defined and created, illustrating the use of task attributes such as stack size and priority:

```
/* Definitions for SensorCommTask */
osThreadId_t SensorCommTaskHandle;
const osThreadAttr_t SensorCommTask_attributes = {
  .name = "SensorCommTask",
  .stack_size = 512 * 4,
  .priority = (osPriority_t) osPriorityHigh,
};
/* creation of SensorCommTask */
SensorCommTaskHandle = osThreadNew(StartSensorCommTask, NULL,
                                   &SensorCommTask_attributes);
```

2.3.5 Main Program Flow and RTOS Task Interaction

The `main()` function serves as the entry point of the application. It begins by initializing the STM32 HAL library, configuring the system clock, and setting up the

necessary peripherals. Following this, critical hardware components and communication interfaces are initialized through functions such as `DHT11_Init()`, `scan_i2c_bus()`, and `lcd_init()`. These initializations occur before the FreeRTOS scheduler starts, ensuring that all essential components are fully operational.

Once the hardware setup is complete, the application transitions from a traditional polling-based, bare-metal approach to a more robust and scalable RTOS-driven system. This is evident in the commented-out code in `main()`, which shows prior polling logic replaced by FreeRTOS task management.

The FreeRTOS kernel is initialized with `osKernelInitialize()`, after which all application tasks are created using `osThreadNew()`. These tasks include:

- **defaultTask** — an idle-like background task
- **StepperTask** — manages stepper motor control
- **FanTask** — controls the fan based on sensor data
- **BulbTask** — controls lighting or heating relays
- **LEDTask** — updates LED indicators
- **SensorCommTask** — periodically reads sensor data and handles communication
- **DisplayTask** — updates the LCD display

Finally, the scheduler is started with `osKernelStart()`, which hands over CPU control to FreeRTOS. From this point, the kernel manages task execution according to their priorities and runtime states.

Task Execution and Synchronization

The SensorCommTask plays a pivotal role by periodically acquiring temperature and humidity data from the DHT11 sensor. After successfully reading the data and performing necessary calculations, it transmits formatted data over UART interfaces both for debugging and for communication with an ESP32.

Once new sensor data is obtained, SensorCommTask resumes the FanTask, BulbTask, LEDTask, and DisplayTask using the following calls:

```
osThreadResume(FanTaskHandle);
osThreadResume(BulbTaskHandle);
osThreadResume(LEDTaskHandle);
osThreadResume(DisplayTaskHandle);
```

This mechanism signals these dependent tasks to process the fresh sensor data.

Each of these tasks executes its respective function — updating fan speed, toggling relays, refreshing LED states, or redrawing the display — and then suspends itself to wait for the next resumption by SensorCommTask:

```
osThreadSuspend(FanTaskHandle);
osThreadSuspend(BulbTaskHandle);
osThreadSuspend(LEDTaskHandle);
osThreadSuspend(DisplayTaskHandle);
```

This suspend-resume pattern creates an efficient producer-consumer relationship where:

- SensorCommTask acts as the producer, gathering data periodically.
- The other tasks act as consumers, performing actions triggered by new data availability.

Meanwhile, the StepperTask runs continuously with a short delay to provide responsive real-time control of the stepper motor, independent of the suspend-resume cycle.

The defaultTask functions as a background idle task, yielding CPU time with a minimal delay, allowing higher-priority tasks to run as needed.

2.4 High-Level Code and Dashboard Architecture

This section outlines the high-level system connecting the STM32-based application to a cloud dashboard. It includes an ESP32 for HTTP communication, InfluxDB for data storage, and Grafana for visualization and alerts.

2.4.1 ESP32 HTTP and UART Logic

Role of ESP32

The ESP32 module functions as a network gateway between the STM32 microcontroller and the cloud infrastructure. It receives real-time environmental data from the STM32 over UART and forwards it to an InfluxDB instance via HTTP.

Wi-Fi Initialization

In the `setup()` function, the ESP32 connects to a Wi-Fi network using the credentials defined in the code. It uses `Serial.begin()` for debugging and `Serial2.begin()` for UART communication with the STM32:

```
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
```

UART Communication

The `loop()` function constantly checks if data is available on Serial2, which is connected to the STM32. When a complete line of data is received, it is parsed using `sscanf()`:

```
String uartData = Serial2.readStringUntil('\n');
int result = sscanf(uartData.c_str(),
    "T:%fF H:%f%% Tmin:%f Tmax:%f Hmin:%f Hmax:%f Uptime:%s Door:%s",
    &temperature, &humidity, &tmin, &tmax, &hmin, &hmax, uptime, door_state);
```

HTTP POST to InfluxDB

Once the data is successfully parsed, the ESP32 formats it according to the InfluxDB Line Protocol and sends it via HTTP POST:

```
String data = "environment,sensor=stm32 temperature=" + String(temperature) +
              ",humidity=" + String(humidity) + ",tmin=" + String(tmin) +
              ",tmax=" + String(tmax) + ",hmin=" + String(hmin) +
```

```
            ",hmax=" + String(hmax) + ",uptime=\"" + String(uptime) +
            "\",door_state=\"" + String(door_state) + "\"";

http.POST(data);
```

2.4.2 InfluxDB Setup and Communication

InfluxDB Overview

InfluxDB is a time-series database optimized for handling timestamped data, making it ideal for IoT applications. It stores sensor readings with precise timestamps, enabling efficient querying and analysis of historical trends.

Database Configuration

The InfluxDB instance is configured with:

- **Organization:** A logical grouping for data management
- **Bucket:** `incub_dataa` - the storage container for sensor data
- **Token:** Authentication credential for secure API access

Line Protocol Format

Data is sent to InfluxDB using the Line Protocol format:

```
environment,sensor=stm32 temperature=37.5,humidity=65.0,tmin=35.2,
tmax=38.1,hmin=60.5,hmax=70.2,uptime="00:03:10:24",
door_state="Opening"
```

This format includes:

- **Measurement:** `environment`
- **Tags:** `sensor=stm32` (indexed metadata)
- **Fields:** Actual sensor values and system status
- **Timestamp:** Automatically added by InfluxDB

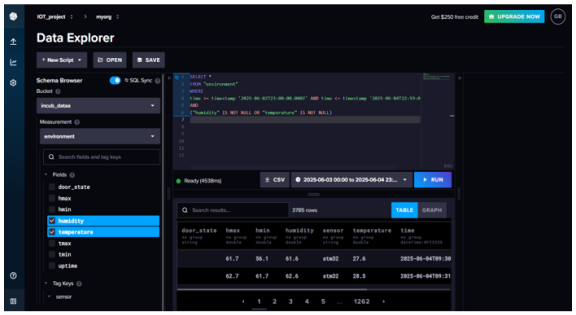

Figure 2.18: Data Explorer Interface Of InfluxDB

Advantages of Using InfluxDB

- **Time-series optimization:** Efficient storage and querying of timestamped data
- **High write throughput:** Handles frequent sensor updates without performance degradation
- **Built-in retention policies:** Automatic data lifecycle management
- **SQL-like query language:** Familiar syntax for data analysis
- **Grafana integration:** Seamless connection for visualization

2.4.3 Grafana Dashboard and Alert System

Dashboard Overview

Grafana provides a comprehensive visualization platform for the incubator monitoring system. The dashboard includes multiple panels displaying real-time and historical data.

Real-time Temperature and Humidity Gauges

Gauge panels provide immediate visual feedback on current environmental conditions:

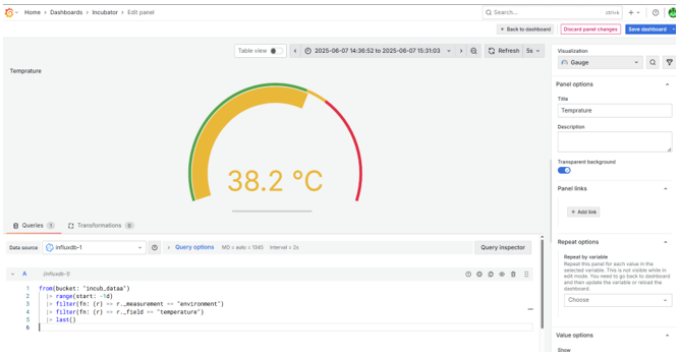

Figure 2.19: Real Time Temperature Gauge Panel in Grafana

Time-Series Line Graphs: Time-series graphs help in identifying historical trends and fluctuations in temperature and humidity. This panel is essential for analyzing long-term behavior or anomalies, such as sudden drops or spikes.

Figure 2.20: Temperature Time-Series Line Graph

Uptime and Door State Display: A text panel displays the system uptime and the current door status (e.g., "Opening", "Closing"). This is useful for quick reference during real-time operation and can aid in identifying system resets or door malfunctions.

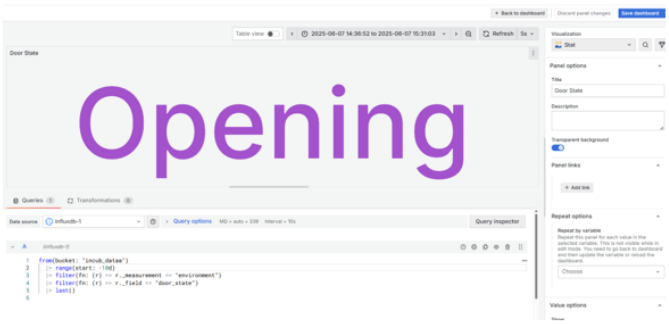

Figure 2.21: System Uptime and Door State Panel

Alert Configuration

Grafana's alerting system monitors critical parameters and sends notifications when thresholds are exceeded:

Parameter	Condition	Duration	Description
Temperature	`temperature > 38 °C`	For 2 minutes	Indicates risk of overheating that could harm system operation or contents
Humidity	`humidity > 70 %`	For 2 minutes	Warns of high humidity that may lead to condensation or sensor inaccuracy

Figure 2.22: Configured Alert Rules

Temperature Alerts: Triggered when temperature falls below 35°C or exceeds 40°C **Humidity Alerts:** Activated when humidity drops below 55% or rises above 75%

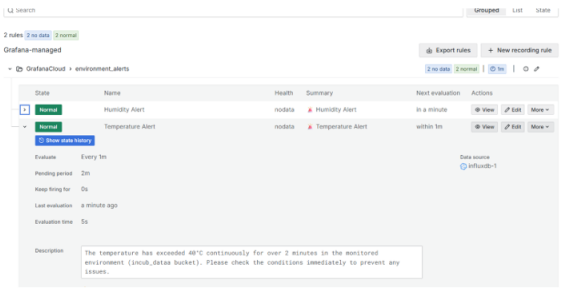

Figure 2.23: Grafana Alert Rule Setup (Temperature & Humidity)

Benefits of Grafana

- **Real-time monitoring:** Live data visualization with automatic refresh
- **Historical trend analysis:** Long-term data analysis capabilities
- **Custom dashboards:** Flexible panel arrangements and visualizations
- **Alerting based on thresholds:** Proactive notification system for critical conditions

2.4.4 Conclusion

This high-level architecture bridges the gap between low-level embedded sensing and high-level cloud monitoring. The STM32 handles real-time data acquisition, the ESP32 forwards that data to a centralized InfluxDB database, and Grafana

provides visualization and alerts. This architecture ensures reliability, scalability, and maintainability for smart environmental monitoring systems.

2.5 Mobile Application Development

As part of the smart egg incubator system, a dedicated mobile application named EggLab was developed using Flutter to serve as the primary user interface for interacting with and monitoring the incubation process. The app is designed to make the system intuitive, accessible, and practical for users.

2.5.1 Cross-Platform Development

The application was developed using Flutter, a modern and flexible framework that enables cross-platform deployment on both Android and iOS. Flutter's widget-based architecture allowed for a clean, responsive, and visually appealing interface that adapts smoothly to various screen sizes and devices.

2.5.2 Key Features and Functionalities

The EggLab application provides several key features:

- **Incubation Timer:** The app features a 21-day countdown timer for each egg batch, representing the standard incubation period. This feature helps users visually track the progress of incubation and stay informed about upcoming milestones such as candling, lockdown, and hatching.
- **Real-Time Monitoring:**

  The application connects to the incubator via Wi-Fi

  It receives live sensor data, including temperature and humidity readings, providing users with up-to-date environmental conditions inside the incubator..
- **Alerts and Reminders:** Users receive automatic notifications when:

  Temperature or humidity goes outside predefined thresholds. Itś time to turn the eggs, based on a regular schedule.

  Key incubation events are approaching.

These alerts ensure proper care is maintained without constant manual supervision.

- **Graphical Visualization:** The app displays historical temperature and humidity data in a graphical format.

  This allows users to analyze fluctuations and ensure environmental stability over time.

  Graphs are easy to interpret and help in identifying patterns or anomalies.

2.5.3 Impact and Benefits

The EggLab mobile application enhances the usability and efficiency of the smart incubator system by:

- It reduces the need for constant manual checks by providing a convenient, real-time remote interface.
- It is especially useful for small-scale farmers, schools, educators, or hobbyists who want to learn or work with egg incubation processes.
- The app makes the system more educational, practical, and user-friendly, turning a traditionally technical task into an accessible experience.

By combining modern mobile technology with embedded systems, EggLab bridges the gap between complex hardware processes and user-centric design, making smart incubation not only possible but also enjoyable

Figure 2.24: Home Page

Figure 2.25: New Batch

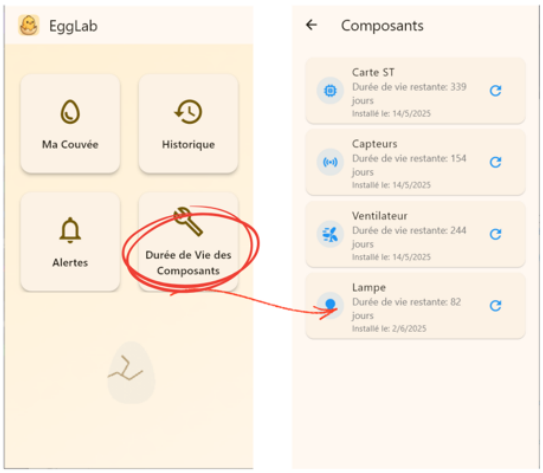

Figure 2.26: Life Cycle Of Components

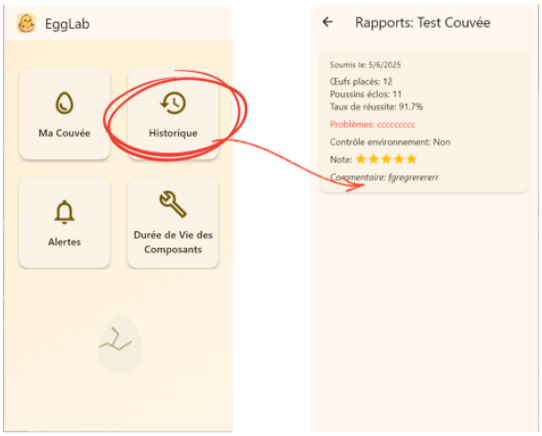

Figure 2.27: History

Chapter 3

Realization, Results and Analysys

This chapter presents the different stages involved in the practical realization of the egg incubator project. From prototyping and testing to full assembly and trial runs, it includes photos of the system, test phases, and monitoring dashboards that demonstrate the incubatorś functionality.

3.1 Prototype Assembly

3.1.1 Electronic Circuit Setup

Before integrating the components into the wooden box, the electronic circuit was first assembled and tested on a breadboard. This allowed us to validate the behavior of sensors, relays, the stepper motor, and the fan under controlled conditions.

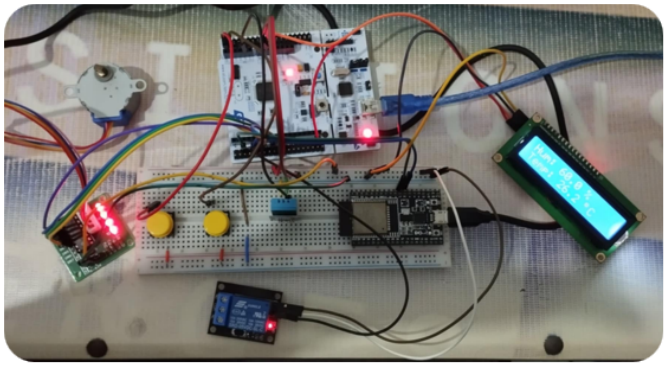

Figure 3.1: Electronic circuit Testing

3.1.2 Physical Construction

The incubator box was built from wood, with an opening top and a drawer at the bottom to hold the water containers. All components: light bulbs, fan, humidity vent system and the metallic egg support were installed according to the design.

Figure 3.2: Prototype image

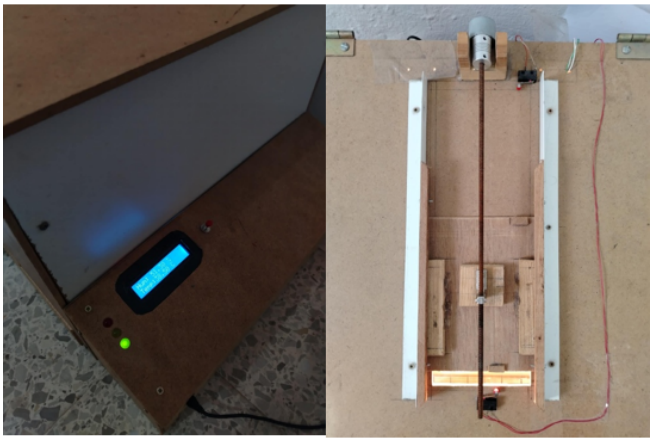

Figure 3.3: LCD and venting system testing

3.2 First Incubation Trial

3.2.1 Egg Placement

A first incubation test was carried out using fertilized eggs placed directly on the metallic support grill. Water was added to the containers to ensure sufficient humidity. The temperature and humidity control systems were monitored throughout the incubation process.

Figure 3.4: Egg Placement for incubation test

3.2.2 Hatching Results

A total of 11 eggs were placed in the incubator : 2 were unfertilized, leaving 9 fertile eggs. Out of these, 8 hatched successfully, resulting in a hatch rate of 89 percent, while 1 embryo did not hatch. Hatching began slightly earlier than expected, with two chicks emerging late on day 19, three on day 20, and the rest on day 21. This early start is likely due to the temperature being slightly higher than the standard 37.5°C, which can speed up embryo development. All hatched chicks appeared healthy, active, and responsive, showing signs of good vitality. These results confirm that the incubator maintained the right conditions for proper development and successful hatching.

Figure 3.5: Hatching results

3.3 Monitoring and Data Logging

Throughout the incubation process, temperature and humidity values were recorded and visualized in real time using a dashboard developed with Grafana. The data confirmed the stability and responsiveness of the control systems.

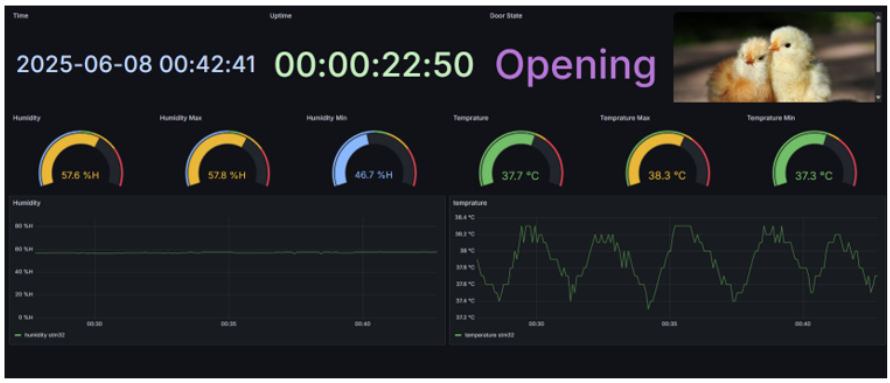

Figure 3.6: Grafana dashboard screenshot

The Grafana dashboard provides a real-time overview of the egg incubatorś vital conditions, displaying current humidity and temperature values through intuitive gauges, along with their respective maximums and minimums. Below these, time-series graphs illustrate the historical trends of both humidity and temperature, allowing for immediate visual analysis of environmental stability and the effectiveness of the automated control systems, all while indicating the current status of the D̈oor Statef̈or humidity regulation.

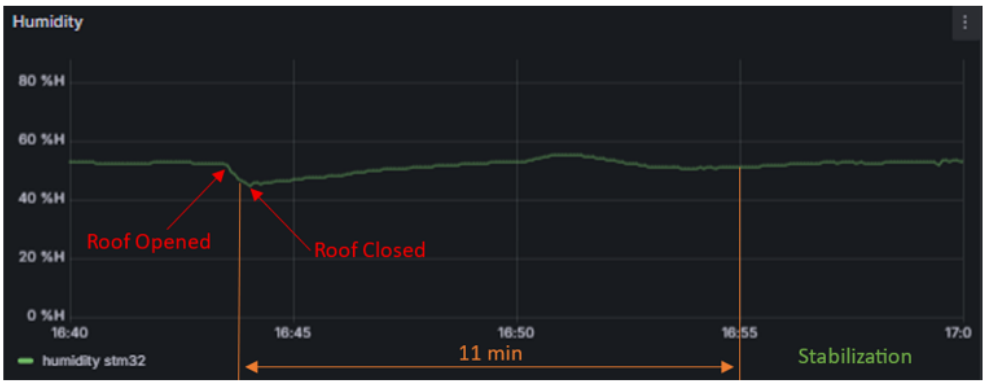

Figure 3.7: Humidity Recovery After Manual Roof Opening

At approximately 16:43, the entire incubator roof was manually opened, triggering a rapid drop in relative humidity as warm, moist air escaped. After about one minute, the roof was closed again, and the stepper motor automatically closed the small vent to retain and regain humidity. Over the following eleven minutes, the internal humidity steadily rose until it reached and held the target setpoint by 16:55. This sequence clearly demonstrates the systemś ability to recover from a major disturbance and then rely on the motorized vent to return the environment to a stable humidity level.

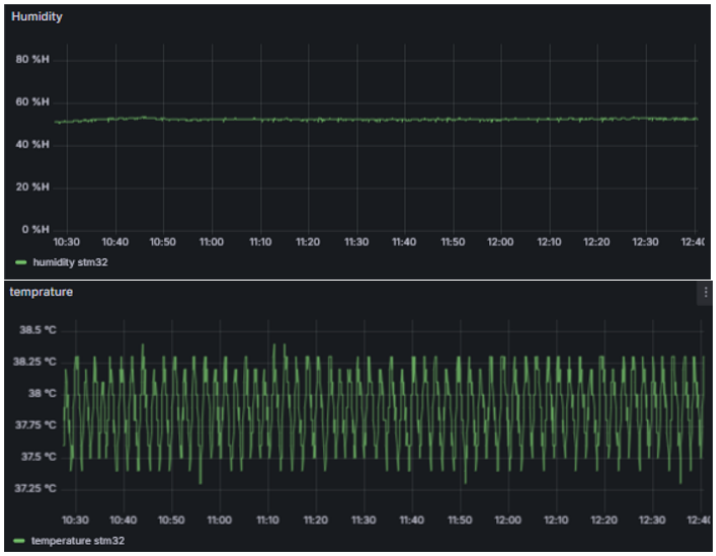

Figure 3.8: Temperature and Humidity Stability Over Time

The two figures above show, over approximately two hours, how the incubator maintained stable conditions: temperature held between 37°C and 38°C, while relative humidity stayed between 50% and 55% (around the 52.5% setpoint). The small, rapid temperature oscillations reflect the on/off cycling of the heating elements, and the nearly flat humidity curve demonstrates how the stepper-motor vent and evaporative trays work together to keep moisture levels steady.

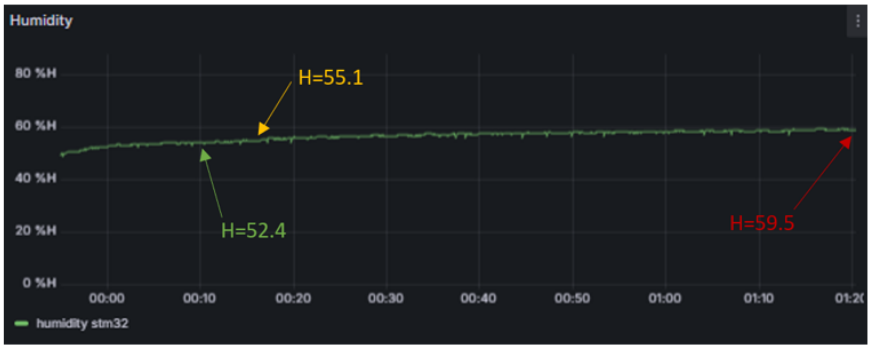

Figure 3.9: Humidity Drift Without Vent Regulation

The graph shows the effect of disabling the humidity regulation mechanism by blocking the stepper motor in the fully closed position, preventing the vent from opening. Initially, the humidity remains within the desired range (50–55%), but as evaporation continues without any means for excess moisture to escape, the relative humidity gradually increases. Over time, it exceeds the target range, highlighting the importance of active ventilation for maintaining balanced humidity levels. This result confirms the necessity of the automated vent system to prevent oversaturation and ensure a stable incubation environment.

3.4 Conclusion

This chapter presented the realization of the egg incubator prototype, from its physical assembly to the validation of its environmental control system. The integration of heating, humidification, and ventilation elements enabled the incubator to maintain temperature and humidity within the required incubation ranges. Experimental results, including real-time data visualized on a Grafana dashboard, confirmed the systemś ability to respond to environmental variations and maintain stable internal conditions. Most importantly, the incubator successfully passed its first real-world test, with several eggs hatching under controlled conditions—demonstrating not only the technical functionality of the system but also its biological effectiveness. While certain operations, such as egg turning, remain manual in this initial version, the overall outcome validates the design and provides a strong foundation for future enhancements and full automation.

General Conclusion

The development of this smart egg incubator — combining embedded hardware, IoT connectivity, and a mobile application — successfully addresses the need for a reliable, controlled, and accessible incubation system. This project highlights the strength of an interdisciplinary approach, merging electronics, embedded programming, wireless communication, and mobile development into a unified and functional solution.

By integrating the STM32 microcontroller running FreeRTOS for task management, the ESP32 for wireless data transmission, the InfluxDB time-series database for efficient data storage, and the Grafana dashboard for real-time visualization, the system enables precise environmental control and continuous monitoring. The EggLab mobile application enhances usability, providing users with intuitive, portable access to critical incubation parameters.

Real-world testing validated the system's robustness, achieving stable temperature and humidity control and demonstrating promising hatch rates. Nonetheless, certain limitations — such as manual egg turning, reliance on Wi-Fi connectivity, and limited scalability — point toward opportunities for further development.

In conclusion, this project provides a solid proof of concept for a smart, user-friendly, and cost-effective egg incubator. It opens the door to future enhancements that could lead to full automation, improved reliability, and broader applicability across agricultural and educational sectors.

Perspectives

- **Automated Egg Turning**: Integrate a fully automated egg turning mechanism to eliminate manual intervention and further increase hatch rates.
- **Offline Functionality**: Develop local data storage and offline monitoring features to reduce dependence on continuous Wi-Fi connectivity.
- **Solar Power Integration**: Add renewable energy options, such as solar panels, to improve sustainability and adaptability in off-grid environments.
- **Expanded Data Analytics**: Implement machine learning models to analyze incubation data and optimize incubation parameters dynamically.
- **Scalability Enhancements**: Redesign the system for higher-capacity incubation, supporting larger batches of eggs while maintaining environmental uniformity.
- **Industrialization Potential**: Explore the development of a commercial version with enhanced durability, certifications, and mass production feasibility.
- **Educational Platform Development**: Adapt the incubator as a teaching tool, providing modular access for students to learn about IoT, embedded systems, and biological processes.

Bibliography

[1] J. G. Berry, "Artificial Incubation," *Oklahoma State University Extension*, Fact Sheet AFS-8100, (2017).

[2] S. Yalcin and S. Uysal, "Incubation Temperature and Lighting: Effect on Embryonic Development," *Poultry Science*, vol. 101, no. 7, pp. 101532, (2022).

[3] Poultry Keeper, "Guide to Incubation Humidity," *Poultry Keeper Website*, (2025).

[4] Incubator Warehouse, "Beginner's Guide to Hatching Eggs," *Incubator Warehouse Publications*, (2015).

[5] E. Petkov, et al., "Fault Tolerant Smart Egg Incubation System with Computer Vision," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 14, no. 2, pp. 125–133, (2023).

[6] J. Gibson and A. McGowan, "Homemade vs. Commercial Incubators: Performance Analysis," *CS Science Fair Reports*, (2002).

[7] NEHERP, "DIY Incubator Guide for Reptile and Poultry Eggs," *NEHERP Website*, (2025).

[8] M. C. A. Prabowo, et al., "Development of an IoT-Based Egg Incubator with PID Control," *Journal of Informatics and Visualization (JOIV)*, vol. 8, no. 1, pp. 45–52, (2024).

[9] A. Mujcic, et al., "Development of a Fuzzy Logic-Based Temperature Controller for Egg Incubators," *Journal of Automation and Control Engineering*, vol. 9, no. 2, pp. 45–50, (2021).