BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

EEE 416 (January 2025)
Microprocessors and Embedded Systems Laboratory

Final Project Report

Section: B1 Group: 06

IoT-Based System for Real-Time Monitoring and Control of FRUIT Storage Conditions

Course Instructors:

Dr. Sajid Muhaimin Choudhury, Associate Professor
Akif Hamid, Lecturer (PT)

Signature of Instructor: ______________________

Academic Honesty Statement:

IMPORTANT! Please carefully read and sign the Academic Honesty Statement, below. Type the student ID and name, and put your signature. ***You will not receive credit for this project experiment unless this statement is signed in the presence of your lab instructor.***

"In signing this statement, We hereby certify that the work on this project is our own and that we have not copied the work of any other students (past or present), and cited all relevant sources while completing this project. We understand that if we fail to honor this agreement, We will each receive a score of ZERO for this project and be subject to failure of this course."

Signature: Puspita Mobarak **Full Name: Puspita Mobarak** **Student ID: 2006087**	**Signature:** Al Amin **Full Name: Md. Al Amin** **Student ID: 2006088**
Signature: Muaz R. **Full Name: Md. Muaz Rahman** **Student ID: 2006089**	**Signature:** [illegible] Hosan **Full Name: Mohammad Al Hosan** **Student ID: 2006090**

Table of Contents

1 Abstract

This project aims to design an IoT-based system for real-time monitoring and control of fruit storage conditions. The system uses sensors to track temperature, humidity, and air quality inside a storage container. Based on predefined threshold values, it automates misting, cooling, and ventilation using relay-controlled modules. Sensor readings are displayed live on an LCD screen, allowing users to observe environmental changes instantly. The overall goal is to reduce spoilage and maintain optimal conditions, thereby improving storage efficiency for fruits.

2 Introduction

Post-harvest losses are a significant concern in agricultural systems, especially in countries like Bangladesh where storage infrastructure is limited. A considerable portion of fruits deteriorate due to suboptimal storage conditions such as excessive humidity, high temperatures, or gas accumulation from ripening or spoilage. These environmental factors not only accelerate the degradation process but also make fruits more susceptible to pests and microbial growth. Without proper intervention, a large amount of produce becomes unsuitable for consumption or sale, contributing to economic loss and food insecurity.

Traditionally, monitoring of storage conditions has relied on manual checks — measuring temperature or inspecting fruit visually. However, this method is slow, inconsistent, and labour-intensive. In large-scale or even household-level storage setups, manual monitoring is rarely conducted with the frequency or precision needed to catch early signs of spoilage. As a result, spoilage often goes undetected until it becomes severe, leading to preventable waste.

To address this issue, our project proposes an IoT-based solution that enables real-time, automated monitoring of fruit storage environments. By continuously tracking parameters such as temperature, humidity, and air quality, the system can react proactively using cooling, misting, or ventilation mechanisms based on predefined thresholds. This not only reduces the need for manual intervention but also creates a controlled environment that helps preserve fruit freshness, reduce waste, and improve overall storage efficiency.

3 Design

3.1 Problem Formulation (PO(b))

3.1.1 Identification of Scope

- **Sensing Capabilities:** The system continuously monitors the storage environment using two key sensors. The DHT22 tracks both temperature and humidity to ensure conditions stay within a safe range, while the MQ135 provides a general sense of air quality by detecting harmful gases. Together, they help maintain a healthy and stable atmosphere for whatever is being stored.

- **Actuator Systems:** The controller manages three key environmental regulation systems, each operating independently and automatically:

  1. **Temperature Control** uses a thermoelectric Peltier module along with a cooling fan to maintain optimal temperature levels within the storage environment.
  2. **Humidity Control** represented by an LED indicator, this system simulates the monitoring and regulation of humidity conditions.
  3. **Air Quality Control** operates through a dedicated ventilation fan that helps improve air circulation and maintain cleaner air by reducing pollutants.

- **Control System and Logic:** The project features an intelligent control system that operates automatically based on predefined threshold limits set earlier in the firmware. It continuously monitors environmental conditions and activates the appropriate actuators whenever those thresholds are crossed. This ensures consistent and hands-free regulation of the environment without requiring any manual intervention.

- **IoT and Remote Interface:** The system's IoT functionality is managed by an ESP8266 Wi-Fi module integrated with Arduino Mega. It provides a real-time data feed to a custom dashboard built on the Arduino IOT Cloud platform. This dashboard serves as the primary user interface for both data visualization and manual system control.

3.1.2 Literature Review

Recent developments in microcontrollers and IoT technologies have significantly improved how we approach agricultural storage, especially in managing post-harvest losses. By combining embedded systems with low-cost sensors and wireless connectivity, researchers have shown that it’s possible to automate storage monitoring, reduce manual labor, and minimize spoilage.

For example, Mehta and Patel [4] introduced a GSM-based grain monitoring system using Arduino. Their setup allowed real-time tracking of environmental conditions in storage units, cutting down on manual checks while ensuring stable, continuous data collection. The system proved effective in improving storage outcomes due to its simplicity and reliable data transmission.

Another study highlighted the use of temperature, humidity, and gas sensors—particularly for detecting ammonia—as key indicators of food grain quality. Sensor data was wirelessly sent to a central platform, allowing storage managers to receive timely alerts and act quickly when conditions deviated from safe limits. This showed how combining multiple environmental parameters could set a new benchmark for food safety and quality control.

Building on these ideas, our project—the Smart Fruit Storage System—integrates real-time monitoring of temperature, humidity, and gas levels with IoT-based communication using ESP8266 (with Arduino Mega) and the Arduino IOT Cloud platform. By automating control systems like misting, cooling, and ventilation, we aim to provide a more efficient and low-maintenance solution for preserving fruit quality during storage.

3.1.3 Formulation of Problem

Problem: In many households and small-scale markets, there is no effective or automated method to ensure that fruits are stored under proper environmental conditions. Storage environments often experience uncontrolled fluctuations in temperature, humidity, and gas concentration, leading to early spoilage and food waste. Manual monitoring methods are not only time-consuming and labor-intensive, but also inconsistent and prone to human error. They lack scalability and real-time responsiveness, making them unsuitable for continuous monitoring or timely intervention, especially in regions without advanced infrastructure.

Scope: This project focuses on developing a real-time monitoring system that continuously tracks environmental parameters crucial to fruit storage—specifically temperature, relative humidity, and the presence of spoilage-related gases. The system utilizes a set of integrated sensors and threshold-based automation to take corrective actions, such as cooling, misting, or venting, thereby maintaining ideal storage conditions. The solution is designed to be compact, low-cost, and accessible for everyday users in both rural and urban settings.

Goal: The primary goal of the project is to prevent premature spoilage of stored fruits by maintaining a stable and controlled environment inside the storage container. By leveraging IoT technologies and basic automation, the system aims to extend the shelf life of perishable items, improve food safety, and reduce post-harvest loss. This aligns with the broader objective of promoting food sustainability and reducing waste through affordable and scalable technological solutions.

3.1.4 Analysis

The Smart Fruit Storage System has been developed to actively manage and maintain the critical environmental factors that affect the longevity and freshness of stored fruits. By leveraging sensor integration, wireless connectivity, and automated control, the system provides a practical solution for reducing post-harvest waste.

Key Environmental Parameters Monitored:

- **Temperature:** Essential for slowing biochemical and microbial activity in fruits. The system ensures temperatures remain below 15°C using a Peltier cooling setup and fan system.
- **Humidity:** Excess moisture encourages mold and spoilage. The system maintains humidity between 80–90%, with a misting module that activates or deactivates based on real-time humidity readings.
- **Gas Concentration:** The buildup of gases like ammonia and CO_2 often signals early spoilage. An MQ135 sensor detects these gases, triggering exhaust fans, servo-operated vents, and an alert system when gas levels rise above safe limits.

Core System Features:

1. **Precision Sensing:** Environmental data is collected using DHT22 and MQ135 sensors, enabling responsive and accurate system behavior.
2. **Dependable Actuation:** Components like fans, coolers, misting units, and servos are controlled through relays that react to changing environmental conditions.
3. **Autonomous Operation with User Control:** The system is programmed to run independently based on set thresholds, but users can intervene remotely using a mobile interface powered by Arduino Mega with ESP8266 and Arduino IOT Cloud.
4. **Real-Time Data Communication:** Real-time updates are sent to a cloud dashboard, giving users continuous access to sensor data and control functions.

Expected Outcomes:

1. **Enhanced Fruit Quality and Shelf-Life**: By maintaining ideal storage conditions, the system minimizes waste and extends freshness.
2. **Remote Monitoring and Control**: Users can check and manage storage conditions from any location using a smartphone or internet-connected device.
3. **Affordable and Scalable Solution**: Built with cost-effective and easily accessible components, the system is well-suited for small farms, vendors, and decentralized storage units.

3.2 Design Method (PO(a))

The design of the Smart Fruit Storage System combines core principles from electronics, sensor interfacing, embedded programming, and control systems to tackle a pressing agricultural problem: post-harvest fruit spoilage caused by inadequate storage conditions.

To ensure optimal preservation, the system continuously monitors three critical environmental parameters—temperature, humidity, and air quality. These are measured using two sensors: the DHT22 for temperature and relative humidity, and the MQ135 for detecting harmful gases such as ammonia, CO_2, and other volatile organic compounds released during fruit decay. These sensors provide either digital or analog outputs, which are read through the input pins of an Arduino Mega, forming the sensory backbone of the system.

At the heart of the system's functionality is a threshold-based control logic, carefully designed in accordance with commonly recommended fruit storage standards. Specific threshold values are hard-coded into the firmware to determine when each actuator should be triggered:

Temperature Threshold:

- If $T > 15°C$, the **Peltier cooling module** and its associated fan are activated to reduce the temperature.

Humidity Threshold:

- If H < 80%, the misting module turns ON to increase humidity.
- If H > 90%, the misting system is turned OFF to avoid excess moisture.

Gas Level Threshold:

- If MQ135 reading > 1000, the ventilation fan, inlet/outlet servo vents, and warning buzzer-light are triggered for air exchange and spoilage alerts.

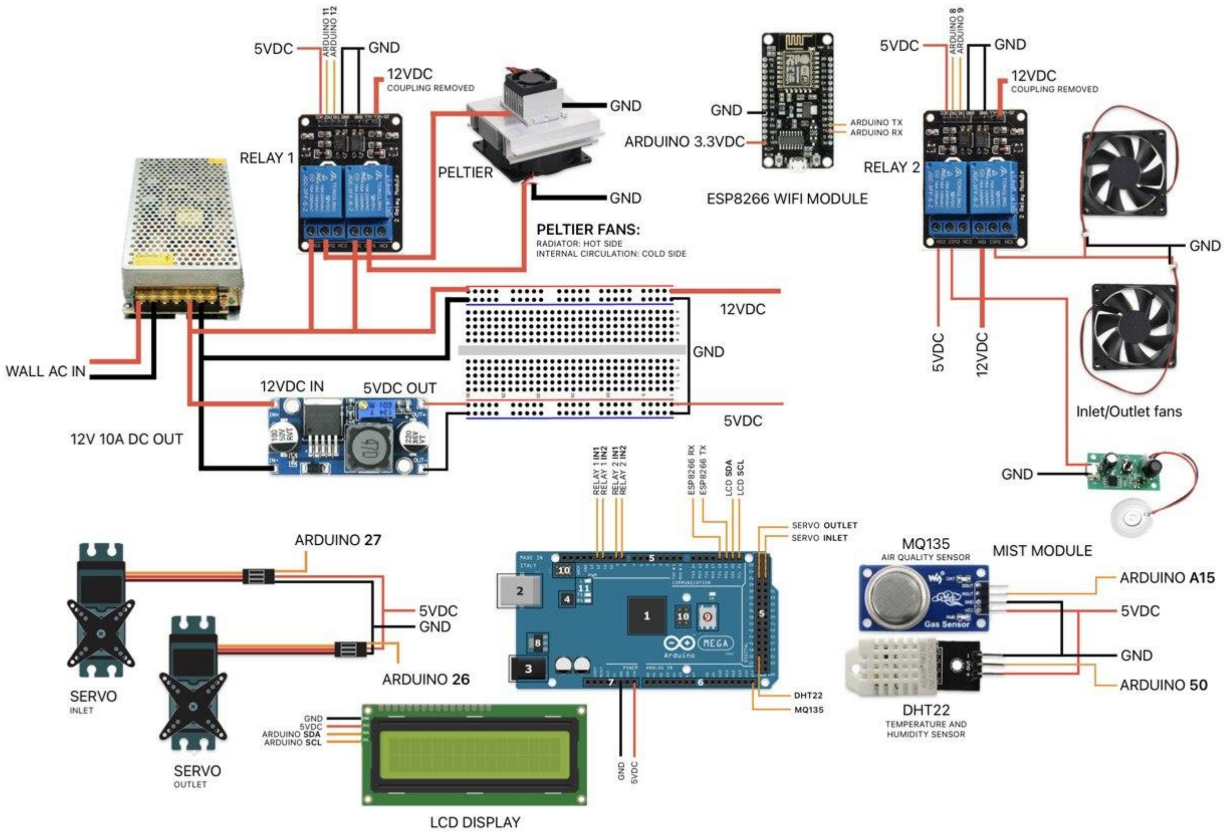
These simple yet effective control mechanisms are implemented using Boolean logic within the Arduino firmware, using *digitalWrite()* commands for actuator control. This allows real-time system response with minimal computational overhead.

To enable remote data access and visualization, the ESP8266 transmits live sensor readings to a Arduino IOT Cloud-powered IoT dashboard, allowing users to monitor the system from a mobile device in real time.

In terms of actuation, each mechanical component is connected through individual single-channel relay modules, all of which operate using active-LOW logic. These relays are driven by specific Arduino digital output pins and are responsible for controlling the Peltier module, fans, and misting unit.

By integrating sensor data acquisition, threshold-based decision-making, real-time communication, and automated control, the Smart Fruit Storage System effectively demonstrates the application of multidisciplinary engineering concepts to a real-world agricultural challenge. This meets the requirements of PO(a) by showcasing how theoretical knowledge is translated into a working embedded solution that minimizes waste and improves food storage practices.

3.3Circuit Diagram



3.4Full Source Code of Firmware

```
//Arduino
#include <Servo.h>
#include <DHT.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// =========================
// CONFIGURATION SECTION
// =========================

// --- Pin Assignments ---
const int dhtPin = 50;
const int airQualityPin = A15;
const int mistPin = 8;
const int fanPin = 9;
const int peltierPin = 11;
const int peltierFanPin = 12;
const int inletServoPin = 27;
const int outletServoPin = 26;
const int buzzerPin = 31;

// --- Threshold Values ---
const int airQualityThreshold = 1000;
const float tempThreshold = 15.0;
const float humidityThresholdHigh = 90.0;
const float humidityMistOn = 80.0;
const float humidityMistOff = 90.0;

// --- Servo Positions ---
const int inletClosedPos = 160;
const int outletClosedPos = 75;
const int inletOpenPos = 120;
const int outletOpenPos = 125;

// =========================
// COMPONENT INITIALIZATION
// =========================

#define DHTTYPE DHT22
DHT dht(dhtPin, DHTTYPE);
Servo inletServo;
Servo outletServo;
LiquidCrystal_I2C lcd(0x27, 16, 2);  // Change to 0x3F
if needed

bool ventsOpen = false;  // vent state

// --- Peltier Timer Variables ---
bool peltierCycling = false;
bool peltierState = false; // false = OFF, true = ON
unsigned long peltierTimer = 0;
const unsigned long peltierInterval = 5UL * 60UL *
1000UL; // 5 minutes in milliseconds

// =========================
// SETUP
// =========================

void setup() {
  Serial.begin(9600);
  Serial1.begin(9600); // UART for ESP8266
communication

  pinMode(mistPin, OUTPUT);
  pinMode(fanPin, OUTPUT);
  pinMode(peltierPin, OUTPUT);
  pinMode(peltierFanPin, OUTPUT);
  pinMode(buzzerPin, OUTPUT);

  dht.begin();
  lcd.begin(16, 2);
```

```
// ESP32
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>
#include <Servo.h>
#include "thingProperties.h"

// --- Pin Mapping ---
#define DHTPIN 18
#define DHTTYPE DHT22
#define AQ_PIN 34
#define MIST_PIN 19
#define FAN_PIN 23
#define PELTIER_PIN 25
#define PELTIER_FAN_PIN 26
#define INLET_SERVO_PIN 13
#define OUTLET_SERVO_PIN 12
#define BUZZER_PIN 27

// --- Servo positions ---
const int inletClosed = 160;
const int outletClosed = 75;
const int inletOpen = 120;
const int outletOpen = 125;

// --- Peltier cycle variables ---
bool peltierCycling = false;
bool peltierState = false;
unsigned long peltierTimer = 0;
const unsigned long peltierInterval = 5UL * 60UL *
1000UL; // 5 minutes

// --- States ---
bool ventsOpen = false;

// --- Components ---
DHT dht(DHTPIN, DHTTYPE);
Servo inletServo;
Servo outletServo;
LiquidCrystal_I2C lcd(0x27, 16, 2);  // LCD I2C Address

void setup() {
  Serial.begin(115200);
  delay(1500);

  // Initialize Arduino Cloud
  initProperties();
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();

  // Pin setup
  pinMode(MIST_PIN, OUTPUT);
  pinMode(FAN_PIN, OUTPUT);
  pinMode(PELTIER_PIN, OUTPUT);
  pinMode(PELTIER_FAN_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);

  // Attach servos
  inletServo.attach(INLET_SERVO_PIN);
  outletServo.attach(OUTLET_SERVO_PIN);
  inletServo.write(inletClosed);
  outletServo.write(outletClosed);

  // LCD
  lcd.begin(16, 2);
  lcd.backlight();

  // DHT
  dht.begin();
  lcd.backlight();

  inletServo.attach(inletServoPin);
  outletServo.attach(outletServoPin);

  // Initial States
  digitalWrite(mistPin, HIGH);         // Mist
initially ON (HIGH = OFF)
  digitalWrite(fanPin, HIGH);          // Fan OFF
  digitalWrite(peltierPin, HIGH);      // Peltier OFF
  digitalWrite(peltierFanPin, HIGH);   // Peltier fan
OFF
  digitalWrite(buzzerPin, HIGH);       // Buzzer OFF
  inletServo.write(inletClosedPos);
  outletServo.write(outletClosedPos);
}

// =========================
// MAIN LOOP
// =========================

void loop() {
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();
  int airQuality = analogRead(airQualityPin);

  // --- LCD Display ---
  lcd.setCursor(0, 0);
  lcd.print("T:");
  lcd.print(temperature);
  lcd.print(" H:");
  lcd.print(humidity);

  lcd.setCursor(0, 1);
  lcd.print("AQ:");
  lcd.print(airQuality);
  lcd.print("       "); // Clear residual chars

  // --- Mist Logic ---
  if (humidity < humidityMistOn) {
    digitalWrite(mistPin, LOW);  // Turn ON
  } else if (humidity >= humidityMistOff) {
    digitalWrite(mistPin, HIGH); // Turn OFF
  }

  // --- Vent + Fan + Buzzer Logic ---
  if (airQuality > airQualityThreshold) {
    if (!ventsOpen) {
      inletServo.write(inletOpenPos);
      outletServo.write(outletOpenPos);
      delay(500);
      ventsOpen = true;
    }
    digitalWrite(fanPin, LOW);       // Fan ON
    digitalWrite(buzzerPin, LOW);    // Buzzer ON
  } else {
    digitalWrite(fanPin, HIGH);      // Fan OFF
    digitalWrite(buzzerPin, HIGH);   // Buzzer OFF
    if (ventsOpen) {
      inletServo.write(inletClosedPos);
      outletServo.write(outletClosedPos);
      ventsOpen = false;
    }
  }

  // --- Peltier Logic with 5-min ON/OFF cycle ---
  bool peltierCondition = (temperature > tempThreshold
|| humidity > humidityThresholdHigh);

  if (peltierCondition) {
    if (!peltierCycling) {
      // Start cycle
      peltierCycling = true;
      peltierState = true;
      peltierTimer = millis();
      digitalWrite(peltierPin, LOW);
      digitalWrite(peltierFanPin, LOW);
    } else {
      // Continue cycling
      if (millis() - peltierTimer >= peltierInterval) {
        peltierState = !peltierState; // Toggle state
        peltierTimer = millis();      // Reset timer
        if (peltierState) {
          digitalWrite(peltierPin, LOW);      // ON
```

```
  // Initial OFF states
  digitalWrite(MIST_PIN, HIGH);
  digitalWrite(FAN_PIN, HIGH);
  digitalWrite(PELTIER_PIN, HIGH);
  digitalWrite(PELTIER_FAN_PIN, HIGH);
  digitalWrite(BUZZER_PIN, HIGH);
}

void loop() {
  ArduinoCloud.update();

  // Read sensors
  temperature = dht.readTemperature();
  humidity = dht.readHumidity();
  airQuality = analogRead(AQ_PIN);

  // Update LCD
  lcd.setCursor(0, 0);
  lcd.print("T:");
  lcd.print(temperature);
  lcd.print(" H:");
  lcd.print(humidity);

  lcd.setCursor(0, 1);
  lcd.print("AQ:");
  lcd.print(airQuality);
  lcd.print("       ");

  // --- Mist Logic ---
  if (humidity < 80) {
    digitalWrite(MIST_PIN, LOW);
    mistOn = true;
  } else if (humidity >= 90) {
    digitalWrite(MIST_PIN, HIGH);
    mistOn = false;
  }

  // --- Vent, Fan, Buzzer Logic ---
  if (airQuality > 1000) {
    if (!ventsOpen) {
      inletServo.write(inletOpen);
      outletServo.write(outletOpen);
      delay(500);
      ventsOpen = true;
    }
    digitalWrite(FAN_PIN, LOW);
    digitalWrite(BUZZER_PIN, LOW);
    fanRunning = true;
    alarmOn = true;
  } else {
    digitalWrite(FAN_PIN, HIGH);
    digitalWrite(BUZZER_PIN, HIGH);
    fanRunning = false;
    alarmOn = false;
    if (ventsOpen) {
      inletServo.write(inletClosed);
      outletServo.write(outletClosed);
      ventsOpen = false;
    }
  }

  // --- Peltier 5-min Cycle Logic ---
  bool peltierCondition = (temperature > 15.0 ||
humidity > 90.0);
  if (peltierCondition) {
    if (!peltierCycling) {
      peltierCycling = true;
      peltierState = true;
      peltierTimer = millis();
      digitalWrite(PELTIER_PIN, LOW);
      digitalWrite(PELTIER_FAN_PIN, LOW);
      peltierRunning = true;
    } else {
      if (millis() - peltierTimer >= peltierInterval) {
        peltierState = !peltierState;
        peltierTimer = millis();
        if (peltierState) {
          digitalWrite(PELTIER_PIN, LOW);
          digitalWrite(PELTIER_FAN_PIN, LOW);
          peltierRunning = true;
        } else {
          digitalWrite(PELTIER_PIN, HIGH);
          digitalWrite(PELTIER_FAN_PIN, HIGH);
          peltierRunning = false;
          digitalWrite(peltierFanPin, LOW);   // ON
        } else {
          digitalWrite(peltierPin, HIGH);     // OFF
          digitalWrite(peltierFanPin, HIGH);  // OFF
        }
      }
    }
  } else {
    // Stop cycle
    peltierCycling = false;
    peltierState = false;
    digitalWrite(peltierPin, HIGH);
    digitalWrite(peltierFanPin, HIGH);
  }

  // =========================
  // ESP8266 Data Transmission
  // =========================
  bool fanState = (digitalRead(fanPin) == LOW);
// Fan ON = LOW
  bool peltierActive = (digitalRead(peltierPin) ==
LOW);  // Peltier ON = LOW

  Serial1.print("DATA,");
  Serial1.print(temperature);
  Serial1.print(",");
  Serial1.print(humidity);
  Serial1.print(",");
  Serial1.print(airQuality);
  Serial1.print(",");
  Serial1.print(peltierActive);
  Serial1.print(",");
  Serial1.println(fanState);

  // =========================
  // ESP8266 Command Reception
  // =========================
  if (Serial1.available()) {
    String cmd = Serial1.readStringUntil('\n');
    if (cmd.startsWith("CMD")) {
      int peltierCmd = cmd.charAt(4) - '0';  // CMD,P,F
      int fanCmd = cmd.charAt(6) - '0';

      digitalWrite(peltierPin, peltierCmd ? LOW :
HIGH);
      digitalWrite(peltierFanPin, peltierCmd ? LOW :
HIGH);
      digitalWrite(fanPin, fanCmd ? LOW : HIGH);
    }
  }

  delay(2000); // 2 seconds loop delay
}
```

```
        }
      }
    }
  } else {
    peltierCycling = false;
    peltierState = false;
    digitalWrite(PELTIER_PIN, HIGH);
    digitalWrite(PELTIER_FAN_PIN, HIGH);
    peltierRunning = false;
  }

  delay(2000); // Slow down loop
}
```

Table: Source Code for the main program

4 Implementation

The Smart Fruit Storage System is designed to maintain ideal conditions for keeping fruits fresh and extending their shelf life. It continuously monitors temperature and humidity using a microcontroller and automatically adjusts devices like fans and cooling modules to preserve quality in real-time.

The system uses a modular design where an Arduino manages sensor data and actuator controls, while an ESP32 provides the same functionality with added wireless connectivity. This allows for remote monitoring and manual control through the internet, combining reliable local operation with convenient cloud access to ensure fruits stay fresh longer.

4.1 Description

The system acquires temperature, humidity, and gas concentration data using a combination of digital and analog sensors. These sensor readings are processed by a microcontroller to regulate actuators such as fans, mist modules, and Peltier-based cooling units, ensuring that environmental conditions remain within desired thresholds. While the Arduino Mega serves as the main controller for local processing and actuator control, the ESP8266 handles IoT functionality. Alternatively, a single ESP32 can be used to perform both sensing and control operations, with the added advantage of built-in Wi-Fi for seamless cloud integration and real-time remote monitoring via platforms like Arduino IOT Cloud.

Components Used:

1) Arduino Mega (Central controller for sensor reading, actuator control, and relay logic)
2) ESP8266 (IoT interface and remote-control unit)
3) DHT22 (Measures temperature and humidity)
4) MQ135 (Measures air quality)
5) Relay Module 1 (2-channel) – Channel 1 controls Peltier module, Channel 2 controls Peltier fans
6) Relay Module 2 (2-channel) – Channel 1 controls inlet/outlet fan, Channel 2 controls mist module
7) Arduino IOT Cloud (Remote monitoring and control interface via ESP8266)

In this project, two microcontrollers are used: the Arduino Mega serves as the central controller for sensor data acquisition and actuator control, while the ESP8266 provides the IoT connectivity to the Arduino IOT Cloud platform.

The DHT22 sensor, used to monitor temperature and humidity, is connected to digital pin D50 of the Arduino Mega. It is powered through the Arduino's 5V and GND pins and provides digital output directly to the microcontroller. The MQ135 gas sensor, which measures air quality by detecting gases like CO_2, NH_3, and others, is connected to analog pin A15 on the Arduino. It is similarly powered using the 5V and GND rails.

For environmental control, two 2-channel relay modules are used. Each relay channel provides electrical isolation and safe switching of external high-power devices:

Relay Module 1:

Channel 1 (D11) controls the Peltier module, which is used for active cooling.

Channel 2 (D12) controls the fans attached to the Peltier for heat dissipation.

Relay Module 2:

Channel 1 (D9) switches the inlet/outlet fan, responsible for air exchange.

Channel 2 (D8) controls the mist module, which adds humidity to the environment when it drops below a certain threshold.

Two servo motors are also employed to open and close inlet and outlet vents for airflow, connected to pins D27 and D26, respectively. A buzzer is connected to pin D31 and provides an audible alarm when poor air quality is detected.

A bidirectional UART interface (Serial1) is used for communication between the Arduino Mega and ESP8266. Real-time sensor data (temperature, humidity, air quality) is transmitted from Arduino pin D18 (TX1) to ESP8266 RX using a voltage divider circuit to step down 5V to 3.3V, ensuring safe communication. The ESP8266 sends control commands from its TX pin to Arduino pin D19 (RX1), enabling remote override functions via the Arduino IOT Cloud app.

The system is powered using a 12V DC adapter, which is sufficient to drive the relay modules, Peltier system, fans, and mist module. Internal logic components (Arduino and ESP8266) are powered through regulated 5V and 3.3V supplies as needed.

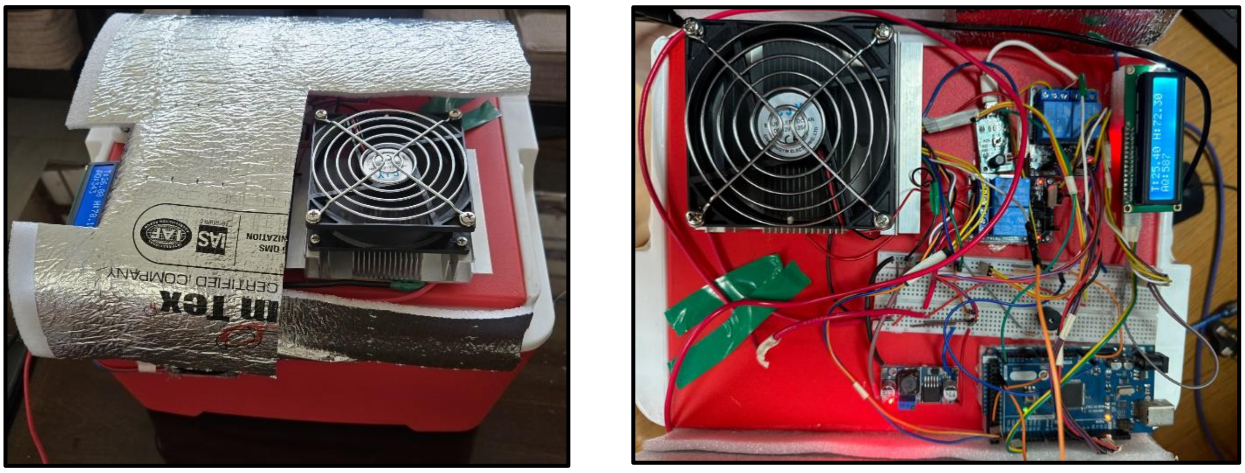
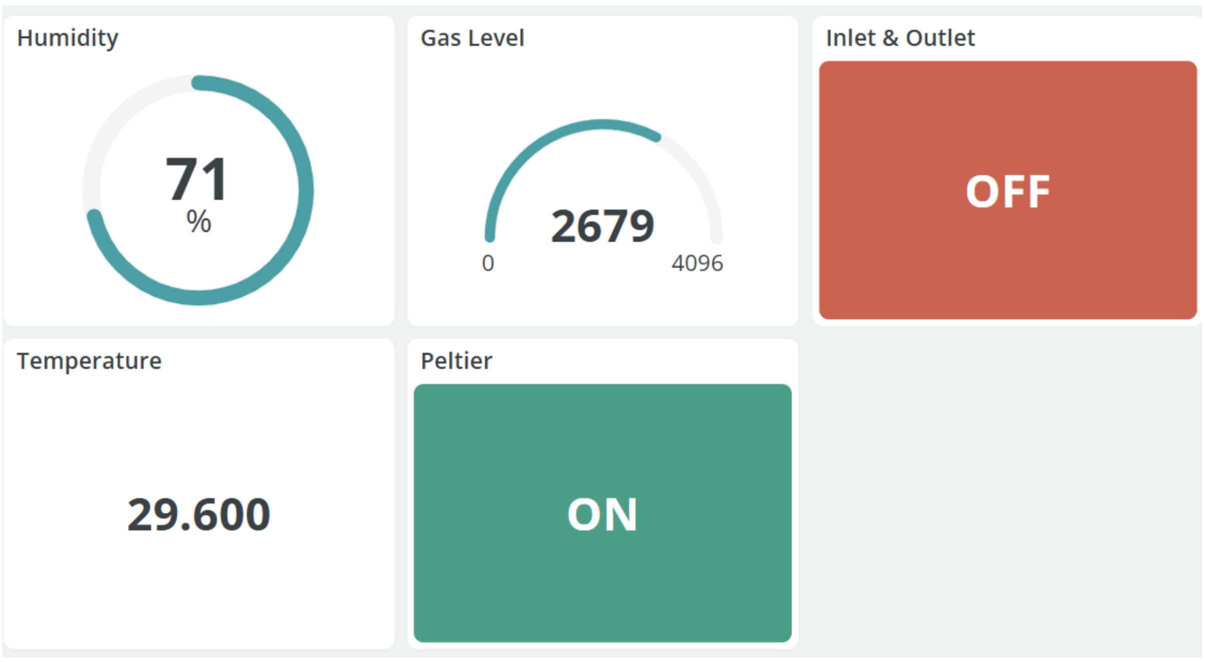


Fig: Final Hardware Setup



Humidity
71
%

Gas Level
2679
0
4096

Inlet & Outlet
OFF

Temperature
29.600

Peltier
ON

Fig: Arduino IOT Cloud Dashboard

5 Design Analysis and Evaluation

5.1 Novelty

• Integrated Multi-Functionality in a Single Unit
The system uniquely combines cooling, humidifying, and gas detection features within one compact box. Unlike traditional systems that focus on only one environmental factor, this integrated approach allows for comprehensive control of fruit storage conditions using a unified setup.

• Threshold-Based Automated Control
The system automatically activates misting, cooling, or venting based on real-time sensor readings. This threshold-triggered automation eliminates the need for manual intervention, ensuring timely and consistent environmental regulation to preserve fruit quality.

• Dual Role of Peltier Module
The Peltier module is not only used for cooling but also assists in dehumidification. As the cold side of the module condenses moisture from the air, it effectively reduces humidity levels while simultaneously lowering the internal temperature, offering two benefits from a single component.

• Gas-Triggered Ventilation System
Spoilage-related gases are detected by the MQ135 sensor, and if levels rise above a set threshold, a servo motor opens the ventilation system. This targeted approach helps in removing accumulated gases and preventing further degradation of stored fruits.

• Low-Cost and Compact for Household Use
The design prioritizes affordability and accessibility, using locally available components and basic electronic modules. Its compact form makes it suitable for everyday home users and small vendors, enabling better fruit preservation without relying on large, expensive cold storage setups.

5.2 Design Considerations (PO(c))

5.2.1 Considerations to public health and safety

The system is designed to prevent the storage and consumption of spoiled fruits by maintaining safe environmental conditions inside the storage unit. By automatically responding to changes in temperature, humidity, and air quality, it reduces the risk of microbial growth and toxin buildup, thereby helping ensure that fruits remain safe for consumption. Electrical safety is also addressed through proper insulation and current isolation of high-power components.

5.2.2 Consideration to environment

The solution promotes environmental sustainability by reducing food waste through early detection and prevention of spoilage. Components such as the Peltier module and mist system are used efficiently, and the design minimizes power consumption with threshold-triggered control instead of continuous operation. Moreover, the use of reusable containers and low-voltage electronics aligns with eco-friendly practices.

5.2.3 Considerations to cultural and societal needs

In many local communities, especially in Bangladesh, people store seasonal fruits in bulk at home or in small shops. The system addresses this societal need by offering an affordable, scalable, and easy-to-use solution for safe fruit storage. It helps reduce post-harvest losses and supports food security, aligning with both cultural practices and economic constraints in the local context.

5.3Investigations (PO(d))

This project involved investigating the effects of environmental parameters—temperature, humidity, and air quality—on storage conditions and determining the appropriate control mechanisms to maintain optimal levels. Various test scenarios were created to simulate abnormal environmental conditions, such as elevated humidity or poor air quality, to observe system responses. Experimental data were collected from the DHT22 and MQ135 sensors and analyzed to refine control thresholds for misting, ventilation, and Peltier activation.

Further investigation included determining the timing cycle for Peltier module operation and evaluating sensor reliability under changing environmental conditions. The relay response time, servo operation, and buzzer alert logic were also validated through repeated testing. These investigations supported the iterative refinement of control logic and ensured reliable and responsive system behavior under various real-world conditions.

5.3.1 Design of Experiment

To validate the functionality and performance of the prototype, a series of experiments were designed. The primary experimental setup consisted of placing the complete sensor and actuator system inside a sealed Styrofoam ice-box to simulate a small-scale storage environment.

- **Experiment 1: Temperature Control Verification.**
  - **Objective:** To verify that the Peltier cooling unit activates and deactivates at the correct temperature thresholds.
  - **Procedure:** A small heat source was placed inside the box to slowly raise the temperature. The system was monitored to see if the relay for the Peltier unit turned ON when the temperature reported by the DHT22 exceeded 30°C, and turned OFF when the temperature subsequently fell below 28°C.

- **Experiment 2: Humidity Control Verification.**
  - **Objective:** To verify the functionality of the LED light based on humidity levels.
  - **Procedure:** A small container of water was placed in the box to increase humidity. The system was observed to see if the LED remained OFF. Then, a desiccant (silica gel) was introduced to lower the humidity. The experiment aimed to confirm that the LED turned ON when humidity dropped below 60% and turned OFF again once it rose above 40%.

- **Experiment 3: Air Quality Control Verification.**
  - **Objective:** To test the activation of the ventilation fan.
  - **Procedure:** A small amount of volatile substance (e.g., alcohol on a cotton swab) was introduced near the MQ135 sensor to simulate a drop in air quality. The system was

monitored to confirm that the ventilation fan turned ON when the sensor's analog reading surpassed the threshold of 200.

- **Experiment 4: IoT Dashboard and Manual Control Verification.**
  - **Objective:** To confirm the real-time data display and manual override functionality.
  - **Procedure:** Throughout all experiments, the Arduino IOT Cloud dashboard was monitored to ensure that the sensor values displayed on the app matched the values being printed on the Arduino's Serial Monitor. The manual override switch was then activated, and each of the three actuator switches was toggled to confirm that they correctly controlled their respective relays, irrespective of the sensor readings.

5.3.2 Data Collection

Data was collected through multiple channels during the experiments:

I. **Arduino Serial Monitor:** Provided a real-time log of raw sensor values and actuator states as seen by the primary controller.

II. **ESP8266 Serial Monitor:** Provided a log of the data received from the Arduino and the status of the Arduino IOT Cloud connection.

III. **Arduino IOT Cloud Application:** The Super Chart widget in Arduino IOT Cloud could be used to log and visualize the historical data for temperature, humidity, and gas levels.

IV. **Manual Observation:** Visual confirmation of when the relays' indicator LEDs turned on and off, and when the actuators (fans, LED) were physically active.

5.3.3 Results and Analysis

The experiments yielded positive results, confirming that the system operates as designed.

Experiment	**Condition**	**Expected Outcome**	**Observed Outcome**	**Analysis**
Temperature Control	Temp rises to 31.2°C	Peltier relay ON	Relay ON, Peltier unit active	**Success**. The system correctly responded to the high-temperature threshold.
Temperature Control	Temp falls to 27.5°C	Peltier relay OFF	Relay OFF, Peltier unit inactive	**Success**. The hysteresis in the logic prevented rapid switching.
Humidity Control	Humidity falls to 58.9%	LED relay ON	Relay ON, LED lit	**Success**. The system correctly identified the low-humidity condition.
Gas Control	Gas reading rises to 255	Fan relay ON	Relay ON, Fan spinning	**Success**. The ventilation system activated as intended.

In addition to verifying the autonomous control logic, a comprehensive test of the 'Manual Mode' was conducted. This feature is crucial for providing the user with the ability to override the automated system for specific scenarios such as system calibration, maintenance, or responding to unique environmental conditions. The experiment involved activating the 'Manual Control' switch on the Arduino IOT Cloud dashboard, which successfully disengaged automatic logic. Subsequently, each actuator was individually toggled using its respective switch on the mobile app. The results were successful: each relay responded instantaneously to the manual commands, allowing for direct control over the Peltier cooling unit, the humidity-control LED, and the ventilation fan, regardless of the current sensor readings. Importantly, the system continued to stream live temperature, humidity, and air quality data to the Arduino IOT Cloud dashboard throughout the manual operation, ensuring the user maintained complete situational awareness.

5.3.4 Interpretation and Conclusions on Data

The collected data and experimental results lead to the following conclusions:

1. The dual-microcontroller architecture is effective. The Arduino successfully handles real-time control while the ESP8266 reliably manages IoT communication.
2. The sensor integration is successful, with both the DHT22 and MQ135 providing consistent data to the control system.
3. The actuator control logic is implemented correctly. The system can autonomously maintain the environment within the specified temperature, humidity, and air quality ranges.
4. The IoT functionality is fully operational. The system provides accurate remote monitoring and gives the user effective manual control over the storage environment.

In conclusion, the investigation validates that the prototype is a functional and effective solution to the problem formulated.

5.4 Limitations of Tools (PO(e))

- **MQ135 sensor :** The MQ135 gas sensor is sensitive to a wide range of gases, but its readings can vary significantly without proper calibration. The sensor outputs an analog voltage based on gas concentration, but this value depends on environmental factors like temperature and humidity. For precise detection (e.g., spoilage gases), baseline calibration using clean air and reference values is essential. Without this, the system may generate false positives or fail to detect gradual gas buildup.
- **DHT22 Sensor:** While DHT22 is a reliable sensor for basic temperature and humidity measurement, it has a relatively slow sampling rate and limited precision during rapid environmental changes. This means that sudden changes in humidity or temperature may not be captured in real time, potentially delaying system responses such as activating the mist or cooling module.
- **Peltier Module:** The Peltier module is effective for compact cooling but consumes high current (often 6A), which demands a strong power supply. Additionally, it takes time to create a thermal gradient between the hot and cold sides, making the system slower to react to sudden temperature changes. This can reduce energy efficiency and delay environmental correction during critical spoilage windows.
- **Relay**: Relays are excellent for simple ON/OFF control of high-current devices, but they cannot offer variable speed or fine-tuned actuation. For example, you cannot control fan speed or mist output intensity with a relay — only turn them fully on or off. This limits the system’s ability to apply gradual environmental adjustments, which may be important for precise humidity control.
- **Mist Module**: The mist module delivers moisture without feedback-based regulation. If the environment is already near the upper humidity threshold, even a short mist cycle may push it too far, creating excessive condensation. This could promote mold or moisture-related spoilage if not managed carefully with timing or better sensing.

- **Wiring**: The use of breadboards and jumper wires during prototyping exposes the system to signal interference, especially in analog sensors like MQ135. Long or loose wires can pick up ambient electrical noise, leading to unstable readings. Moreover, open wiring increases the risk of accidental disconnection, particularly in practical deployment or testing conditions.

5.5 Impact Assessment (PO(f))

5.5.1 Assessment of Societal and Cultural Issues

The system promotes responsible fruit storage at the household and small-vendor level, where access to refrigeration or large-scale preservation methods is limited. By reducing spoilage, it contributes to food security and helps minimize economic loss in low-resource communities. Culturally, it supports the habit of buying and storing seasonal fruits in bulk, which is common in many local markets.

5.5.2 Assessment of Health and Safety Issues

Spoiled fruits can emit gases like ammonia and CO_2, which may cause unpleasant odors or health risks in confined storage spaces. By actively monitoring air quality and controlling temperature and humidity, the system helps reduce these risks. This improves food hygiene and ensures that fruits remain safe for consumption over extended periods.

5.5.3 Assessment of Legal Issues

The system is a low-voltage, enclosed device using commonly available components. It does not involve radio transmission or hazardous materials. Therefore, it complies with general electrical and safety standards. If expanded to commercial production, basic electrical and food safety certifications would suffice, and no regulatory conflict is expected.

5.6 Sustainability Evaluation (PO(g))

1. **Encourages sustainable fruit storage practices in local households and markets** The system promotes sustainability by enabling small-scale users—such as households and local vendors—to preserve fruit longer without relying on large refrigeration systems. By using low-cost components and simple control logic, the system is accessible and replicable, making it a practical solution for promoting food sustainability in developing regions.
2. **Improves long-term food safety by preventing early spoilage and contamination** By maintaining a controlled environment, the system helps prevent the buildup of harmful gases and excess moisture that often led to spoilage. This reduces the likelihood of fungal or bacterial growth, thereby extending shelf life and enhancing food safety over longer periods without the need for chemical preservatives.

3. **System design supports sustainability goals without conflicting with any existing safety or electrical standards**
   The system operates within safe electrical limits and uses energy-efficient modules like Peltier coolers and relay switching. It does not introduce any hazardous waste or interfere with standard electrical practices. Thus, it aligns with sustainability goals such as reduced energy consumption, safer food systems, and minimal environmental impact.

5.7 Ethical Issues (PO(h))

- **Electrical Safety**
  We prioritized user and device safety during the design phase by implementing proper insulation and ensuring all high-current components were electrically isolated. This was particularly important for modules like the Peltier cooler and 12V fans, which draw significant current. Relay modules were used to safely switch these components without exposing low-voltage circuits, minimizing the risk of electrical hazards or accidental short circuits.
- **Ethical Component Sourcing**
  Component selection was done with care to ensure that only reliable and ethically sourced parts were used. We avoided counterfeit, unverified, or low-grade modules which could compromise performance or safety. Components such as the ESP8266, DHT22, and MQ135 were chosen not only for their functionality but also for their availability from reputable local vendors, ensuring transparency and traceability in sourcing.
- **Data Privacy Consideration**
  At its current stage, the system operates offline without collecting or transmitting any user-specific data. However, we anticipate that future upgrades may include cloud-based monitoring. In such cases, we acknowledge the importance of ethical data practices. This would involve implementing data security measures such as encryption, anonymized logging, and informed user consent to protect privacy and maintain trust in the system's use.

6 Reflection on Individual and Team work (PO(i))

6.1 Individual Contribution of Each Member

Team Member	Responsibility
87,88	Sensor selection & calibration
88, 89	Testing & validation; documentation
89, 90	Cloud setup & dashboard development
87, 90	Microcontroller coding & integration

6.2 Mode of TeamWork

Our team followed a collaborative and task-based working mode. Responsibilities were divided based on individual strengths, while key decisions and integration steps were discussed collectively. Regular check-ins and shared documentation ensured smooth progress and accountability throughout the project.

6.3 Diversity Statement of Team

Our project team brought together members with diverse strengths—ranging from hardware prototyping and circuit design to coding, testing, and documentation. This diversity allowed us to divide tasks effectively, share different perspectives during decision-making, and ensure each part of the system—from sensor integration to final packaging—was handled with both technical depth and collaborative effort.

6.4 Log Book of Project Implementation

Date	**Activity**	**Key Decision / Observation**
Week 1	Formed project group and received initial instructions	Group members finalized; received guidance on project expectations, scope, and timeline from course instructor.
Week 2-3	Explored project topics and reviewed references	Explored project ideas by reviewing course content and relevant papers; finalized the topic after team discussion.
Week 4	Shortlisted project ideas for presentation	Selected 3 potential projects and planned to present them in the following week for feedback and final decision.
Week 5	Finalized project topic and goals	Was instructed to focus on smart fruit storage using sensors
Week 6	Component selection and sourcing, Sensor testing (DHT22, MQ135)	Chose ESP32, DHT22, MQ135, Peltier module, Verified basic readings; observed slight DHT delay
Week 7	Relay control setup	Decided on 2 relays: one for cooling, one for mist
Mid-break	Optional cloud integration and code refinement	Explored cloud-based UI options and optimized control logic for smoother sensor-actuator response.
Week 8	Peltier + fan integration	Cooling confirmed; cold side causes condensation

Week 9	Mist module and humidity threshold testing	Overshoot noted when humidity very low
Week 10	Vent + servo control based on gas levels	Servo works well; adjusted position for airflow
Week 11	Full system integration	All modules respond to thresholds as expected
Week12	Box setup and internal layout optimization	Airflow and wiring adjusted for better reliability
Week 13	Live test with stored fruit	System maintained stable temperature and humidity; vent triggered after mild gas rise; fruit stayed visibly fresh over 48 hours
Week 14	Final packaging and report preparation	Minor wiring cleanup needed; logbook and cost table done

7 Communication to External Stakeholders (PO(j))

7.1 Executive Summary

We developed an IoT-based smart fruit storage system that monitors and controls temperature, humidity, and air quality in real time. Using sensors and automated controls, the system activates misting, cooling, and ventilation when needed to prevent spoilage. This low-cost, compact solution displays live data and can be adapted for household or vendor use. By reducing waste and maintaining freshness, the project offers a practical innovation for safer and more efficient fruit storage across Bangladesh.

7.2 User Manual

The primary interface for monitoring and controlling the system is the Arduino IOT Cloud mobile dashboard. After connecting, the dashboard provides a continuous, real-time display of the critical environmental parameters: Temperature (°C), Relative Humidity (%), and a qualitative value for Air Quality.

The system operates in two distinct modes:

1. **Automatic Mode:** This is the default operational state. In this mode, the system autonomously manages the storage environment. It uses the live sensor data to automatically activate or deactivate the temperature control unit (Peltier and fan), the humidity control unit (LED light), and the air quality ventilation fan according to the predefined logic in the firmware.

2. **Manual Mode:** For direct intervention, the user can engage Manual Mode by toggling the "Manual Control" switch on the dashboard to the 'ON' position. Upon activation, the automatic control logic is disengaged, and all actuators are turned off by default. The operator can then

use the individual switches on the dashboard to independently control each component. To revert to autonomous operation, the "Manual Control" switch is toggled back to the 'OFF' position. Throughout manual operation, the dashboard continues to display live sensor data, ensuring the operator remains fully aware of the environmental conditions.

System Status and Troubleshooting Guide:

- **Offline Status:** If the Arduino IOT Cloud application indicates that the device is "Offline," it suggests a loss of connection between the ESP8266 controller and the cloud server. The most common causes are a loss of power to the controller unit or an interruption in the local Wi-Fi network service. Verifying that both the power supply and the Wi-Fi router are operational should resolve the issue.
- **Inaccurate Sensor Readings:** Should the sensor values display on the dashboard appear erroneous or inconsistent, it may point to an issue with the physical sensor unit. It is recommended to check that the sensor module is positioned correctly within the storage environment and that its sensing elements are free from dust or obstruction.

7.3 YouTube Link

https://youtu.be/SsQY30of0jE

7.4 GitHub Link

https://github.com/mohammad-al-hosan/IOT_Fruit_Storage_System.git

8 Project Management and Cost Analysis (PO(k))

8.1 Bill of Materials

Component	Qty	Unit Cost (BDT)	Total Cost (BDT)
12V 10A SMPS	1	700	700
USB micro B	1	60	60
CPU Cooler	1	400	400
funnel	1	30	30
insulating foil	1	100	100
Heat sink (large)	1	300	300
Heat sink (small)	1	100	100
Peltier Fan (large)	1	400	400

Peltier Fan (small)	1	150	150
Mounting Screw and Drilling	1	100	100
Ice Box 8L	1	920	920
Motor Driver	1	350	350
LCD	1	350	350
Magnets	1	150	150
DHT22	3	220	660
MQ-135	2	120	240
ESP32	2	400	800
Peltier Tec1-12706	1	200	200
Buck Converter	2	60	120
Mist Module	1	120	120
Relay Module	3	80	240
Cooling Fan (12V)	2	90	180
Alarm Buzzer Module	2	50	100
Servo motor	2	200	400
Arduino UNO	1	900	900
Arduino MEGA	1	1790	1790
Lipo charger(	1	300	300
Enclosure & Cabling	1	500	500
Total		**10,660**	

8.2 Calculation of Per Unit Cost of Prototype

Component	Qty	Unit Cost (BDT)	Total Cost (BDT)
12V 10A SMPS	1	700	700
funnel	1	30	30
Heat sink (large)	1	300	300
Heat sink (small)	1	100	100
Peltier Fan (large)	1	400	400
Peltier Fan (small)	1	150	150
Ice Box 8L	1	920	920
LCD	1	350	350
Magnets	1	150	150
Servo motor	1	200	200
Arduino MEGA	1	1790	1790
DHT22	1	220	220
MQ-135	1	120	120
Peltier Tec1-12706	1	200	200
Buck Converter	1	60	60
Mist Module	1	120	120
Relay Module	2	80	160

Cooling Fan (12V)	2	90	180
Alarm Buzzer Module	1	50	50
Enclosure & Cabling	1	300	300
Total			**6,500**

8.3 Calculation of Per Unit Cost of Mass-Produced Unit

Component	**Qty**	**Unit Cost (BDT)**	**Total Cost (BDT)**
12V 10A SMPS	100	650	65000
Funnel	100	15	1500
Heat sink (large)	100	300	30000
Heat sink (small)	100	100	10000
Peltier Fan (large)	100	350	35000
Peltier Fan (small)	100	120	12000
Ice Box 8L	100	750	75000
LCD	100	250	25000
Magnets	100	100	10000
Servo motor	100	180	18000
ESP32	100	380	38000
DHT22	100	190	19000

MQ-135	100	110	11000
Peltier Tec1-12706	100	180	18000
Buck Converter	100	50	5000
Mist Module	100	120	12000
Relay Module	200	70	14000
Cooling Fan (12V)	200	70	14000
Alarm Buzzer Module	100	40	4000
Enclosure & Cabling	100	180	18000
Total			**434,500**
Per Unit Cost			**4,345**

8.4 Timeline of Project Implementation

- Week 6: Finalize components and plan circuit design
- Week 7: Interface sensors and test basic hardware functions
- Mid-break: Optional work on cloud UI and code optimization
- Week 8-10: Perform testing, debugging, and edge case handling
- Week 11-12: Assemble final setup with enclosure and safety
- Week 12-13: Final testing and documentation
- Week 14: Project Presentation and demonstration

9 Future Work (PO(l))

- Create a mobile app for easier remote access.
- Use AI to predict spoilage early.
- Power the system with solar for remote areas.
- Expand system to monitor multiple storage sites.

10 References

1.Ansari et al., "IoT-Enabled Smart Storage System," Security and Privacy, 2023.
https://onlinelibrary.wiley.com/doi/10.1002/spy2.282

2.Hema et al., "IoT based real-time control for grain storage," IOP Conf. Series, 2020.

https://iopscience.iop.org/article/10.1088/1757-899X/993/1/012008

3.Gupta et al., IoT Based Cold Storage Monitoring System for Food Grains, Materials Today: Proceedings, 2021.

https://www.sciencedirect.com/science/article/pii/S2214785321011983

4.Mehta & Patel, An IoT Framework for Post-Harvest Crop Management, Journal of Agricultural Informatics, 2022.

https://journal.magisz.org/index.php/jai/article/view/144