CM 1900- Intelligent Machines Inspirational Project

PROJECT PROPOSAL REPORT

Level 01

Automatic Dehydrator

Examiner -Mr. B. H. Sudantha- ……………………

Submitted by:

L.Laavanjan	**225527H**
Subramaniam A	225544G
Prainila R	225535F
P.K.I Didulantha	225510B
K.M.T.N.B.Kulasekara	225524V

Faculty of Information Technology

University of Moratuwa

Table of Contents

1. Introduction

Food dehydration is a time-honored method for preserving various types of food by removing moisture, which inhibits the growth of microorganisms. This process not only extends the shelf life of food but also reduces its weight and volume, making it easier to store and transport. Historically, sun drying has been the most common method, but with advancements in technology, modern dehydration techniques have evolved to include electric dehydrators that offer more consistent and reliable results.

Despite these advancements, current food dehydration processes face several challenges. Traditional methods often rely on manual monitoring and control, leading to significant variability in temperature, airflow, and time management. This inconsistency can compromise the quality and safety of the dehydrated food. Additionally, manual methods are labor-intensive and require constant attention, which is not always practical for large-scale operations. These issues highlight the need for a more efficient and automated solution to ensure optimal drying conditions and consistent product quality.

The "Automatic Dehydrator" project aims to address these challenges by developing an intelligent, automated system that utilizes technology. By integrating advanced sensors like the DS18B20 temperature sensor, MQ-02 gas sensor, and DHT22 humidity sensor, the system actively monitors and controls the dehydration environment. The ESP32 microcontroller serves as the brain of the system, dynamically adjusting parameters based on real-time data to maintain ideal conditions. This project also includes a user-friendly web interface for selecting food types and configuring settings, making it accessible and easy to use.

One of the significant advantages of the "Automatic Dehydrator" is its ability to ensure consistent quality and safety in the dehydration process. The automation of temperature [1], humidity, and gas level monitoring reduces the risk of human error and enhances efficiency. Additionally, the system's advanced safety features, including emergency alerts through SMS, visual cues, and auditory signals, provide immediate responses to any deviations from optimal conditions. This not only improves the reliability of the dehydration process but also promotes energy efficiency and sustainability.

While developing the "Automatic Dehydrator," the team faced several challenges. Integrating various hardware components and ensuring seamless communication between them required meticulous planning and testing. The team had to overcome issues related to sensor calibration, real-time data processing, and maintaining stable WiFi connectivity for the web interface. Additionally, designing a user-friendly interface that meets the needs of diverse users posed its own set of challenges. Despite these hurdles, the project's iterative development process and rigorous testing ensured a robust and reliable final product.

In conclusion, the "Automatic Dehydrator" represents a significant advancement in food preservation technology. By automating the monitoring and control of dehydration parameters, this project addresses the limitations of traditional methods, ensuring consistent quality and safety. The integration of advanced sensors and a user-friendly interface makes it an efficient and accessible solution for both small-scale and large-scale food dehydration needs. Despite the challenges faced during development, the project's success demonstrates the potential of technology to revolutionize food preservation practices.

2. Literature Survey

The concept of food dehydration has been extensively studied and documented in the literature due to its importance in food preservation. Traditional dehydration methods, such as sun drying, have been used for centuries, but they are often inconsistent and reliant on weather conditions. Advances in technology have led to the development of electric food dehydrators, which provide more controlled and efficient drying environments. Studies have shown that electric dehydrators significantly improve the quality and safety of dehydrated foods by maintaining consistent temperatures and airflow, thereby reducing the risk of microbial growth and spoilage.

However, existing electric dehydrators still face challenges related to manual control and monitoring. Research indicates that many commercial dehydrators lack automated systems to adjust drying parameters in real-time, which can result in uneven drying and energy inefficiencies. Moreover, these systems often require significant user intervention to monitor and adjust settings, making them labor-intensive and less practical for large-scale operations. Studies have highlighted the need for more sophisticated automation and control mechanisms to enhance the efficiency and effectiveness of food dehydration processes.

Recent advancements in sensor technology and microcontroller systems offer promising solutions to these challenges. The integration of temperature, humidity, and gas sensors can provide real-time data to optimize the dehydration process. Literature on smart agriculture and IoT-based systems demonstrates the potential of using sensors and microcontrollers, such as the ESP32, to automate and improve various agricultural processes, including food dehydration. These systems can dynamically adjust parameters based on sensor inputs, ensuring optimal drying conditions and reducing the risk of human error.

The proposed "Automatic Dehydrator" project aims to leverage these technological advancements to address the limitations of current dehydration methods. By incorporating advanced sensors and microcontrollers, the system can continuously monitor and adjust drying conditions, providing a more reliable and efficient dehydration process. The literature supports the feasibility and potential benefits of such an automated system, including improved consistency, reduced labor requirements, and enhanced safety. This project builds on existing research by integrating these technologies into a comprehensive, user-friendly solution for food dehydration.

3. Aim and Objectives

Aim:

The primary aim of this project is to develop an "Automatic Dehydrator" using ESP32 technology, offering a smart and automated solution to the challenges posed by manual food dehydration methods

Objectives:

i. Design and implement a robust system that actively monitors and controls temperature, humidity, and gas levels throughout the food dehydration process, guaranteeing optimal conditions for preservation.

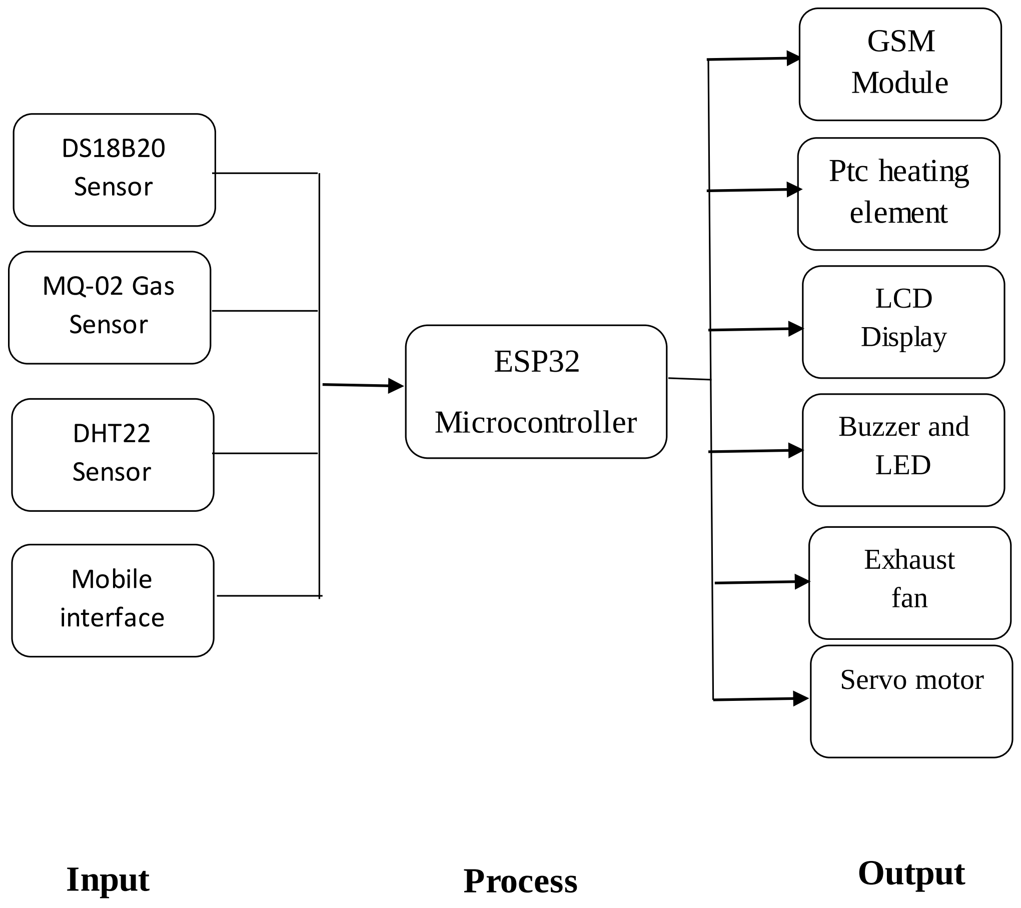
ii. Develop an intuitive web interface enabling users to effortlessly select their food type and automate the ideal drying settings, enhancing user accessibility and customization.

iii. Integrate advanced safety features, including emergency alerts through SMS, visual cues, and auditory signals, to respond effectively to critical situations during the dehydration process.

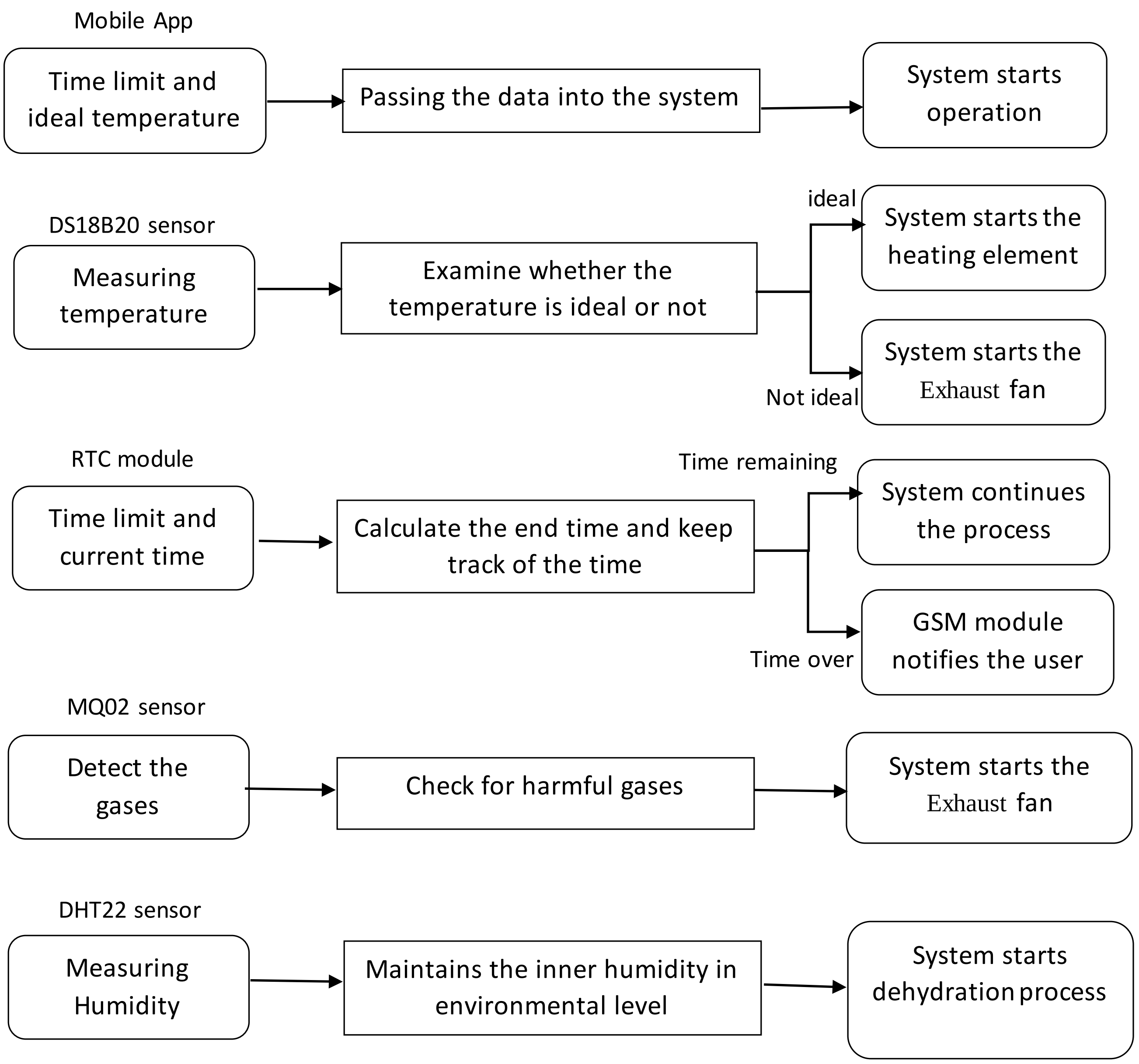
iv. Incorporate energy-efficient mechnisms, such as intelligent scheduling and power management, to promote sustainability and minimize environmental impact

4. Analysis and Design

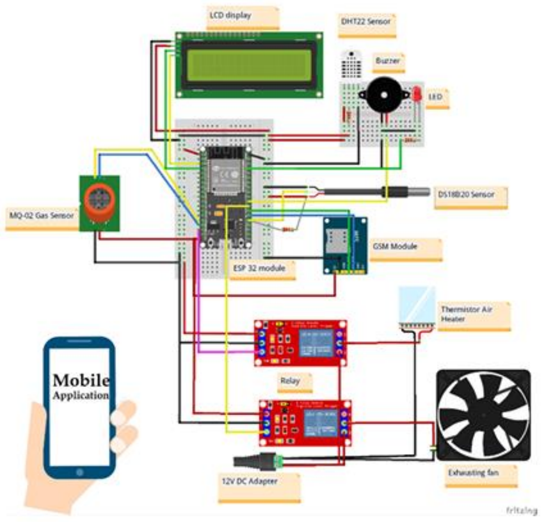
Block Diagram

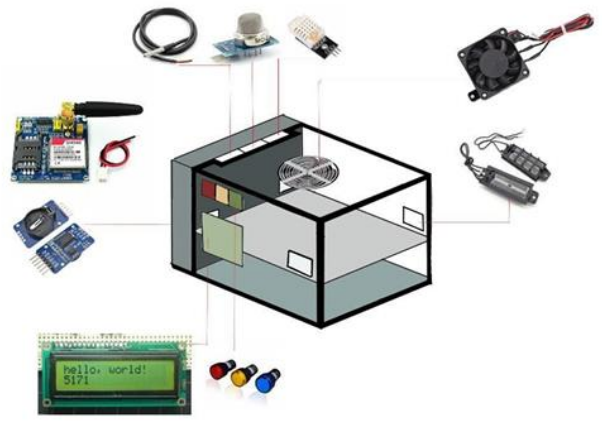


Functional Diagram

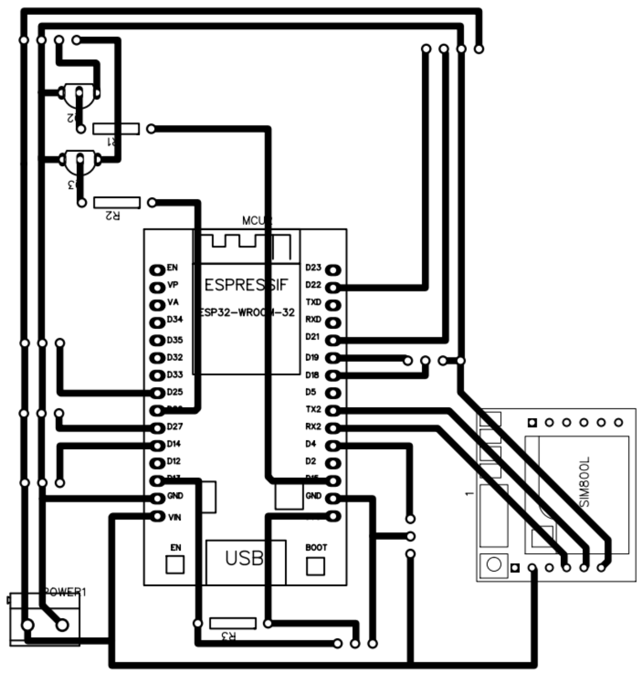

Mobile App
Time limit and ideal temperature
Passing the data into the system
System starts operation
DS18B20 sensor
Measuring temperature
Examine whether the temperature is ideal or not
ideal
System starts the heating element
Not ideal
System starts the Exhaust fan
RTC module
Time limit and current time
Calculate the end time and keep track of the time
Time remaining
System continues the process
Time over
GSM module notifies the user
MQ02 sensor
Detect the gases
Check for harmful gases
System starts the Exhaust fan
DHT22 sensor
Measuring Humidity
Maintains the inner humidity in environmental level
System starts dehydration process

Solution design





PCB Design



5. Testing and Implementation

Implementation

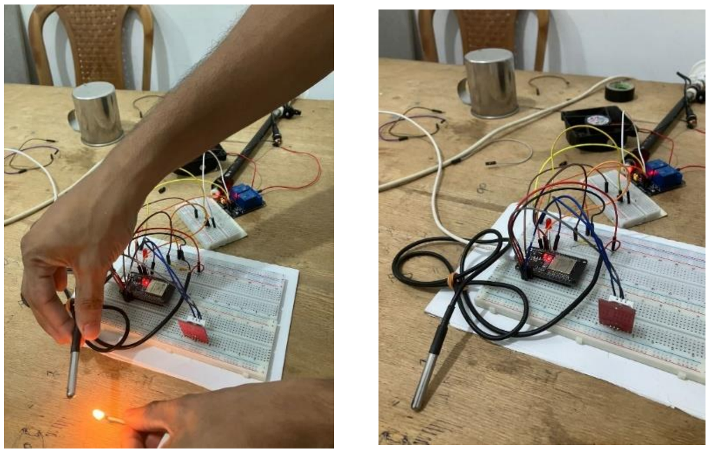
The implementation phase of the "Automatic Dehydrator" project involved a systematic approach to integrating various hardware and software components to achieve a functional and reliable system. The process began with the selection and assembly of the primary hardware components, including the ESP32 microcontroller, DS18B20 temperature sensor, MQ-02 gas sensor, DHT22 humidity sensor, and the PTC heating element. Each component was chosen for its specific capabilities and suitability for the project's requirements.

The ESP32 microcontroller served as the central control unit, managing data inputs from the sensors and controlling the heating element and other actuators. The integration of these components required careful planning of the circuit design to ensure proper power distribution, signal integrity, and communication between the sensors and the microcontroller. Custom PCBs were designed and fabricated to streamline the assembly process and minimize wiring errors.

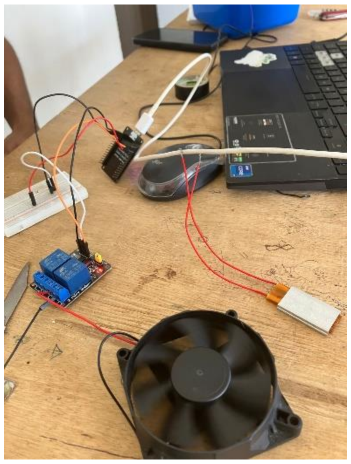
Software development played a critical role in the implementation phase. The firmware for the ESP32 was developed using the Arduino IDE, with code written to handle sensor data acquisition, processing, and control logic. The system was programmed to continuously monitor temperature, humidity, and gas levels, adjusting the heating element operation to maintain optimal drying conditions. A web-based user interface was developed using HTML, CSS, and JavaScript, allowing users to configure settings, monitor real-time data, and receive alerts. This interface was hosted on a local server, ensuring easy access from any web browser on the local network.

Testing

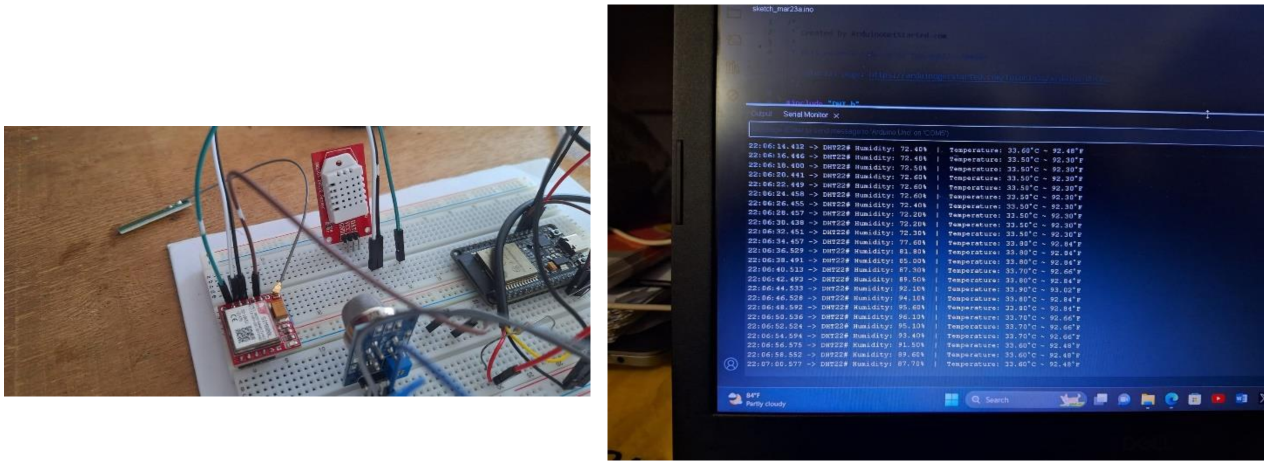
1. Unit Testing: Each hardware component was tested independently to verify its functionality. This included checking the accuracy of the temperature and humidity sensors, ensuring the MQ-02 gas sensor responded correctly to gas presence, and validating the operation of the PTC heating element. Calibration procedures were conducted to fine-tune sensor readings and ensure precision.



Testing the temperature sensor

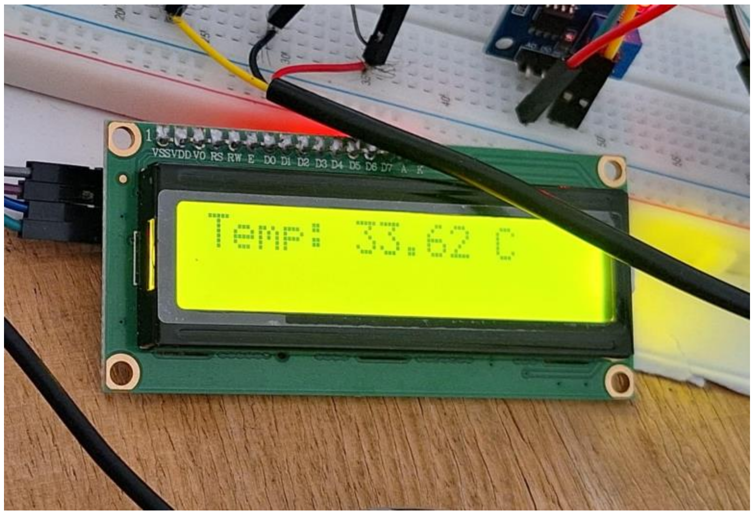


Testing the PTC heating element and fan

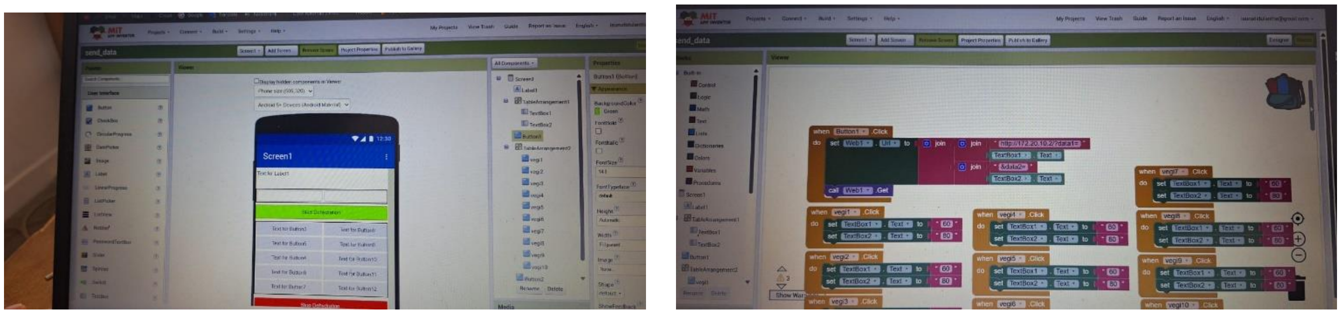


```
22:06:14.412 -> DHT22# Humidity: 72.40% | Temperature: 33.60°C ~ 92.48°F
22:06:16.446 -> DHT22# Humidity: 72.40% | Temperature: 33.50°C ~ 92.30°F
22:06:18.400 -> DHT22# Humidity: 72.50% | Temperature: 33.50°C ~ 92.30°F
22:06:20.441 -> DHT22# Humidity: 72.60% | Temperature: 33.50°C ~ 92.30°F
22:06:22.449 -> DHT22# Humidity: 72.60% | Temperature: 33.50°C ~ 92.30°F
22:06:24.458 -> DHT22# Humidity: 72.60% | Temperature: 33.50°C ~ 92.30°F
22:06:26.455 -> DHT22# Humidity: 72.40% | Temperature: 33.50°C ~ 92.30°F
22:06:28.457 -> DHT22# Humidity: 72.20% | Temperature: 33.50°C ~ 92.30°F
22:06:30.438 -> DHT22# Humidity: 72.20% | Temperature: 33.50°C ~ 92.30°F
22:06:32.451 -> DHT22# Humidity: 72.30% | Temperature: 33.50°C ~ 92.30°F
22:06:34.457 -> DHT22# Humidity: 77.60% | Temperature: 33.80°C ~ 92.84°F
22:06:36.529 -> DHT22# Humidity: 81.80% | Temperature: 33.80°C ~ 92.84°F
22:06:38.491 -> DHT22# Humidity: 85.00% | Temperature: 33.80°C ~ 92.84°F
22:06:40.513 -> DHT22# Humidity: 87.30% | Temperature: 33.70°C ~ 92.66°F
22:06:42.493 -> DHT22# Humidity: 89.50% | Temperature: 33.80°C ~ 92.84°F
22:06:44.533 -> DHT22# Humidity: 92.10% | Temperature: 33.90°C ~ 93.02°F
22:06:46.528 -> DHT22# Humidity: 94.10% | Temperature: 33.80°C ~ 92.84°F
22:06:48.592 -> DHT22# Humidity: 95.60% | Temperature: 33.80°C ~ 92.84°F
22:06:50.536 -> DHT22# Humidity: 96.10% | Temperature: 33.70°C ~ 92.66°F
22:06:52.524 -> DHT22# Humidity: 95.10% | Temperature: 33.70°C ~ 92.66°F
22:06:54.594 -> DHT22# Humidity: 93.40% | Temperature: 33.70°C ~ 92.66°F
22:06:56.575 -> DHT22# Humidity: 91.50% | Temperature: 33.60°C ~ 92.48°F
22:06:58.552 -> DHT22# Humidity: 89.60% | Temperature: 33.60°C ~ 92.48°F
22:07:00.577 -> DHT22# Humidity: 87.70% | Temperature: 33.60°C ~ 92.48°F
```

Testing DHT22 sensor

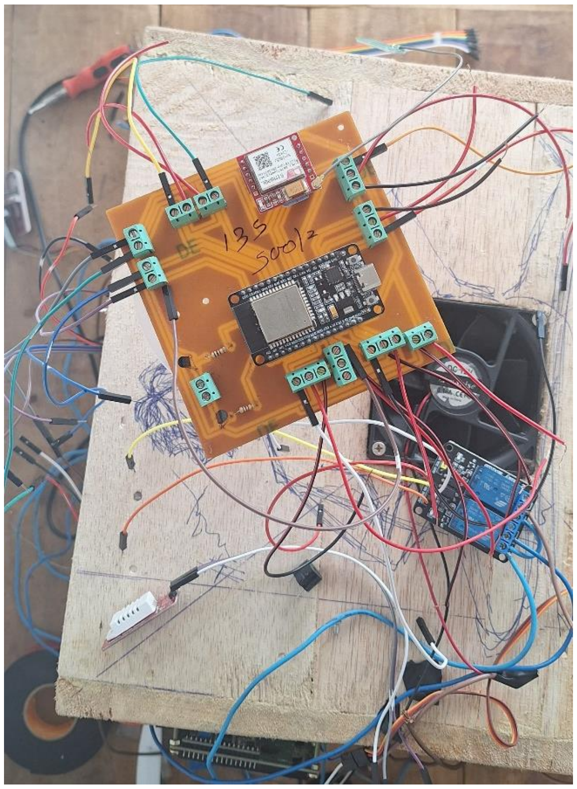


Testing the LCD display



Testing the mobile app

2. System Testing: The fully integrated system was subjected to a series of controlled experiments to evaluate its performance under different environmental conditions. Various types of food items were dehydrated to assess the system's ability to maintain consistent drying conditions. Data was collected on temperature, humidity, and gas levels throughout the drying process to identify any deviations and make necessary adjustments.



4. User Acceptance Testing : The final stage of testing involved real users interacting with the system to provide feedback on usability and functionality. This feedback was crucial for refining the user interface and making any final adjustments to the system's operation.

The testing and implementation phases were iterative, with insights from each testing stage informing further refinements and improvements. The result was a robust and reliable "Automatic Dehydrator" system that effectively addresses the challenges of traditional dehydration methods, providing consistent quality, enhanced safety, and improved efficiency.

6. Source Code

```
#include <OneWire.h>
#include <DallasTemperature.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include "DHT.h"
#include <WiFi.h>
#include <WebServer.h>
#include <ESP32Servo.h>
#include <SoftwareSerial.h>
#include "time.h"
#include "esp_sntp.h"

#define MQ2pin 25
#define fan 26
#define heater 15
#define ONE_WIRE_BUS 13
#define DHTPIN1 14
#define DHTPIN2 4
#define LED1 18
#define LED2 19
#define Buzzer 23
#define servoPin 27
SoftwareSerial mySerial(17, 16);
#define DHTTYPE DHT22

const char* ssid = "M12";
const char* password = "12345678";


const char *ntpServer1 = "pool.ntp.org";
const char *ntpServer2 = "time.nist.gov";
const long gmtOffset_sec = 19800;
const int daylightOffset_sec = 0;
const char *time_zone = "IST-5:30";

time_t startTime;
time_t finishTime;
bool timing = false;


DHT dht1(DHTPIN1, DHTTYPE);
DHT dht2(DHTPIN2, DHTTYPE);


OneWire oneWire(ONE_WIRE_BUS);

const int LCD_ADDRESS = 0x27;
```

```
LiquidCrystal_I2C lcd(LCD_ADDRESS, 16, 2);

Servo myservo;

unsigned long startMillis;
DallasTemperature sensors(&oneWire);

WebServer server(80);
String data1 = "";
String data2 = "";
float temperature = 0.0;
long timeDuration=0;
long durationInSeconds =0;
int sensorValue;


void displayWaitingUntil(time_t finishTime) {
  initializeLCD("Waiting until:", 0, 0);
  printTimeToLCD(finishTime);
}


void printTimeToLCD(time_t time) {
  struct tm *timeinfo;
  timeinfo = localtime(&time);
  char buffer[16];
  strftime(buffer, sizeof(buffer), "%H:%M:%S", timeinfo);
  initializeLCD(buffer, 0, 1);
}
void printLocalTime() {
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    Serial.println("No time available (yet)");
    return;
  }
  Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");
}

void timeavailable(struct timeval *t) {
  Serial.println("Got time adjustment from NTP!");
  printLocalTime();
}

void startTiming() {
  struct tm timeinfo;
  if (getLocalTime(&timeinfo)) {
    startTime = mktime(&timeinfo);
    finishTime = startTime + durationInSeconds;
    timing = true;
    Serial.println("Timing started at:");
    printLocalTime();
    Serial.println("Finishing time will be:");
    printTime(finishTime);
  }
}

void printTime(time_t time) {
  struct tm *timeinfo;
  timeinfo = localtime(&time);
  Serial.println(asctime(timeinfo));
}

void updateSerial() {
    delay(500);
    while (Serial.available()) {
        mySerial.write(Serial.read());
    }
    while (mySerial.available()) {
        Serial.write(mySerial.read());
    }
}

void sendSMS(String message) {
    mySerial.println("AT+CMGS=\"+94768633308\""); // Replace with the actual phone number
    updateSerial();
    delay(100);
    mySerial.print(message);
    updateSerial();
    delay(100);
    mySerial.write(26);
    updateSerial();
    delay(100);
}
void handleRoot() {
  if (server.hasArg("data1") && server.hasArg("data2")) {
    data1 = server.arg("data1");
    data2 = server.arg("data2");
    temperature = data1.toFloat();
    timeDuration = data2.toInt();
    Serial.print("Received data1 (temperature): ");
    Serial.println(temperature);
    Serial.print("Received data2 (time duration): ");
    Serial.println(timeDuration);
    server.send(200, "text/plain", "Data received and processed");
  } else {
    server.send(200, "text/plain", "Send data with 'data1' and 'data2' parameters");
  }
}

float temp(){
    sensors.requestTemperatures();
    float temperatureC = sensors.getTempCByIndex(0);
    if (temperatureC == DEVICE_DISCONNECTED_C) {
          Serial.println("Error: Could not read temperature data");
          lcd.clear();
          lcd.setCursor(0, 0);
          lcd.print("Temp read error");
        } else {
          Serial.print("Temperature: ");
          Serial.print(temperatureC);
          Serial.println(" °C");
          char tempStr[16];
          snprintf(tempStr, sizeof(tempStr), "Temp In:%.2f C", temperatureC);
          initializeLCD(tempStr, 0, 0);
          return temperatureC;
}
}

void initializeLCD(const char* message, int col, int row) {
  Wire.begin();
  lcd.init();
  lcd.backlight();
  delay(1000);
  lcd.clear();
  lcd.setCursor(col, row);
  lcd.print(message);
}



void setup() {
  Serial.begin(9600);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.println("Connecting to WiFi...");
    delay(1000);
  }
  sntp_set_time_sync_notification_cb(timeavailable);
  esp_sntp_servermode_dhcp(1);
  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer1, ntpServer2);
  Serial.println("Connected to WiFi");
  initializeLCD("Connected to WiFi",0,0);
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
  String ipStr = WiFi.localIP().toString();
  char ips[16];
  ipStr.toCharArray(ips, 16);
  initializeLCD(ips, 0, 0);
  server.on("/", handleRoot);
  server.begin();
  pinMode(MQ2pin, INPUT);
  Serial.println("MQ2 warming up!");
  initializeLCD("Gas checking...",0,0);
  delay(20000);
  myservo.attach(servoPin);
  myservo.write(0);
```

```
delay(500);
pinMode(fan, OUTPUT);
pinMode(heater, OUTPUT);
pinMode(LED1, OUTPUT);
pinMode(LED2, OUTPUT);
pinMode(Buzzer, OUTPUT);
dht1.begin();
dht2.begin();
sensors.begin();
mySerial.begin(9600);
Serial.println("Initializing...");
delay(1000);
mySerial.println("AT");
updateSerial();
delay(100);
mySerial.println("AT+CMGF=1");
updateSerial();
delay(1000);
while (true) {
  myservo.write(0);
  server.handleClient();
  durationInSeconds=timeDuration*60;
  startTiming();
  if (temperature > 0.0 && timeDuration > 0) {
    sensorValue = digitalRead(MQ2pin);
    Serial.println(sensorValue);
    if (sensorValue==0) {
       digitalWrite(heater, LOW);
       Serial.println("Gas detected");
       initializeLCD("Gas detected!!!",0,0);
       myservo.write(180);
       digitalWrite(fan, HIGH);
    } else {
      myservo.write(0);
      digitalWrite(fan, LOW);
      Serial.println("No gas detected");
      initializeLCD("Process starting...",0,0);
      Serial.println(temperature);
      Serial.println(timeDuration);
      while(true){
             if (timing) {
      time_t now;
      struct tm timeinfo;
  if (getLocalTime(&timeinfo)) {
    now = mktime(&timeinfo);
    if (now >= finishTime) {
      digitalWrite(heater, LOW);
      Serial.println("Finishing time reached!");
      initializeLCD("Finishing time ",0,0);
      initializeLCD("reached!",0,1);

      while(true){
```

```
  float dhtTemperatureC = dht1.readTemperature(); // temp outside
  float humidity = dht1.readHumidity();// huminity outside
  float humidityI = dht2.readHumidity();// huminity inside
 if (isnan(dhtTemperatureC) || isnan(humidity)) {
    Serial.println("outside DHT22 error!");
    initializeLCD("outside DHT22 error!", 0, 0);
  } else {
      Serial.println("Humidity outside: ");
      Serial.print(humidity);
      Serial.print(" %\t");
      Serial.print("DHT Temperature OUTSIDE: ");
      Serial.print(dhtTemperatureC);
      Serial.print(" °C ");

      char TStr[16];
      snprintf(TStr, sizeof(TStr), "Temp  Out:%.2f ",dhtTemperatureC );
      initializeLCD(TStr, 0, 0);
    }
    //Inside Humidity
    if (isnan(humidityI)) {
      Serial.println("Failed to read from inside  DHT sensor!");
      initializeLCD("inside DHT22 error!", 0, 0);
    } else {
      Serial.print("Humidity inside: ");
      Serial.print(humidityI);
      Serial.print(" %\t");

    }
      char HStr[16];
      snprintf(HStr, sizeof(HStr), "H. out :%.2f ", humidity);
      initializeLCD(HStr, 0, 0);
      char HIStr[16];
      snprintf(HIStr, sizeof(HIStr), "H. in:%.2f ",humidityI );
      initializeLCD(HIStr, 0, 0);


  float t3=temp();
  if ((dhtTemperatureC + 3.0 < t3) || (humidityI > humidity)) {
    initializeLCD("System cool down...", 0, 0);
    digitalWrite(heater, LOW);
    myservo.write(180);
    digitalWrite(fan, HIGH);

  }else{
  myservo.write(0);
  digitalWrite(heater, LOW);
  digitalWrite(fan, LOW);
  digitalWrite(LED1, HIGH);
  Serial.println("sending message");
  sendSMS("process over");
  while(true){
    initializeLCD("process over.....", 0, 0);
```

```
            digitalWrite(LED1, HIGH);
            digitalWrite(Buzzer, HIGH);
            delay(3000);
            digitalWrite(LED1, LOW);
            digitalWrite(Buzzer, LOW);
          }
          }
        }
        timing = false;
      } else {
        Serial.print("Current time: ");
        printTime(now);
        Serial.print("Waiting until: ");
        printTime(finishTime);
        displayWaitingUntil(finishTime);
        Serial.println("dehydrating on");
        initializeLCD("dehydrating on...", 0, 0);
        Serial.println("heating to:");
        Serial.print(temperature);
        char Utemp[16];
        snprintf(Utemp, sizeof(Utemp), "Heating to:%.2f ",temperature );
        initializeLCD(Utemp, 0, 0);
        float t2 = temp();
        if(t2>70){
              myservo.write(180);
              digitalWrite(LED2, HIGH);
              digitalWrite(Buzzer, HIGH);
              sendSMS("Over heating...");
              initializeLCD("Over heating", 0, 0);
              exit(0);
        }
        else if(t2 < temperature) {
              myservo.write(0);
              Serial.println("heater on....");
              initializeLCD("Heater on", 0, 0);
              digitalWrite(heater, HIGH);
            } else {
              myservo.write(0);
              Serial.println("heater off....");
              initializeLCD("Heater off", 0, 0);
              digitalWrite(heater, LOW);
            }
          }
      }
    }
      delay(1000);}}
    delay(2000);}
  delay(2000);}}
void loop(){
}
```

7. Future Work

Solar Power Integration: Adding a solar power system can make the dehydrator more energy-efficient and environmentally friendly. Solar panels can be used to power the entire system or supplement its power needs, reducing reliance on conventional electricity sources. This is particularly beneficial for users in remote or off-grid locations where access to stable electricity may be limited.

Advanced Sensor Array for Enhanced Monitoring: Expanding the sensor array to include additional sensors such as VOC (Volatile Organic Compounds) sensors, CO2 sensors, and more precise humidity sensors can provide a more comprehensive monitoring system. These sensors can offer detailed insights into the drying environment, allowing for finer control and adjustments to maintain optimal drying conditions. Enhanced monitoring can lead to better product quality and safety.

8. Estimated Cost

Component or sensor	Quantity	Unit Price	Total (Rs.)
ESP32	1	1350.00	1350.00
DS18B20 Sensor	1	310.00	310.00
MQ-02 Gas Sensor	1	395.00	395.00
DHT22 Sensor	1	550.00	550.00
Relay 2 channel 12V	2	300.00	600.00
GSM Module	1	2250.00	2250.00
LCD Display 16x2	1	480.00	480.00
12C module for lcd display	1	200.00	200.00
PTC heating element	4	1770.00	7080.00
Buzzer	1	50.00	50.00
Exhausting fan	1	200.00	200.00
Linear actuator	1	1900	1900
Ressistor 1KΩ	4	60.00	240.00
Ressistor 4.7KΩ	1	60.00	60,00
Transistor BC547	2	8.00	16.00
Structure	1	7000.00	7000.00
Servo motor	1	320.00	320.00
Total			**23000.00**

9. Individual Contibutions

L.Laavanjan (225527H)

- Testing and implementation of RTC module.
- Integration of GSM module.
- Testing and implementation of Exhaust fan
- Coordinating the whole project. .

GSM module

In this project, my primary responsibility was the implementation and management of the GSM module [2], which played a crucial role in the communication and safety features of the automatic dehydrator system. The GSM module was configured to send SMS notifications to users under specific conditions.

Firstly, it was programmed to notify the user when the dehydration process was complete, ensuring timely updates and allowing for immediate action. Secondly, the module sent an alert when the internal temperature of the dehydrator exceeded 80 degrees Celsius, providing an essential safety mechanism to prevent overheating and potential damage to the system or the dehydrated food.

To integrate the GSM module effectively, I utilized the <SoftwareSerial.h> library, creating a software serial interface with the lines `SoftwareSerial mySerial(17, 16);` for RX and TX communication. This approach facilitated reliable data exchange between the GSM module and the ESP32 microcontroller, enabling seamless transmission of SMS alerts. My contribution ensured that the system could communicate critical information to users in real-time, enhancing both the functionality and safety of the automatic dehydrator.

RTC module

The RTC module [3]was essential for accurately tracking and managing the timing of the dehydration process. Upon initiation of the operation, the RTC recorded the starting time. The time limit for the dehydration process was provided as an input through the mobile app interface. By calculating the finishing time based on the starting time and the specified time limit, the RTC ensured precise control over the duration of the operation. As the process approached the calculated finishing time, the RTC module sent a signal to the system to terminate the operation, thereby preventing over-dehydration and ensuring optimal results.

Subramaniam A 225544G

- Testing and implementation of DHT22 sensor.
- PCB design.

DHT22 Sensor

Our goal was to integrate DHT22 [4] sensors for temperature and humidity measurement, both inside and outside the dehydrator, ensuring accurate data collection and utilization within the system. For that we have chosen DHT22 sensors due to their reliability and accuracy for both temperature and humidity measurements. It was able to measure temperature in range -40oC – 80oC.

I integrated the DHT library into the project to facilitate easy interaction with the DHT22 sensors. This involved adding the necessary library includes and defining the sensor pins and types.

```
#include "DHT.h"

#define DHTPIN1 14

#define DHTPIN2 4

#define DHTTYPE DHT22

DHT dht1(DHTPIN1, DHTTYPE);

DHT dht2(DHTPIN2, DHTTYPE);
```

Code Implementation

In the setup() function, I initialized the DHT22 sensors to ensure they were ready to start collecting data. I then implemented the sensor reading logic in the loop() function, making sure to handle any potential errors that might arise during data collection. This allowed the system to continuously monitor temperature and humidity levels both inside and outside the dehydrator.

Integration with Overall System

I ensured seamless integration of the DHT22 sensor data with the overall control logic of the dehydrator. The temperature and humidity data collected from the sensors were incorporated into the decision-making processes for controlling the fan and ptc heating element.

Exhaust fan

In addition to managing the GSM and RTC modules, I was also responsible for the integration and operation of the exhausting fan within the automatic dehydrator system. The fan plays a critical role in maintaining a safe and controlled environment. It is programmed to activate immediately after the heating process concludes, facilitating the rapid cooling of the system and ensuring that the dehydrated food is not exposed to excessive residual heat. Furthermore, the fan is triggered by the MQ-02 gas sensor when harmful gases are detected, effectively ventilating the system and mitigating potential health risks or hazards. My contribution in implementing the exhausting fan ensured both operational safety and environmental control, enhancing the overall functionality and reliability of the dehydrator.

P.K.I Didulantha: 225510B

- Testing and implementation of LCD display.
- Mobile App Development
- PCB Design
- Servo motor

LCD display

The LCD display serves as a critical interface for user interaction, providing real-time information about the ongoing processes. It displays key data such as current temperature, humidity levels, time remaining for the dehydration process, and any alerts or notifications. This visual feedback ensures that users are well-informed about the system's status and can make timely decisions or adjustments if necessary. My contribution in implementing the LCD display improved user experience and interaction, making the dehydrator more user-friendly and accessible.

Mobile App

Using MIT App Inventor, I created an intuitive and user-friendly interface that allows users to select from a variety of preset options. Each selection has its own specific time limit and ideal temperature, which are transmitted to the system to customize the dehydration process. This mobile app enhances the flexibility and convenience of the dehydrator, enabling users to tailor the drying parameters to different types of food with ease. My contribution in developing the mobile app ensured that the system is accessible and adaptable to user preferences, significantly improving its usability and functionality.

Servo motor

The servo motor is programmed to open the lid of the dehydrator when the exhaust fan starts working, ensuring effective ventilation and cooling. This mechanism is crucial for maintaining the quality and safety of the dehydrated food, as it allows hot air and any detected harmful gases to be efficiently expelled from the system. My contribution in integrating the servo motor added an additional layer of automation and safety, enhancing the overall functionality and reliability of the dehydrator.

K.M.T.N.B Kulasekara (225524V)

- Testing and implementation of temperature sensor
- PTC heating element Control
- Making the structure

Temperature sensor

The integration and management of the DS18B20 temperature sensor within the automatic dehydrator system is crucial for accurately monitoring the internal temperature during the heating process. The DS18B20 provides precise temperature readings, allowing the system to maintain optimal drying conditions and ensure the safety and quality of the dehydrated food. By continuously tracking the temperature, the sensor helps prevent overheating and ensures that the dehydration process is carried out efficiently. My contribution in implementing the DS18B20 temperature sensor was essential for achieving precise temperature control and enhancing the overall reliability of the dehydrator system.

PTC heating element

The PTC heating element,is crucial for the heating process in the automatic dehydrator. The PTC heating element provides the necessary heat to dry the food items efficiently. I ensured that the heating element operates within the optimal temperature range specified by the user via the mobile app. This involved integrating the heating element with the system's control algorithms to maintain consistent and accurate heating throughout the dehydration process. My contribution in implementing the PTC heating element was essential for achieving the desired drying conditions, ensuring the effectiveness and efficiency of the dehydrator.

Prainila R (225535F)

- Testing and implementation of MQ02 gas sensor
- Testing and implementation of Buzzer and LEDs

MQ02 gas sensor

The MQ-02 sensor [5] is crucial for detecting harmful gases that may be released after the heating process in the dehydrator. When the MQ-02 sensor detects the presence of dangerous gases, it triggers the exhausting fan to activate, ensuring the safe ventilation of the system. My contribution in implementing the MQ-02 sensor enhanced the safety features of the dehydrator, protecting both the equipment and the users from potential hazards.

Integration and Setup:

The MQ2 sensor was connected to the ESP32's analog input pin (A2). The sensor's analog output provides a voltage that corresponds to the concentration of gases detected. This voltage is read by the ESP32 and used to determine the gas levels.

Buzzers and LEDs

Furthermore, I was responsible for integrating and managing the buzzers and LEDs in the automatic dehydrator system. These components play a crucial role in providing auditory and visual feedback to the user. The buzzers are programmed to sound alerts in case of any anomalies, such as the detection of harmful gases by the MQ-02 sensor or the completion of the dehydration process. The LEDs provide visual indicators of the system's status, such as power on/off, heating in progress, and alerts for any issues that require attention. My contribution in implementing the buzzers and LEDs ensured that the system offers clear and immediate feedback, enhancing user awareness and safety during the dehydration process.

Integration and Setup:

The LED and buzzer were connected to the ESP32's digital output pins (LED_PIN and BUZZER_PIN). The ESP32 controls these components based on the gas concentration readings from the MQ2 sensor.

10. References

[1] Linda, "Dehydrator blog," 19 july 2017. [Online]. Available: https://dehydratorblog.com/fooddehydrating-time-temperature-guide/. [Accessed 10 februar 2024].

[2] component101, 30 september 2021. [Online]. Available: https://components101.com/wireless/sim800l-gsm-module-pinout-datasheet-equivalent-circuit-specs. [Accessed 2024 august 2024].

[3] "esp32io.com," ESP32I/O, [Online]. Available: https://esp32io.com/tutorials/esp32-rtc. [Accessed 7 february 2024].

[4] R. peraya, "Microcontroller Tutorials and Resources," 2 february 2019. [Online]. Available: https://www.teachmemicro.com/how-dht22-sensor-works/. [Accessed 2024 february 7].

[5] Sam, "random nerd turtorial," 5 may 2016. [Online]. Available: https://randomnerdtutorials.com/guide-for-mq-2-gas-smoke-sensor-with-arduino/. [Accessed 2 july 2024].