FOOD QUALITY DETECTION USING IOT

Documentation submitted to INDICOLD as per requirement of Internship

Submitted By

TEAM BROWN

- **A.B.Kamalesh (TL)**
- **B. Bhavyasree**
- **Darshan Kumar**
- **Neeluru Bhavana**
- **Priyanka J**

Under the guidance of

Mr. Kartik Jalan

COO of Kailash Agro cold Storage

INDICOLD INTERNSHIP

KAILASH AGRO PRIVATE LIMITED

PALWAL DISTRICT, HARIYANA – 121102
MAY 2021

ABSTRACT

In today's world, technology plays a significant role in shaping our way of life. Cold storage industries extend the shelf life of products and deliver them freshly by maintaining various parameters like temperature, humidity, etc. There are different variables on which food decomposition depends, the parameters like humidity, bacteria, and temperature are major factors on which the rate of decomposition of food depends on. Even after controlling various quantities like temperature and humidity if the food still degrades, they can be preserved or the necessary action can be taken by detecting the methane production. The aim of this project is to deliver fresh products to the customers. To achieve this, we have to constantly monitor the quality of the products. In this IoT project, we have designed a Food Monitoring device using Arduino Mega and connected the data through the Blynk app, to monitor the quality of the food and send alerts if the food is found degrading. To determine the status of the food, the MQ4 gas sensor, MQ135 gas sensor, and DHT 11 temperature modules are used.

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Food is the main energy source for all living beings. So, food quality and food safety will always be in the highest demand. To maintain good food quality, we need to constantly check the quality of food. In this present era of technological advancement, the Internet of Things (IoT) is growing faster and faster. By utilizing this IoT technology in cold storage, we can trace and track the condition of food. IoT sensors are used here to test the freshness level of food. These sensors are directly or indirectly connected to the IoT networks. The data obtained from these IoT sensors will be shared live with consumers and cold storage supervisors through a mobile application called "BLYNK".

It is very important to monitor the quality of food. It should be prevented from spoiling or decaying. Chemical gas sensors like MQ4 and MQ135 are used here. MQ4 gas sensing module is used to detect the emission of methane concentration from food. MQ135 gas sensor is used to detect the emission of carbon dioxide concentration. The quality of food also depends on atmospheric factors like temperature and humidity. So DHT11 sensor is used here, which is used to measure temperature and humidity. The DHT11 sensor mainly consists of two components. They are a humidity sensing component and thermistor or NTC temperature sensor. They sense temperature and humidity from surrounding air.

MQ4, MQ135, and DHT11 sensors indicate analog value. The analog output from these sensors is passed to the analog pin of Arduino Mega, which has an ADC that converts analog to a digital value. These digital values are then displayed on the BLYNK mobile application. This paper further demonstrates a detailed explanation of the IoT framework for detecting food quality in cold storage and delivering fresh products to customers.
It has an upper edge in terms of keeping track of food quality as in some cases even though there's **no change in temperature or humidity but food starts degrading.** In such situations also our model would detect food degradation.

1.2 PROBLEM STATEMENT

To deliver fresh products by monitoring the quality using IoT with the help of sensors like DHT 11, MQ4 and MQ135 through Blynk app.

1.3 EXISTING SYSTEM AND EMERGING TRENDS:

IoT setup in commercial and industrial spaces, some successful examples:

- Smart agriculture systems
- Weather monitoring system
- Home automation system
- Face recognition bot
- Smart alarm detector
- Pollutants monitoring system
- Smart parking system
- Smart traffic management system
- Smart gas leakage detection system
- Smart energy grid
- Smart robotic loading-unloading
- Smart vehicle tracker, etc
- Smart life support system (the room temperature, moisture sensors used in ICU)

1.4 LIMITATIONS OF EXISTING MODELS (IN MARKET):

Some limitations in brief:

- Lack of understanding, in remote users, about the remote interface available to them.
- Leaking of passwords, due to negligent staff.
- Lack of understanding in remote users, about working of sensors.
- The app interface does not maximize the information conveyance, like signalling of errors, in a particular sensor, etc.
- Inefficient code, reduces the efficiency of hardware, thus reducing profit.
- Lack of mechanism to signal errors, for example: to signal that SD card module, is not getting initialized.
- Lack of implementation of "error handling". For example: if the SD module fails to initialize, then wait for 500 ms and try again, at least 5 times, before finally raising an error.
- Lack of proper documentation by manufacturers, about the sensor/SD being used, which makes it difficult for programmers, to be sure of their code.
- The above reason, makes the code unpredictable sometimes.
- Lack of testing of the prototype, in the surrounding, in which it is meant to be installed.

CHAPTER 2
PROPOSED METHODOLOGY

2.1 PROPOSED MODEL

Our model has the following components:

- MQ-4 gas sensor- to detect methane
- DHT-11 sensor- to measure surrounding moisture, temperature
- MQ-135 sensor - to measure CO2
- SD card module, to store the reading, (helpful in case the Wi-Fi connection fails).
- Arduino micro-controller to compile code and instruct the sensors.
- Blynk server
- Remote user (with login, password).
- Blynk App (in remote user's phone)- with standard authentication
- Documentation - regarding how to conclude, from the app.

The above components, enable our model to achieve a full-scale IoT integration, of sensors, and remote users.

2.2 MERITS OF OUR MODEL

- Smart gas detection system.
- Mq135 has high sensitivity towards ammonia, sulfide, benzene, alcohol, SMOKE, CO2.
- MQ-4 can detect high concentrations of methane.
- Pollutant monitoring system: as major pollutants are sulfides; nitrogenous compounds thus can be guessed by Mq135.
- Fire detection system: smoke in the air can be detected by mq135.
- Toxic gas detection: the presence of poisonous gases in a warehouse can contaminate edible products, and is harmful to the workers being employed there.MQ135 can primarily detect smoke.
- MQ4 can detect methane from a range [300,10000] ppm.
- Our prototype can successfully operate from -10 to 50C, without any lagging.
- Smart alarm system: It is programmed to show the measured values of the gases in ppm, with color labels showing the level (safe, normal, dangerous).
- The above alarming system will help even the illiterate workers, to understand the warning, and take necessary action.
- The variations in the readings, in shown live with a gauge widget, with fast-flexible animations.
- The data is stored in an SD card, which acts as an offline database, and keeps storing data, irrespective of internet connection.
- The data stored in an SD card can be used for data analysis, investigation, in case of any fire accident., etc.
- It provides REAL-TIME MONITORING, of a warehouse.
- Provides remote accessibility, alarm system.

2.3 DEMERITS OF OUR MODEL, IN COMPARISON WITH EXISTING MODELS IN MARKET:

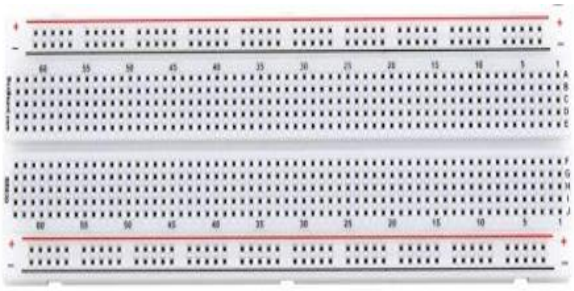
- The front-end role of the BLYNK APP brings some limitations.
- The data analysis cannot be done by the Blynk app (it is out of scope for Blynk)
- It has limited free widgets.
- Once data is pushed to virtual terminals of Blynk Server, we cannot retrieve it back, thus the data is wasted, just for showing animations in widgets.
- The internet speed requirements of Blynk are high, which limits its usage, in remote areas.
- The data stored in SD card, is the ONLY WAY, to do data analysis, and thus DATA ANALYSIS cannot be done in parallel when the device is working (as read, write cannot be done at the same time, in SD card, when the device is working, data will be continuously written in SD card).
- There is no scope of showing analysis of data, in the Blynk app, as no such widget is available.
- The number of users accessing the same project using the Blynk App is limited. This could be a major limitation while testing, deploying.

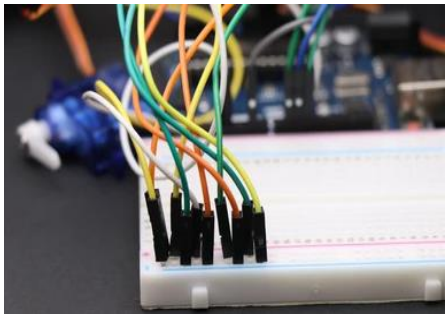
CHAPTER 3
COMPONENTS, WIRING AND SOURCE CODE

3.1HARDWARE DESCRIPTION

3.1.1 BREADBOARD AND JUMPER WIRES:

A breadboard is used in giving the connections between various components like resistors, sensors, and modules, etc. There are holes into which we can insert wires for the connection. Jumper wires are used to give the connections. The components that are to be connected using breadboard are given connection with the help of these wires.





3.1.2 ARDUINO MEGA 2560:

The Arduino Mega 2560 could also be a microcontroller board supporting the ATmega2560. it's programmable via Arduino IDE. a number of the features include 54 digital input/output pins, 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz quartz oscillator, a USB connection, an influence jack, an ICSP header, and a push. This board is meant for larger sketches, more input/output lines, and more RAM.

3.1.3 MQ135:

MQ135 is a sensor with a 4-pin multi-use sensor. It is often wont to sense gas like carbon-di-oxide, benzene, alcohol, and smoke. The MQ135 sensor has an electrochemical sensor inside it, and this sensor is sensitive to a variety of gasses are used at temperatures.

3.1.4 DHT11:

DHT11 is also a 4-pin multi-use sensor. It is a basic and ultra-low-cost sensor used to measure temperature and humidity. It uses a thermistor and capacitive humidity sensor to measure temperature and humidity from the surrounding air i.e., from the environment. No analog input pins are required here. It will spit out digital signals on the data pin. It is very simple to use. It requires proper timing to grab data.

3.1.5 ESP8266-01(WI-FI MODULE):

ESP8266 could even be a Wi-Fi Module, could even be a self-contained SOC (System-on-Chip) with an integrated TCP/IP protocol stack which can give any microcontroller access to your Wi-Fi network. The ESP8266 is capable of either offloading all Wi-Fi networking functions or hosting an application from another application processor.

3.1.6 MQ4:

MQ4 could also be a gas sensing module, which is used to measure methane gas within the atmosphere. It contains Gas sensing layer, which is made from SnO2. SnO2 is sensitive to gases like H2, CH4, LPG, CO, Alcohol, and smoke. because the decaying food emits methane gas (CH4), the MQ4 sensor is often used to measure this gas to observe food quality

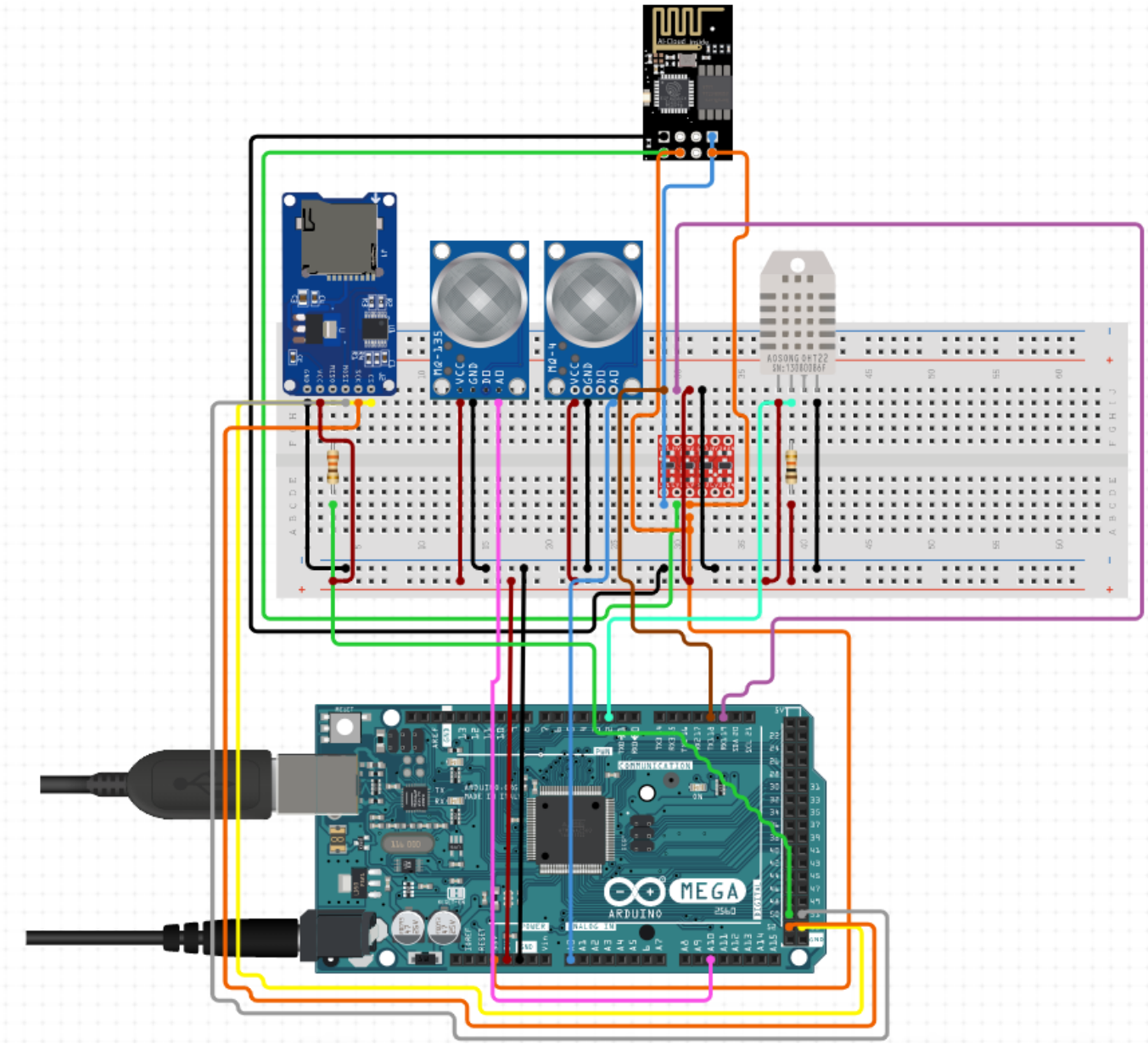
3.2 SOFTWARE USED:

- Blynk app
- Arduino IDE

3.3 DESIGN OF OUR PROTOTYPE:

The prototype is designed on an Arduino Mega 2560 Board. ESP8266-01 Wi-Fi module is connected to the board through Serial1 pins- 19(RX1)/18(TX1) to provide internet connectivity. The gas sensors MQ4 and MQ135 and DHT 11 module are connected to the 5 Volt of the external breadboard power supply. The Micro-SD card module is also powered through the Arduino 5 Volts. For better stability, the ESP8266-01 can be powered through an external 3.3 Volt supply to prevent current/power-related issues.

3.4 WIRING



3.5 DATA CONSUMPTION AND BATTERY USAGE OF COMPONENTS:

S.NO	Components	Data/power consumption	Battery usage/life span
1	ESP8266(Wi-fi module)	The ESP8266 requires a 3.3V power supply and a 3.3 V logic levels for communication. Its GPIO pins are not tolerant. If you want to interface esp8266 with the Blynk app it consumes an extra supply of 3.3V, for that we will need to do some level shifting	ESP8266 chip with a typical 2500mAh LiPo battery would last for about 30 hours.
2	Arduino Mega 2560	It has an influence consumption of 16.74mA for a power supply of 3V.	In high power mode, an Arduino mega draws around 77.7mA and lasts for 54hrs or 2 days. But in low power mode, it draws around 31.7mA and would run for about 132 hrs or 6 days.
3	MQ4 gas sensor	5V	12-18 months
4	MQ135 gas sensor	5V	12-18 months
5	Micro SD card module	When reading data, a further 3-5 Watts are used. It uses about 1 Watt at max load and is far slower than a traditional drive.	10 years or more

3.6WORKING OF THE HARDWARE COMPONENTS IN DETAIL:

3.6.1 CALIBRATION OF MQ4 SENSOR:

To calculate the concentration of the gases we are exposing the sensor to, we need to calibrate it. These calculations can be done from the MQ-4 sensitivity characteristic chart that is provided by its datasheet.

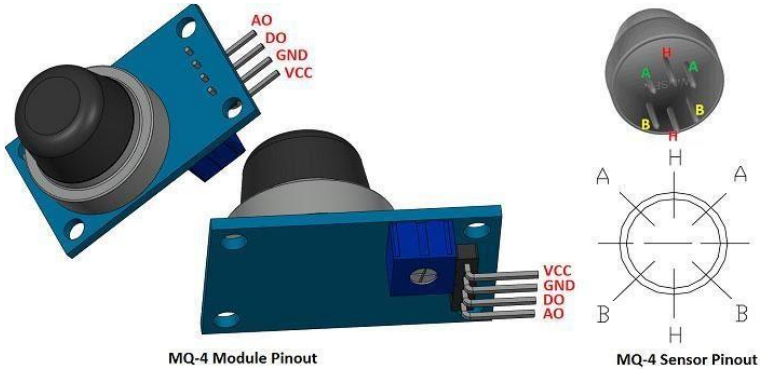
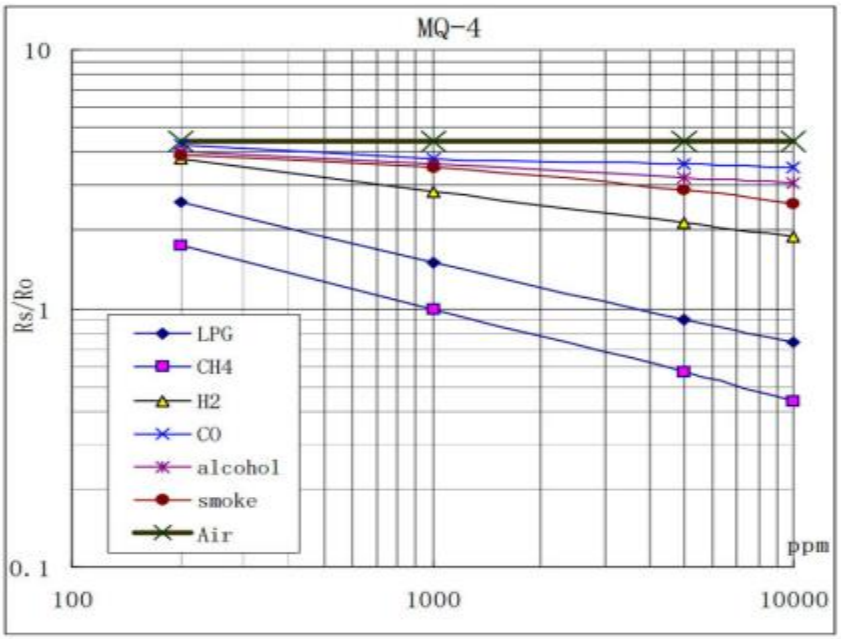


Fig: Sensitivity characteristics



The graph represents the concentration of gas ranging from 100 PPM to 10000 PPM on the x-axis and (RS/ R0) ration on the y-axis. Here, RS is the resistance of the concentration-dependent resistor, and R0 is the resistance of the same resistor when it is exposed to fresh air or a known concentration of gas. First, it is important to reference Figure to have a clear understanding of the path we will take to calculate concentration.

Let, RL is the resistance of the load resistor, VC is the input voltage powering the gas sensor, and VRL is the voltage over the load resistor. Ohm's Law gives us:

$$V = IR$$

where the quantities are voltage (V), current (I), and resistance (R). We can then combine RS and R0 since they are in series with each other and derive the current to be: I = VC / (RS + RL). Now we can attempt to solve for the output voltage at the load resistor by again using Ohm's Law. If we have: VRL = I. R

By substituting the current from the above equation, we get:

$$VRL = VC*RL/ (RS +RL)$$

Then solve for RS: $VRL *(RS + RL) = VC \cdot RL$

$$RS + RL = (VC \cdot RL) /VRL$$

$$RS = (VC \cdot RL /(VRL)) - RL$$

At this point, the values of VC and RL are known and the values of VRL are being outputted from the gas sensor for Arduino to calculate RS. Next, we can use the provided Arduino code in Appendix A.2 to calculate R0, the resistance of the concentration-dependent resistor when exposed to the fresh air. Within this code, R0 is calculated by calculating RS in fresh air using the same method as above then relating RS to R0 using the ratio:

$$RS/ R0 = 4.4 \text{ ppm}$$

We know this because the fresh air curve on the MQ-4 sensitivity characteristic chart has a constant y value of 4.4 ppm. Now we are prepared to analyze the sensitivity chart in Fig. The first thing we must notice is that the graph scale is logarithmic. This means that when put into a linear scale, gas concentration will change exponentially with the resistance ratio. First, it is best to treat the curve for each gas as linear so we can use a formula that linearly relates the concentration and resistance ratio. Once we have a linear equation relating these two values, we will be able to determine the concentration from the resistance values gathered from the Arduino board and the gas sensor. We will start with the equation of a line:

$$y = m x + b$$

where y is the resistance ratio value, m is the slope of the line, x is the concentration value, and b is the y-intercept. The logarithmic interpretation of this linear equation is as follows:

$$\log(y) = m \log(x) + b$$

Our next step is to calculate the slope. To do this we will need to choose two points on the same line, take the ratio of the change in y position to the change in x position, then use the logarithmic quotient rule to simplify the expression as follows:

$$m = (\log (yf) - \log(yi))/ (\log (xf) - \log(xi)) = (\log (yf/ yi)) / (\log (xf/ xi))$$

The slope of the desired line can be found by rearranging the equation

$$b = \log(y) - m \log(x)$$

Now that we have this information, we can once again rearrange Equation, so that we can solve for x, the concentration of a particular gas:

$$\log(y) = m \log(x) + b$$

$$m \log(x) = \log(y) - b$$

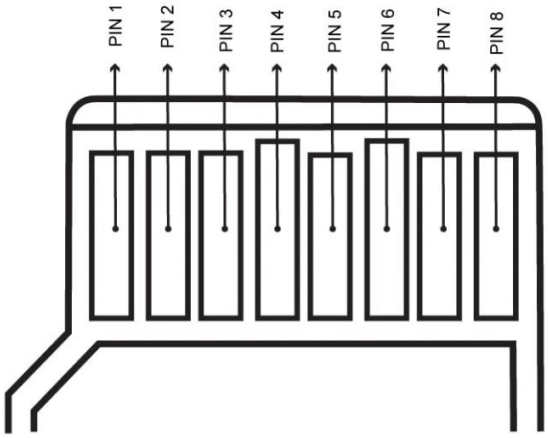
$$\log(x) = (\log(y) - b)/m$$

$$x = 10^{(\log(y) - b/ m)}$$

where y is the resistance ratio that was previously calculated.

3.6.2 MICRO SD CARD

Data storage of sensor values is a very important part of a project. There are various ways to store data according to the data type and size. Among all the storage devices, micro-SD card is one of the most practical ones, which are used in devices such as mobile phones, minicomputers, etc. In this project, we are implementing micro-SD card programming to store the data of sensor values for regular intervals of time. THE micro-SD card is the smallest memory card. Its size will be one quarter the size of a normal-sized SD card.
The Micro SD card has 8 pins as shown in the figure given below.



The table given below describes the function of each pin:

Pin	Name	Description
1	NC	Not Connected
2	CS	Chip Select/Slave select (SS)
3	DI	Master Out/Slave In (MOSI)
4	VDD	Supply Voltage
5	CLK	Clock (SCK)
6	VSS	Supply voltage ground
7	DO	Master In/Slave Out (MISO)
8	RSV	Reserved

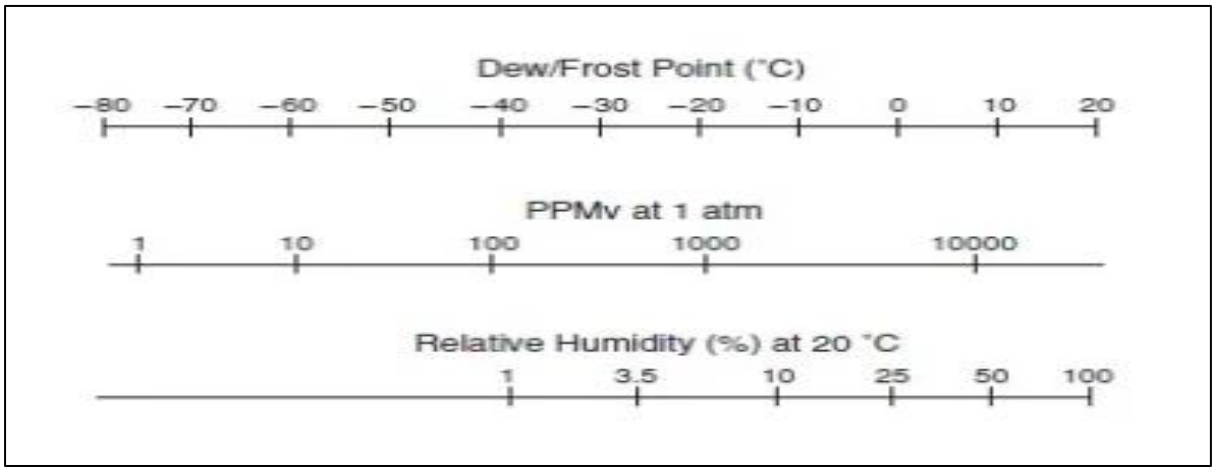
Connect micro-SD card to the Arduino Mega. Ensure that you have connected the pins of the SD card to the appropriate pins of the Arduino Mega board. We can communicate with the memory card using the micro-SD card modules by write or read functions. This SD card module interfaces in the SPI protocol. We need an SD library to use these SD card modules with Arduino. By default, the SD library will be already installed in the Arduino application.
Some of the important SD Module library commands:

- **SD. begin(#spin):** This command initializes the SD library and micro-SD card. Take the function's argument as the pin connected to the SS pin.
- **SD. exists(filename):** This command tests whether a file or directory exists on the SD card.
- **SD. open (filepath, mode):** This command opens a file on the SD card in read or write mode. By default, the file will open in read mode if you do specify any mode. If a file is opened in write mode and if it doesn't already exist, then with this name a file will be created.
- **file. close ()**: This closes the file. It also makes sure that the data written in the file is physically stored on the SD card.

3.6.3 DHT-11 SENSOR(MOISTURE)

- The percentage of water, in the form of vapors, present in the surrounding air, is called "humidity".
- Humidity can play an important role in the rusting process which may create discomfort for humans, in a room, ICU, etc. The presence of excessive humidity, results in, what is popularly known as "seasickness".
- In case we are dealing with chemicals, the presence of moisture may start, stop a chemical reaction, which was otherwise expected to happen. Thus humidity becomes an important parameter to judge, while performing operations, in ICU.

- Most common types of humidity:
  - relative humidity (RH)
  - dew/frost point
  - ppm (parts per million)



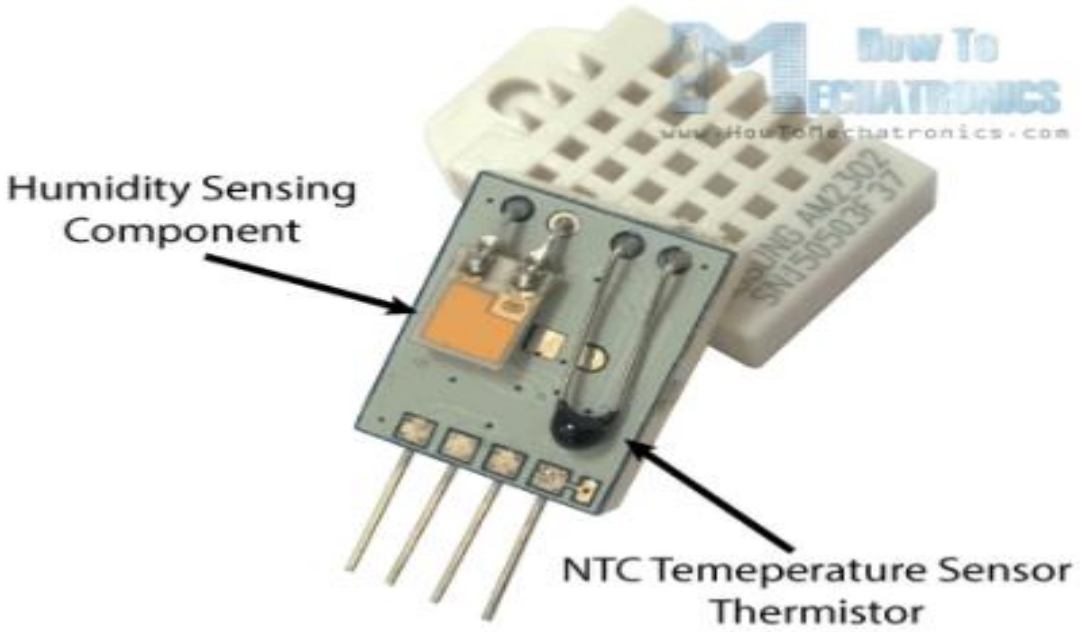
DHT11 Technical Specifications:

Humidity Range: 20-90% RH
Humidity Accuracy: ±5%RH
Temperature Range: 0-50 °C
Temperature Accuracy: ±2% °C
Operating Voltage: 3V to 5.5V

- RH depends on temperature, thus a relative measure of humidity, whereas other 2 are absolute humidity measurements.
- The humidity sensing component of the DHT11 is a moisture holding substrate with the electrodes applied to the surface. When water vapor is absorbed by the substrate, ions are released by the substrate which increases the conductivity between the electrodes. The change in resistance between the two electrodes is proportional to the relative humidity. Higher relative humidity decreases the resistance between the electrodes while lower relative humidity increases the resistance between the electrodes.

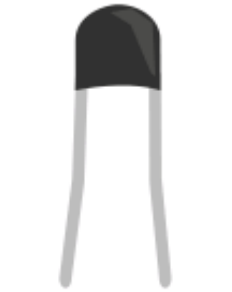
3.6.4 DHT-11 SENSOR(TEMPERATURE)

	DHT11
Operating Voltage	3 to 5V
Max Operating Current	2.5mA max
Humidity Range	20-80% / 5%
Temperature Range	0-50°C / ± 2°C
Sampling Rate	1 Hz (reading every second)
Body size	15.5mm x 12mm x 5.5mm
Advantage	Ultra low cost

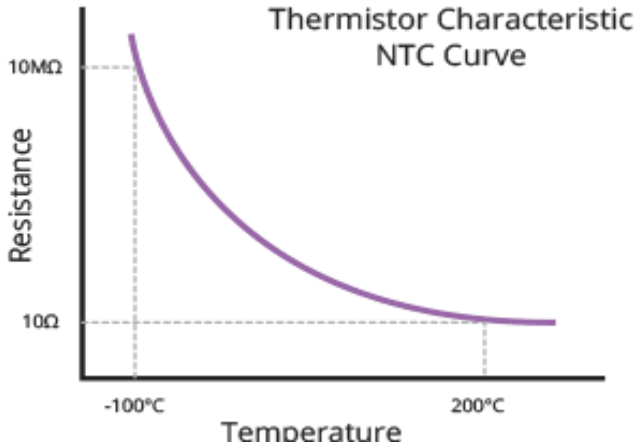


- NTC (negative temperature coefficient) thermistor is used to measure the temperature.
- A thermistor is a temperature-sensitive resistor, which is highly sensitive to temperature change.
- example of its sensitivity: a single degree change, will induce resistance change of 100 ohms and more.
- NTC means, resistance decreases with an increase in temperature.

- There is a small PCB block, with an IC. It processes the analog signal, does the calculation with the stored calibration coefficients, and finally transmits the digital signal.



NTC Thermistor



3.7 SOURCE CODE:

```
#define BLYNK_PRINT Serial
#include <ESP8266_Lib.h>
#include <BlynkSimpleShieldEsp8266.h>

#include <SD.h>                                          // SD card
#include <SPI.h>
File sdcard_file;
int CS_pin = 53;

#include "DHT.h"                                         // DHT11 sensor
#define DHTPIN 8                                         // digital pin of DHT11
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

#define co2Zero 81                                        // In fresh air analog value
#define anInput     A9                                    //analog feed from MQ135


const float m= -3.720;                                    // slope of the line
#define R0 11.820                                         // Resistance of the resistor when exposed to fresh air
#define con 1.111
#define ch4gas_sensor A0                                  // Analog feed from MQ4

#define BLYNK_GREEN "#23C48E"
#define BLYNK_RED "#D3435C"
#define BLYNK_YELLOW "#ED9D00"
#define BLYNK_BLUE "#04C0F8"

char auth[] = "qBV-KaSrDD18huI_BJaG2GSOc8EEfIAO";
char ssid[] = "****";                                      // Name of WiFi network
char pass[] = "********";                                  // Password

float matrix[][2]={{40,65},{27,35}};                       // "1" is for moisture.."0" is for temp
int counterr=0;
int checkpoint=1;


#define EspSerial Serial1

#define ESP8266_BAUD 115200

ESP8266 wifi(&EspSerial);

BlynkTimer timer;

WidgetLED led1(V5);

void myTimerEvent()
{
  delay(2000);
  led1.setValue(255);

  ///// Ch4 concentration calculation - MQ4 sensor /////

  float sensor_volt=0.0;                                   //Define variable for sensor voltage
  float RS_air=0.0;                                        //Define variable for sensor resistance
  double ratio=0.0;
  double ppm_log=0.0;
  float ch4ppm=0.0;
```

```
float analog_value=0;                                   //Define variable for analog readings
for (int x = 0 ; x < 500 ; x++)                         //Start for loop
{
  analog_value = analog_value + analogRead(ch4gas_sensor);      //Add analog values of sensor 500 times
}
analog_value = analog_value / 500.0;                    //Take average of readings
analog_value = analog_value + analogRead(ch4gas_sensor);
sensor_volt = analog_value * (5.0 / 1023.0);            //Convert average to voltage
RS_air = ((5.0 * 10) / sensor_volt) - 10;               //Calculate RS in fresh air
ratio = RS_air/R0;
ppm_log = (log10(ratio)-con)/m;                         //Get ppm value in linear scale according to the the ratio va
ch4ppm = pow(10, ppm_log);                              //Convert ppm value to log scale

Blynk.virtualWrite(V6, ch4ppm);                         //writing to Blynk

///// Representing Ch4 ppm values through gauge meter /////

float k=ch4ppm;
if(k<2.7)
  {Blynk.setProperty(V6,"color",BLYNK_GREEN);}
else if(k<2.9)
  {Blynk.setProperty(V6,"color",BLYNK_YELLOW);}
else
  {Blynk.setProperty(V6,"color",BLYNK_RED);}

/////Temperature and Humidity Calculation - DHT11 sensor /////

float h = dht.readHumidity();

///// Representing temperature and humidity through Gauge meter /////

if(checkpoint==1)
{ /////for moisture/////
   if(counterr==0)
   {
      //changing label
      Blynk.setProperty(V7, "label", "DHT-11 MOISTURE SENSOR(IN %)");
      //changing range
      Blynk.setProperty(V7, "min", 0);
      Blynk.setProperty(V7, "max", 100);

   }
   float reading=h;// todo
   counterr=counterr+1;
   if(counterr>10)
    {checkpoint=0;counterr=0;}

   Blynk.virtualWrite(V7,reading);                      // Writing value
   if(reading<matrix[checkpoint][0])                    // changing color
     {Blynk.setProperty(V7, "color", "#23C48E");}
   else if(reading<matrix[checkpoint][1])
     {Blynk.setProperty(V7, "color", "#ED9D00");}
   else
     {Blynk.setProperty(V7, "color", "#D3435C");}
  }
```

```
else
{ ///// for temp /////
  if(counterr==0)
  {//changing label
    Blynk.setProperty(V7, "label", "DHT-11 TEMP SENSOR(IN CELCIUS)");
    //changing range
    Blynk.setProperty(V7, "min", -10);
    Blynk.setProperty(V7, "max", 70);
  }
  float reading=t;// todo
  counterr=counterr+1;
  if(counterr>10)
    {checkpoint=1;counterr=0;}

  //writing value
  Blynk.virtualWrite(V7,reading);
  //changing color
  if(reading<matrix[checkpoint][0])
    {Blynk.setProperty(V7, "color", "#23C48E");}
  else if(reading<matrix[checkpoint][1])
    {Blynk.setProperty(V7, "color", "#ED9D00");}
  else
    {Blynk.setProperty(V7, "color", "#D3435C");}
}

///// CO2 concentration calculation - MQ135 /////

int co2now[10];                               //int array for co2 readings
int co2raw = 0;                               //int for raw value of co2
int co2comp = 0;                              //int for compensated co2
int co2ppm = 0;                               //int for calculated ppm
int avg = 0;                                  //int for averaging

for (int x = 0;x<10;x++)
{                                             //samplpe co2 10x over 2 seconds
  co2now[x]=analogRead(A9);
  delay(200);
}

for (int x = 0;x<10;x++){                     //add samples together
  avg=avg + co2now[x];

}
co2raw = avg/10;                            //divide samples by 10
co2comp = co2raw - co2Zero;                 //get compensated value
co2ppm = map(co2comp,0,1023,400,5000);      //map value for atmospheric levels
Blynk.virtualWrite(V8,co2ppm);

///// Representing CO2 conc through Gauge meter /////

if(co2ppm<400)
  {Blynk.setProperty(V8,"color",BLYNK_BLUE);}
else if(co2ppm<450)
  {Blynk.setProperty(V8,"color",BLYNK_GREEN);}
else if(co2ppm<1000)
  {Blynk.setProperty(V8,"color",BLYNK_YELLOW);}
else
  {Blynk.setProperty(V8,"color",BLYNK_RED);}
```

```
  ///// Writing to micro SD card /////

  Blynk.setProperty(V5,"color",BLYNK_RED);
  led1.setValue(64);

  if (sdcard_file) {                                          //If the file is found
   Serial.println("Writing to file is under process");
   sdcard_file.println();
   sdcard_file.println(ch4ppm);
   sdcard_file.print("\t\t");
   sdcard_file.print(co2ppm);
   sdcard_file.print("\t\t");
   sdcard_file.print(t);
   sdcard_file.print("\t\t");
   sdcard_file.print(h);
   sdcard_file.println();
   sdcard_file.flush();                                        // Writing data to the SD card physically
   led1.off();
   }
  else {
   Serial.println("Failed to open the file");
    led1.setValue(255);
   }
}


void setup()
{
                                                              // Debug console
  Serial.begin(115200);
  pinMode(anInput,INPUT);
  EspSerial.begin(ESP8266_BAUD);                              // Set ESP8266 baud rate
  delay(10);
   pinMode(CS_pin, OUTPUT);                                   //declaring CS pin as output pin
   if (SD.begin())
   {
   Serial.println("SD card is initialized and it is ready to use");
   } else
   {
   Serial.println("SD card is not initialized");
   }

  Blynk.begin(auth, wifi, ssid, pass);

  sdcard_file = SD.open("data.txt", FILE_WRITE);              //Looking for the data.txt in SD card
  if (sdcard_file) { //If the file is found
   Serial.println("Writing to file is under process");
   sdcard_file.println();
   sdcard_file.println("Methane conc(ppm)-----CO2 conc(ppm)------Temperature (deg C)------Humidity"); //Writing to file
   sdcard_file.flush();
  }
  else {
   Serial.println("Failed to open the file");
   Blynk.setProperty(V5, "color",BLYNK_RED);
   }

  // for MQ4 SENSOR
```

```
  Blynk.setProperty(V6, "label", "MQ4-CH4 SENSOR (in ppm)");
  Blynk.setProperty(V6, "min", 0);
  Blynk.setProperty(V6, "max", 4);

  // for MQ135 SENSOR

  Blynk.setProperty(V8, "label", "MQ135-CO2 SENSOR (in ppm)");
  Blynk.setProperty(V8, "min", 0);
  Blynk.setProperty(V8, "max", 2000);

  timer.setInterval(1000L, myTimerEvent);
}

void loop()
{
  Blynk.run();
  timer.run();                              // Initiates BlynkTimer

}
```

CHAPTER 4

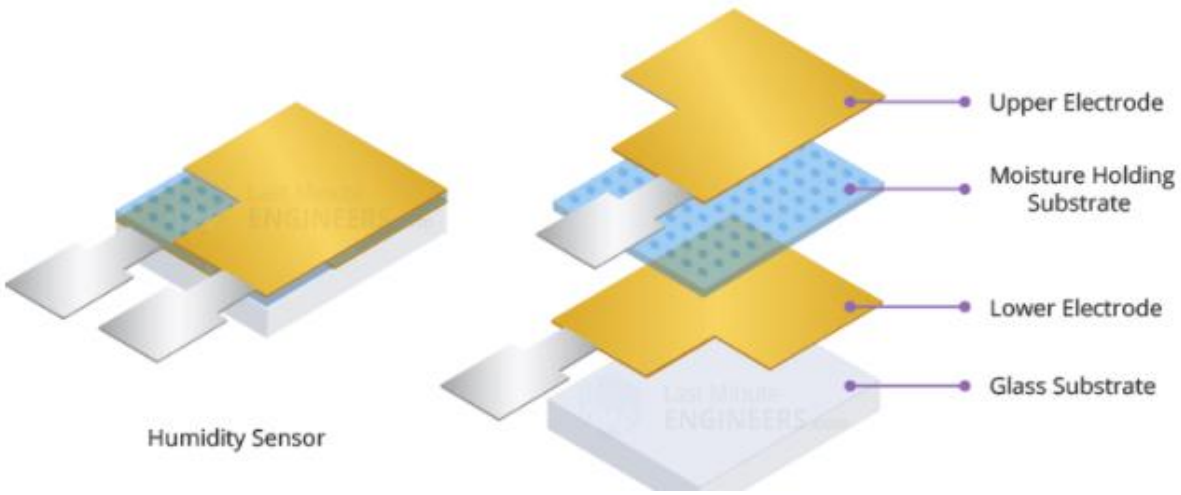
IMPLEMENTATION (DETAILED EXPLANATION ABOUT WORKING PROCESS):

4.1 ROLE OF GAS SENSOR MQ4:

- The internal resistance of the sensor changes when exposed to a rotten or overripe sample. The change is proportional to the concentration of methane produced by the sample.
- Accordingly, the raw data also changes. This can be taken as analog output and converted to the corresponding voltage value.
- This in turn is converted to ppm through the calibration process.

4.2 ROLE OF DHT-11 TEMPERATURE-MOISTURE SENSOR

- The temperature sensor uses a thermistor of NTC (negative temperature coefficient) type (resistance decreases with increase in temperature).
- Thermistors are resistors, which are highly sensitive to temperature change.
- The science behind moisture sensor, can be understood from this diagram:
- Higher relative humidity decreases resistance, between electrodes, and the vice-versa in case of lower relative humidity.



4.3 ROLE OF SD CARD:

- It acts as an offline Database, to store the sensor data.
- Its role is crucial, in investigations, to any fire accident, which may hamper, internet access to the hardware.
- The data stored can be reliably used for “Data analysis”, to understand the dynamics of the warehouse.
- It is timely formatted, through the code, and its data entries are time stamped, to increase the uniqueness of each reading.

4.4 IOT INTEGRATION (ROLE OF PROGRAMMED HARDWARE):

- ESP8266 WIFI Module implements the IP Layer of the OSI stack and has a NIC (network interface card) implemented in its hardware. It links the board to the internet.
- The Arduino board is pushed with a code that is responsible to start a dedicated connection to BLYNK SERVER.
- The frequency at which data will be transmitted to the server is decided by the timers, set through code.
- The hardware code programs the board to store data into an SD card, and in case an internet connection is available, then it should create a dedicated connection to the Blynk server.
- Each sensor data will be written in which the pin while sending data to the server, is decided by the programmer.
- The program is also responsible for receiving data from the server (in case the remote user, pushed data to a pin).
- Exception Handling, must be done, to catch and sort out the errors.
- The label colour of Blynk widgets can be controlled through code.

4.5 IoT INTEGRATION (ROLE OF BLYNK SERVER):

- ESP8266 regularly checks for available internet connection.
- When connected to the internet, it starts sending the sensor readings, to the BLYNK server, in dedicated virtual pins.
- The app with the user needs not to be online (connected with the internet).
- When a remote user connects the Blynk app to the internet, his IP address is sent to the Blynk server, along with the unique “project authentication token”.
- When the hardware (Arduino) sends data to the Blynk server, it uses the same unique “project authentication code”.
- Blynk server, after receiving the data, from Arduino hardware, checks if any remote online user is opening a project, with the same “project authentication

code". In that case, it starts sending the live data, after creating a secure HTTP connection.

- In case no remote user is available, the server simply drops the data, without returning any error.

4.6 FRONT-END WITH BLYNK:

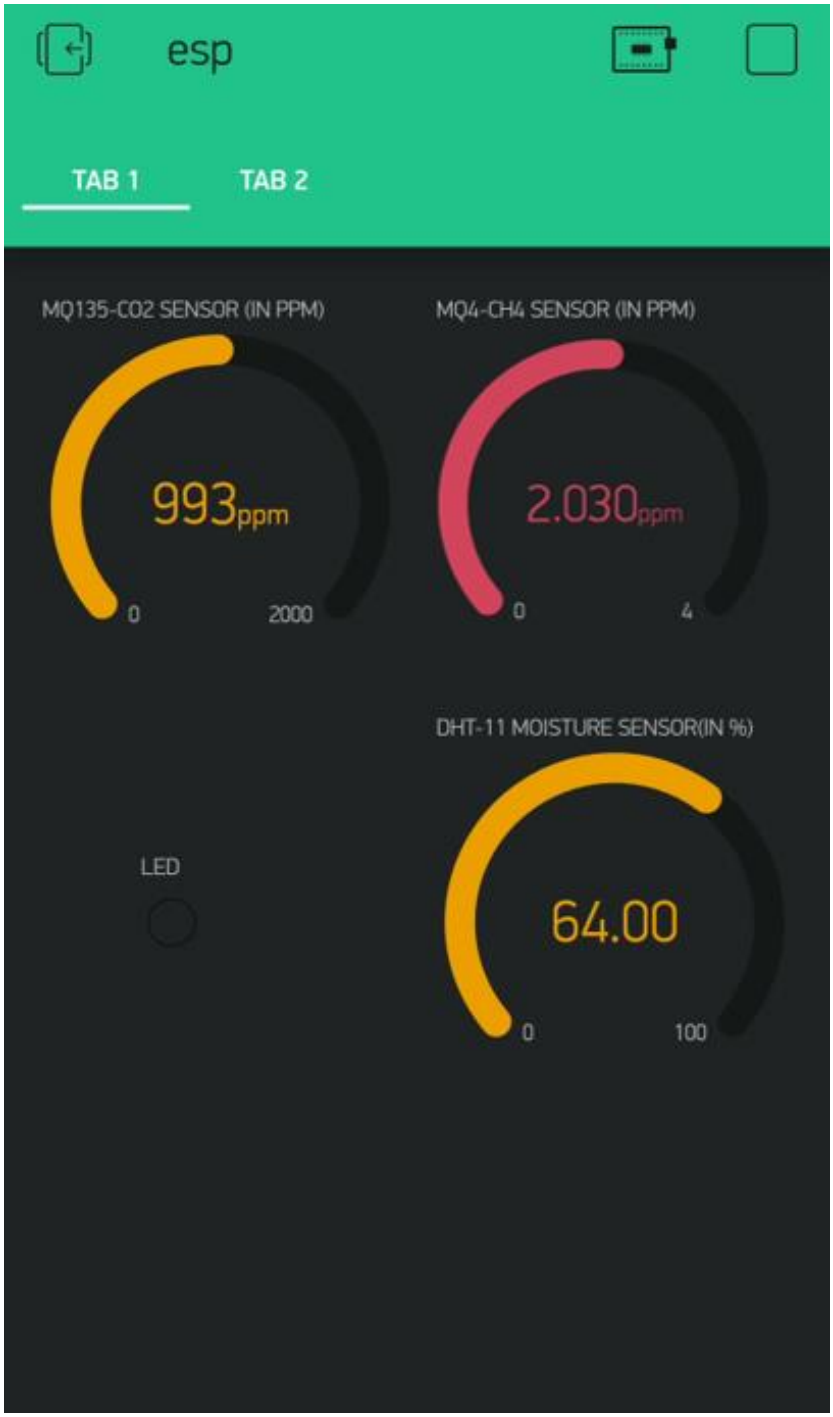
- Blynk provides several free widgets, to display sensor data.
- Each project created has a unique "project authentication code".
- Each project provides access to manipulate/read a limited number of pins.
- The category of pins in Blynk projects includes analog, digital, virtual.
- Each widget, while its induction into the project, is assigned a pin/(pins).
- A typical widget can read the data from an assigned pin, and display it, based on the accuracy decided by the app settings, with animations.
- Widgets with the ability to push data into pins, also exist, for example, a switch widget, with 2 states as - ON, OFF.

CHAPTER 5
RESULTS

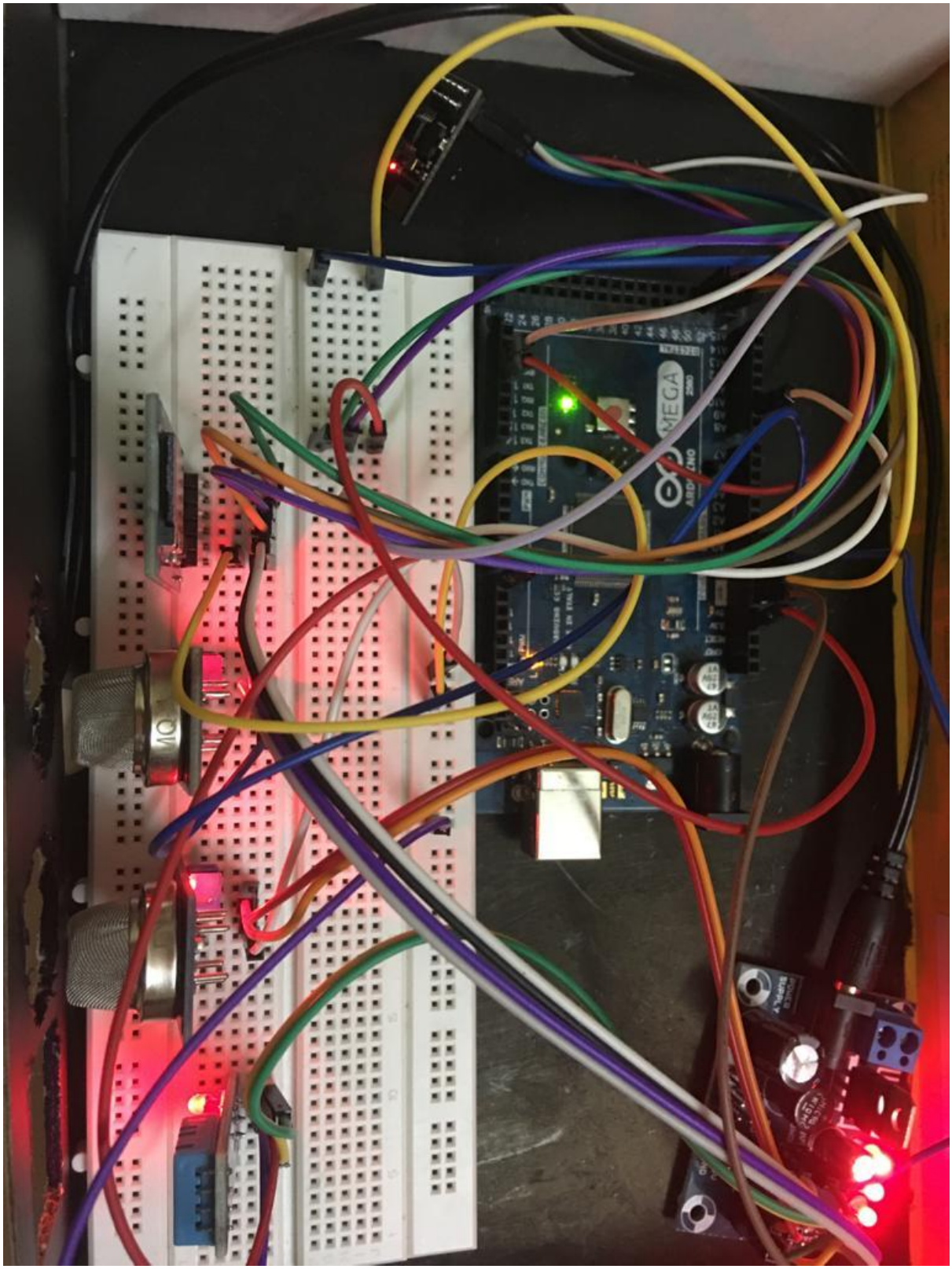
RESULTS (OUTPUT SCREENSHOTS FROM BLYNK APP)

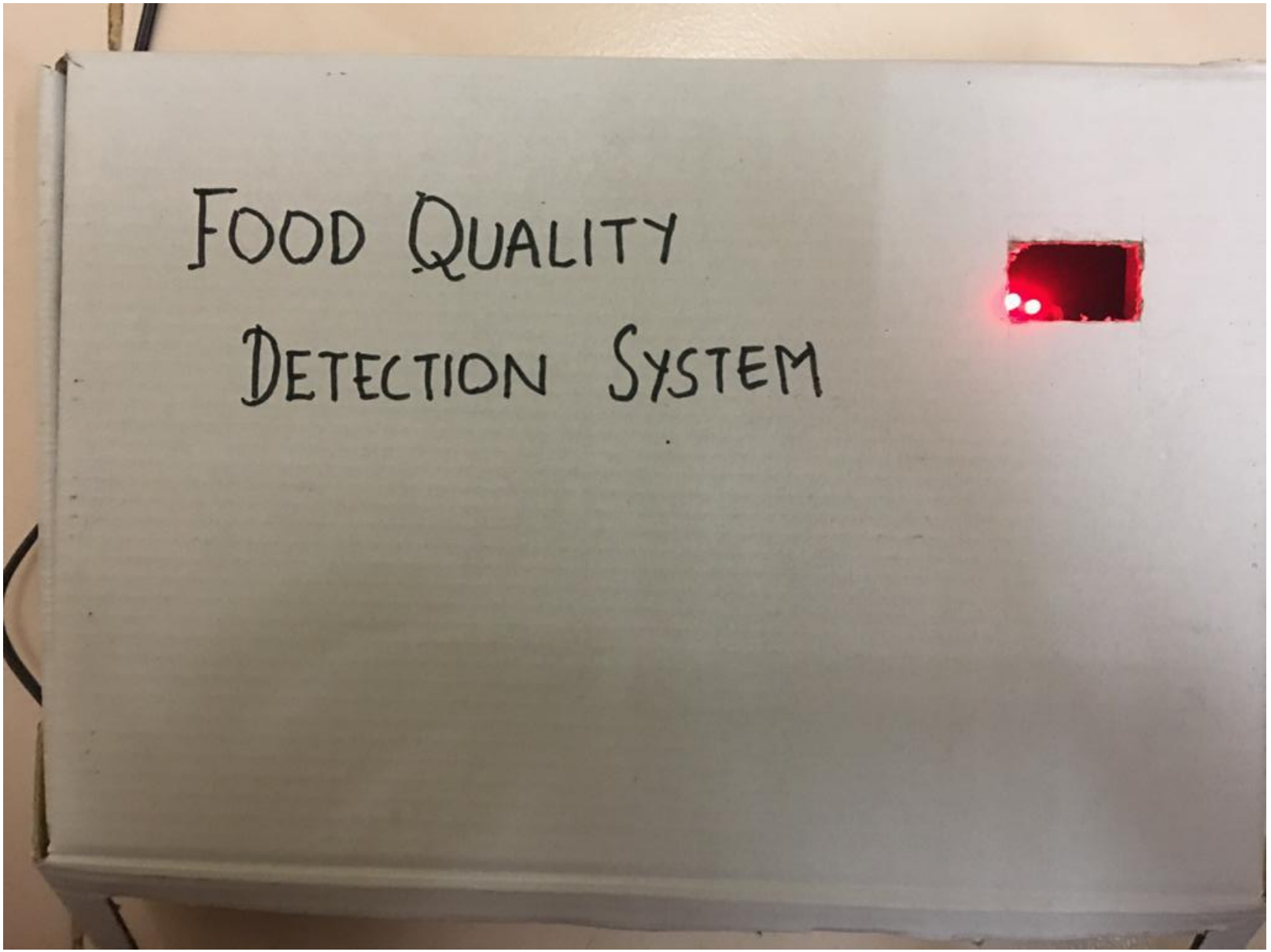
NORMAL CONDITIONS, VALUES IN FRESH AIR

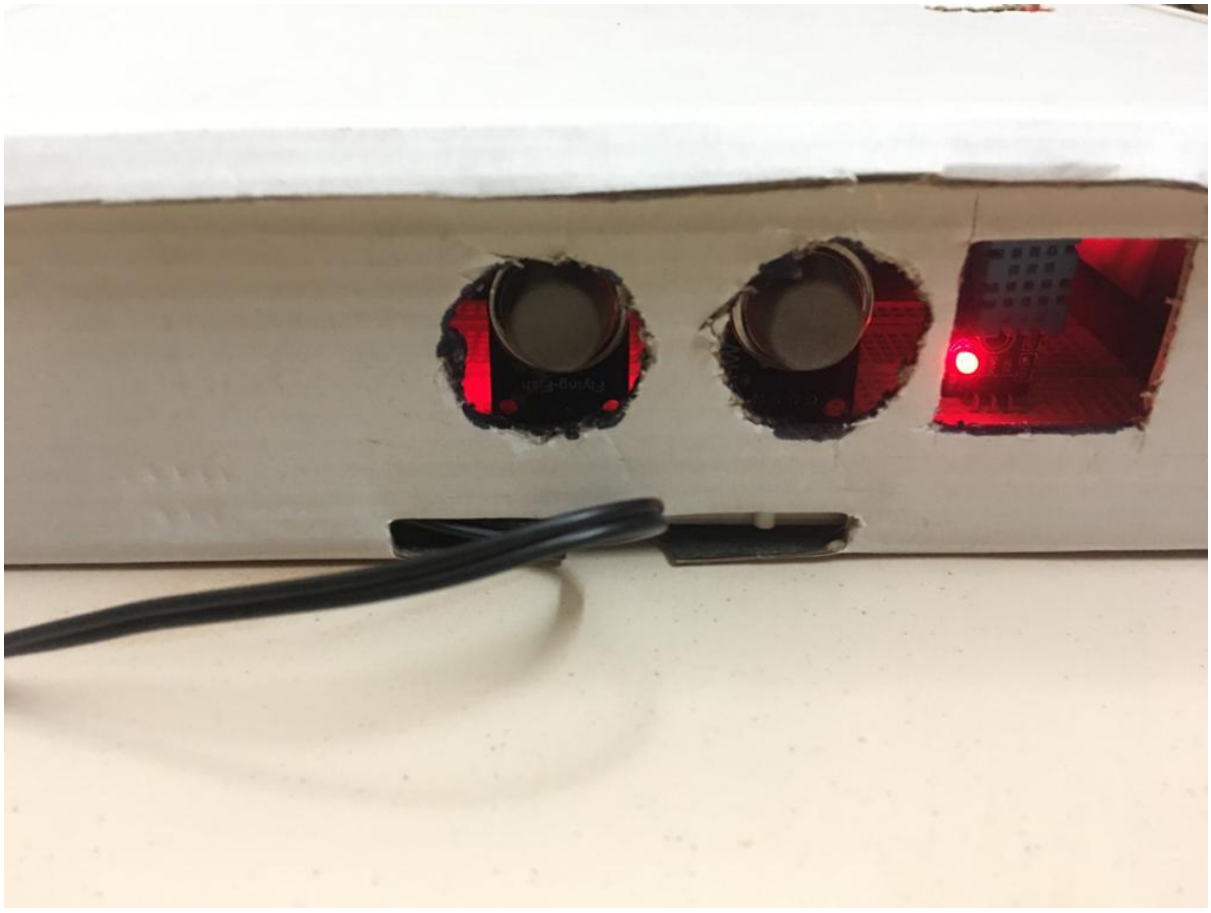
WHEN IT WAS EXPOSED TO THE ROTTEN FRUIT (BEGINNING)



THE VALUES INDICATING THE DECAYED STATE


MEGA
ARDUINO


FOOD QUALITY
DETECTION SYSTEM



CHAPTER 6
CONCLUSION AND FUTURE WORKS

The aim of this project is to create a Food Quality Detection system. IoT devices have become one of the most upbeat trends present and more the IoT devices used, more automation, and better control over objects. IoT has helped in providing more new concepts which result in making our surroundings smarter. Our prototype can detect rotting and decay using input from various sensors like MQ4, MQ135, and DHT 11. The micro-SD card provides backup for data in case of a power failure. The entire setup is connected to the internet and the data is sent to the Blynk server for analysis. The real-life implementation may be challenging and difficult but it may yield optimal results.

FUTURE POSSIBILITIES WITH OUR PROTOTYPE:

- It can be integrated with SMART ENERGY GRID, to provide, round a clock power supply, and save appliances, from voltage variation.
- It can be integrated with SMART ROBOTIC LOADING-UNLOADING.
- It can be integrated with a SMART AIR FILTRATION system, which provides filtration based on the pollutant detected by the sensor.
- The IoT setup can be extended to track the vehicles, carrying the load for the cold storage, and it can give an alert to workers, to keep the warehouse ready, to receive new inputs.
- Additional sensors to detect the moisture in the air can be integrated into the prototype, to provide a complete solution to preserve/protect edibles being stored.
- To overcome the limitations of Blynk, the other free alternatives should be looked at. A suggestion will be to use the “telegram app bot”, because of its robust, powerful, super-easy API.
- Elaborating on the above point, “/game” can be used on telegram, which provides the developer to run an HTML-5 interactive environment, which can be used to show data, visualize data, and thus achieve what Blynk does.
- Only the users, currently meant to receive sensor data, can be included in a private group, where the bot is a member.
- Full control remains with the administrator, and security features of the market best, can obviously, can't be questioned.

CHAPTER 7

REFERENCES

1. Mohsin, A., & Yellampalli, S. S. (2017). *IoT-based cold chain logistics monitoring. 2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI).* doi:10.1109/icpcsi.2017.8392059

2. Karim, A. B., Hassan, A. Z., & Akanda, M. M. (2018). Monitoring food storage humidity and temperature data using IoT. *MOJ Food Process Technol*, *6*(4), 400-404.

3. Mr. Deepak V, Ms. Megha Tatti, Ms. Prithvi, G Hardikar, Mr. Syed Saqlain Ahmed "Cold Storage Management System For Farmers Using IoT (Internet Of Things) Technology" http://www.kscst.iisc.ernet.in/spp/41_series/40S_awarded_&_selected_projs_further_devpt/40S_BE_1424.pdf

4.V.Mythili; "Embedded based food quality detection with biosensor technology" control, communication and computing technologies (ICACCCT), IEEE Conference Publications 2016 page;333-336.

5.ZHIBO PANG, QIANG CHEN, WEILI HAN, LIRONG ZHENG. 2012. "Value-Centric Design of the Internet-of-Things Solutions for Food Supply Chain".

6.ZHIBO PANG, JUN CHEN, ZHI ZHANG, QUIANG CHEN, LIRONG ZHENG. 2009. "Global Fresh Food Tracking Service Enabled by Wide Area Wireless Sensor Network".

7.CN. VERDOUW, J. WOLFERT, A.J.M. BEULUNS, A. RIALLAND. 2015. "Virtualization of Food Supply Chains with the Internet of Things".