\Date: 21/06/25

**Lab 01**

**Aim and Objective:**

Use of sensors to detect the temperature/humidity in a room and having appropriate actions

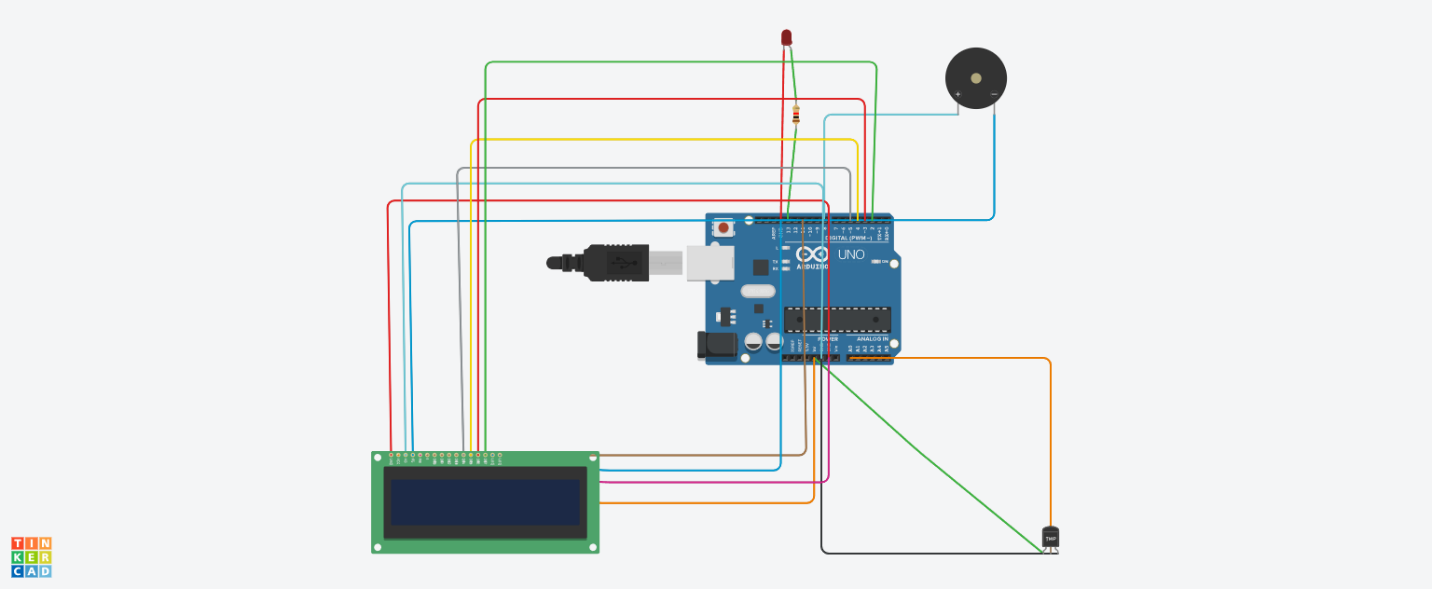
performed such as changing the LED color and turning the speaker on as an alarm and using

serial monitor to see these values.

**Problem Statement:**

Design and implement a real-time room monitoring system using an Arduino, capable of detecting temperature and humidity levels with the help of a temperature sensor. The system should respond to different temperature thresholds by changing the color of LEDs. Additionally, the system must display real-time sensor readings on the Serial Monitor for observation.

**System Design:**

****

**Code:**

**#include** <LiquidCrystal.h>

const int sensorPin = A0;

const int ledPin = 13;

const int buzzerPin = 8;

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {

pinMode(ledPin, OUTPUT);

pinMode(buzzerPin, OUTPUT);

Serial.begin(9600);

lcd.begin(16, 2);

}

void loop() {

int value = analogRead(sensorPin);

float voltage = value \* (5.0 / 1023.0);

float temperatureC = (voltage - 0.5) \* 100;

lcd.setCursor(0, 0);

lcd.print("Temp: ");

lcd.print(temperatureC);

lcd.print(" C");

if (temperatureC > 30) {

digitalWrite(ledPin, HIGH);

digitalWrite(buzzerPin, HIGH);

lcd.setCursor(0, 1);

lcd.print("!! ALERT HOT !!");

} else {

digitalWrite(ledPin, LOW);

digitalWrite(buzzerPin, LOW);

lcd.setCursor(0, 1);

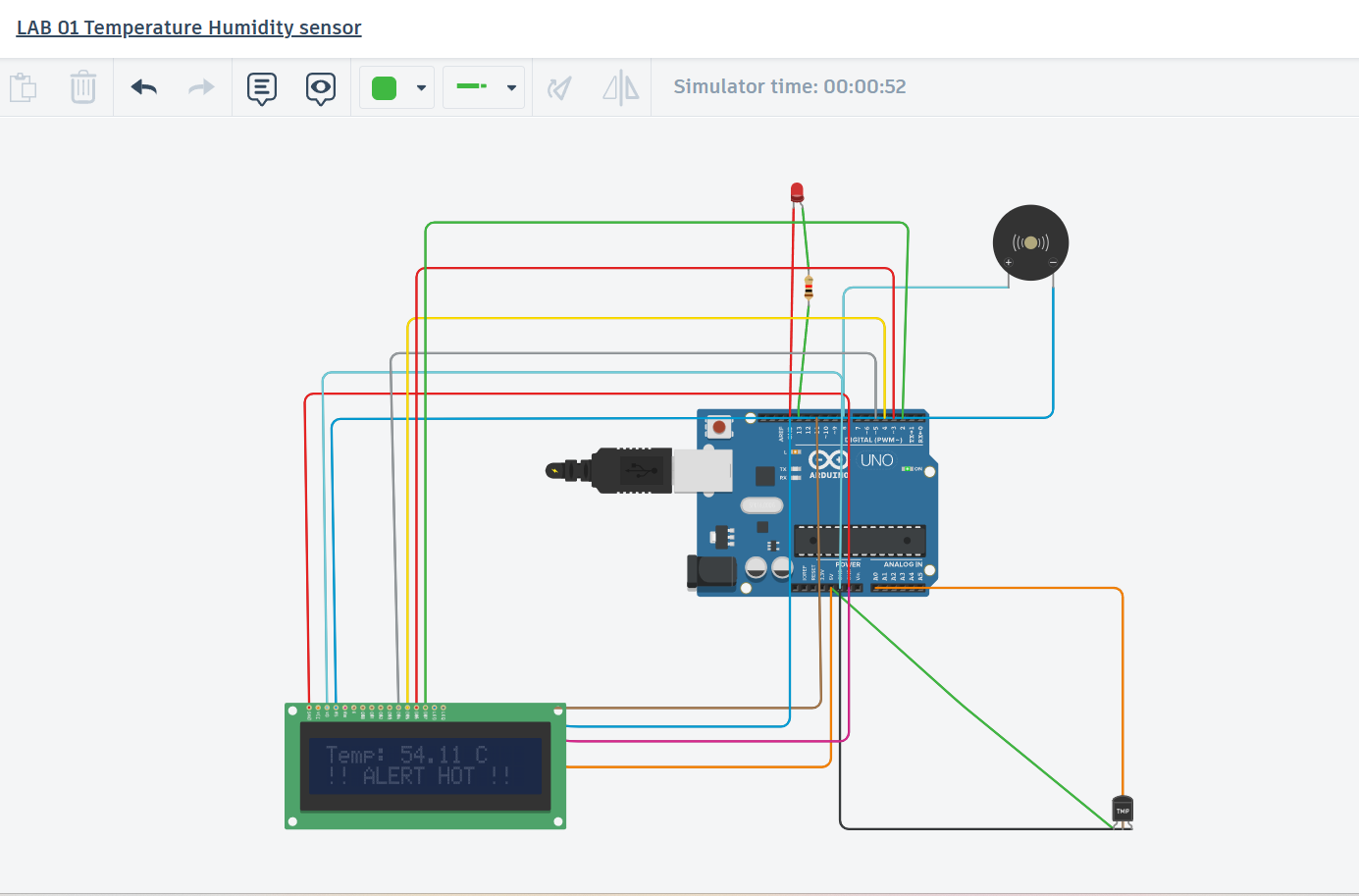
lcd.print("Normal ");

}

delay(1000);

}

**Sample Output:**

****

**Challenges:**

* Connected cathode of LED to (13) digital pin and anode to digital pin so LED was not glowing as the temperature was rising 30 degree celcius.
* Faced difficulty in connecting the LCD screen with arduino
* I2C LCD screen was used first but faced difficulty in connecting it therefore used 16 x 2 (I2C) screen
* Many attempts were given to run the code

**Application Design:**

* Smart home systems.
* Weather monitoring stations.
* Room temperature alerts in case of emergency

**Reflection:**

**Through this lab, I learned**

* How the components are connected with Arduino
* Usage of Buzzer
* Usage of code to read and interpret real time data in LCD screen
* How IOT techniques works for different environmental monitoring
* How the temperature sensors are connected and their working

Date:23/06/25

**Lab 02**

**Aim and Objective:**

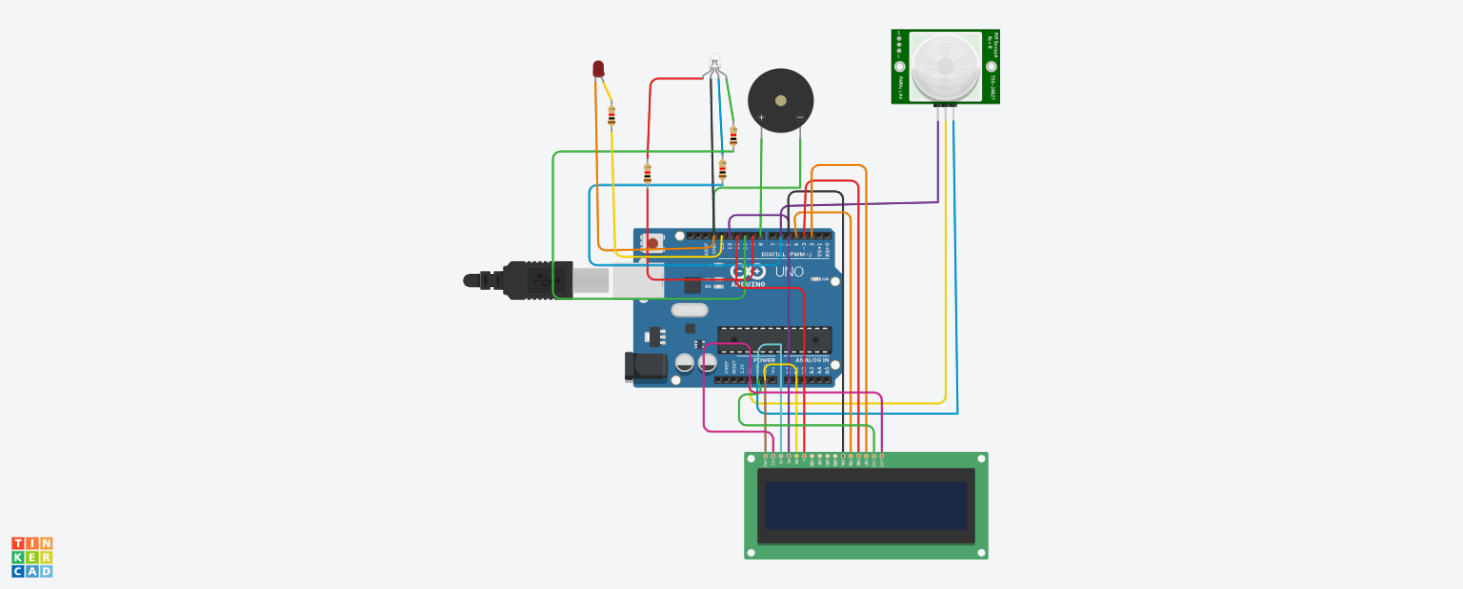
A basic environment monitoring or geo-area monitoring using IR sensors, Ultrasonic Sensors,

LED bulbs, Speakers etc., and perform the alert operations accordingly.

**Problem Statement:**

Design and implement a real-time environment monitoring system using an Arduino, capable of detecting objects with the help of a motion sensor/ultrasonic sensor. The system should respond to motions by changing the color of LEDs. Additionally, the system must display real-time sensor readings on the Serial Monitor for observation.

**System Design:**

****

**Code:**

#include <LiquidCrystal.h>

// LCD pin mapping: RS, E, D4, D5, D6, D7

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

// PIR moved to pin 6 (to free up pin 2 for LCD D7)

const int pirPin = 6;

const int ledPin = 13;

const int buzzerPin = 8;

// RGB LED pins

const int redPin = 9;

const int greenPin = 10;

const int bluePin = 11;

void setup() {

pinMode(pirPin, INPUT);

pinMode(ledPin, OUTPUT);

pinMode(buzzerPin, OUTPUT);

pinMode(redPin, OUTPUT);

pinMode(greenPin, OUTPUT);

pinMode(bluePin, OUTPUT);

lcd.begin(16, 2);

lcd.print("System Ready");

Serial.begin(9600);

delay(2000);

lcd.clear();

}

void loop() {

int motion = digitalRead(pirPin);

if (motion == HIGH) {

Serial.println("Motion Detected!");

digitalWrite(ledPin, HIGH);

digitalWrite(buzzerPin, HIGH);

setRGBColor(255, 0, 0); // RED

lcd.clear();

lcd.setCursor(0, 0);

lcd.print("Motion Detected");

lcd.setCursor(0, 1);

lcd.print("Someone in house");

} else {

Serial.println("No motion.");

digitalWrite(ledPin, LOW);

digitalWrite(buzzerPin, LOW);

setRGBColor(0, 255, 0); // GREEN

lcd.clear();

lcd.setCursor(0, 0);

lcd.print("House is empty");

}

delay(500);

}

void setRGBColor(int red, int green, int blue) {

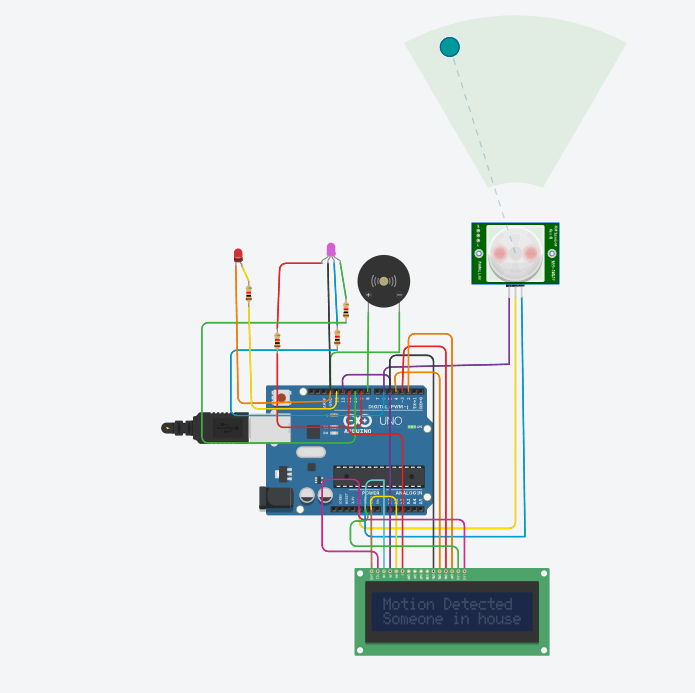
analogWrite(redPin, red);

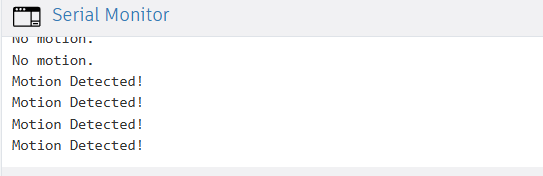
analogWrite(greenPin, green);

analogWrite(bluePin, blue);

}

**Sample Output:**

****

****

**Challenges:**

* The sensor doesn’t auto-detect motion in simulation, so it had to be manually toggled.
* Integrating the LCD required reassigning pins, as pins were already in use

**Application Design:**

This system simulates a basic smart security and surveillance solution**:**

* PIR sensor detects movement (human/presence).
* RGB LED gives status
* Green = Safe
* Red = Alert
* Buzzer provides real-time audible alerts.
* LCD display provides contextual messages for the user (e.g., “Someone in house”, “System Ready”).
* Serial Monitor shows real-time message**.**

**Use cases:**

* Home security systems
* Office or warehouse motion alarms

**Reflection:**

**Through this lab, I learned how to:**

* Integrate multiple components (PIR, RGB LED, buzzer, LCD) into a single cohesive embedded system
* Create meaningful user alerts through both visual (LCD, LED)
* Use digital and analog signals together in real-time.

**Date : 7/07/2025**

**Lab 03**

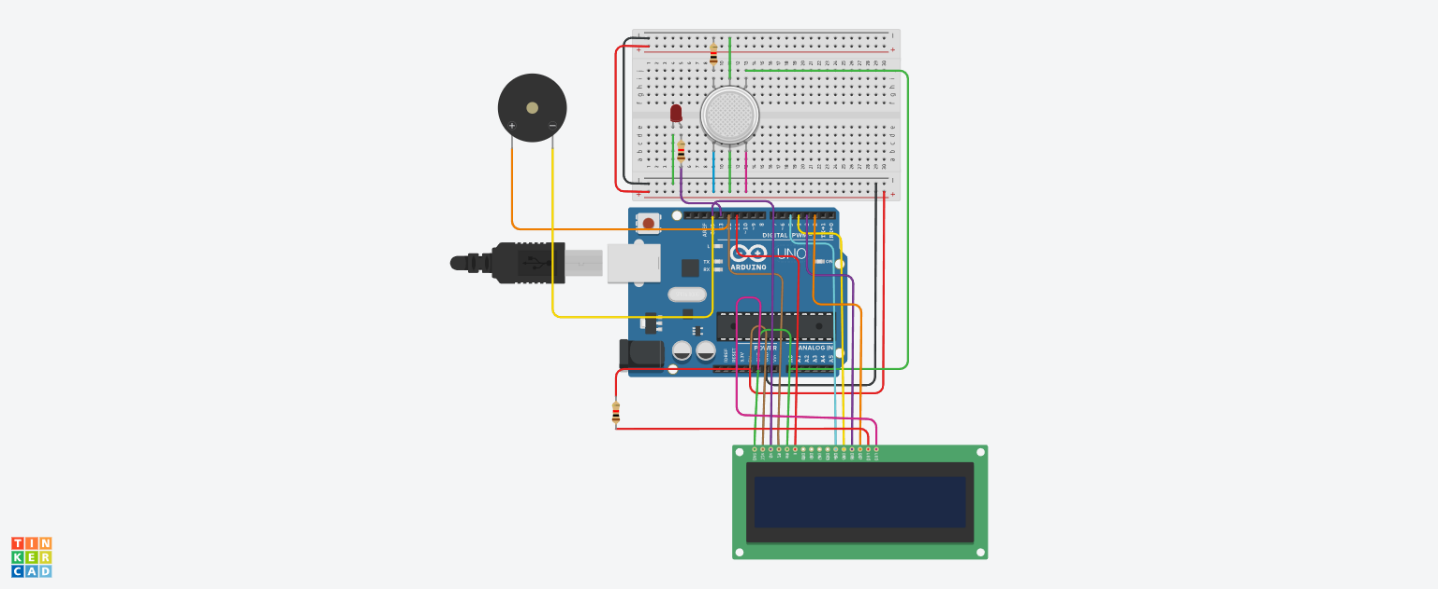
**Aim and Objective:**

Making use of GSM for communication in the obstacle avoiding robot

**Problem Statement:**

Making use of GSM for communication in the obstacle avoiding robot. Using sensors such as flame sensors, PIR human motion sensor, IR sensor, LED bulbs etc for better inputs regarding the environment.

**System Design:**

****

**Code:**

#include <LiquidCrystal.h>

// LCD pins: RS, E, D4, D5, D6, D7

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

int gasSensor = A0;

int ledPin = 13;

int buzzer = 10;

int threshold = 250;

void setup() {

Serial.begin(9600);

pinMode(ledPin, OUTPUT);

pinMode(buzzer, OUTPUT);

lcd.begin(16, 2);

lcd.setCursor(0, 0);

lcd.print("Gas Monitoring");

delay(2000);

lcd.clear();

}

void loop() {

int gasValue = analogRead(gasSensor);

lcd.setCursor(0, 0);

lcd.print("Gas: ");

lcd.print(gasValue);

lcd.print(" ");

if (gasValue >= threshold) {

digitalWrite(ledPin, HIGH);

digitalWrite(buzzer, HIGH);

Serial.print(gasValue);

Serial.println(" |Gas detected|");

lcd.setCursor(0, 1);

lcd.print(" Gas Detected! ");

} else {

digitalWrite(ledPin, LOW);

digitalWrite(buzzer, LOW);

Serial.print("sensorValue: ");

Serial.println(gasValue);

lcd.setCursor(0, 1);

lcd.print("Safe Environment ");

}

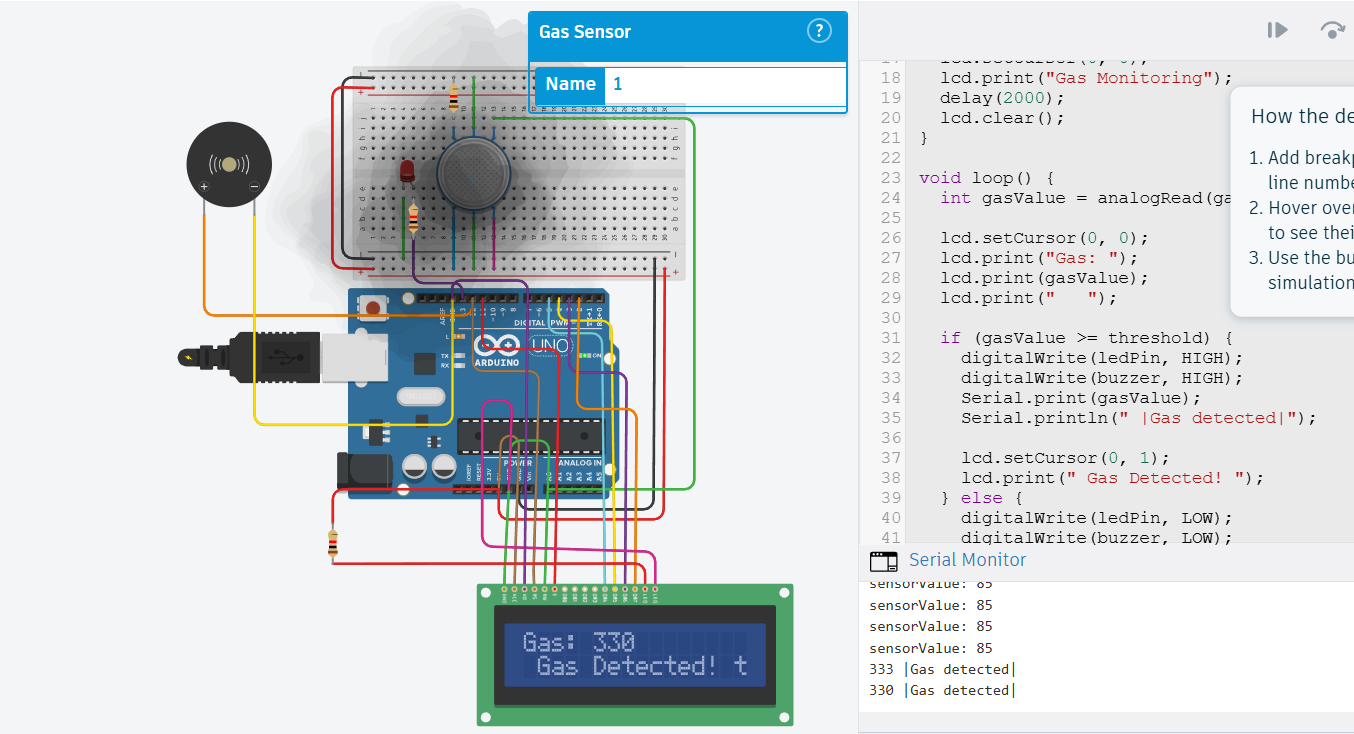
delay(1000);

}

**Challenges:**

* The MQ2 sensor needed careful calibration to avoid false gas detection due to environmental factors (like smoke, dust, alcohol, etc.).
* Buzzer was continuously ringing due to incorrect threshold logic or analog noise from the sensor.
* LED was drawing more than 20 mA — above safe limits — and required a current-limiting resistor to prevent damage.

**Output:**



**Application Design:**

* Detect gas leaks, fire, or human presence and alert homeowners remotely.
* Early detection of hazardous gases and fire in factories or warehouses to prevent accidents.
* Use PIR sensors to monitor unauthorized movement and send immediate alerts via GSM.
* Autonomous robots navigating spaces while sensing obstacles and environmental risks.

**Use Cases:**

**L**

* Security Patrol Robot
* Home Safety Monitoring

**Reflection:**

* How to interface multiple sensors (gas sensor, PIR, IR, flame sensor) with Arduino for environmental monitoring.
* The concept and implementation of obstacle avoidance using IR sensors.
* Using an LCD display to show real-time sensor readings and system status.
* How to set sensor thresholds and program conditional responses (e.g., activating buzzer and LED).
* Basics of GSM communication for sending remote alerts via SMS.
* Integrating sensor data and GSM module to create an autonomous safety alert system.

**Date : 8/08/2025**

**Lab 04**

**Aim and Objective:**

The aim is to build a real-time environmental monitoring system using Arduino and

ESP8266 WiFi module for IoT-based data transmission. The system uses sensors

such as the PIR motion sensor and a temperature sensor (TMP36) to detect human

presence and ambient temperature. The objective is to collect sensor data and send

it to a ThingSpeak cloud server for remote monitoring. This enables smart

surveillance and environmental sensing for applications like security, smart homes,

or basic automation systems.

**Problem Statement:**

Design and implement an IoT-based monitoring system using Arduino, a PIR motion

sensor, and a temperature sensor. The system should detect motion and measure

temperature, then transmit the collected data to a ThingSpeak channel using the

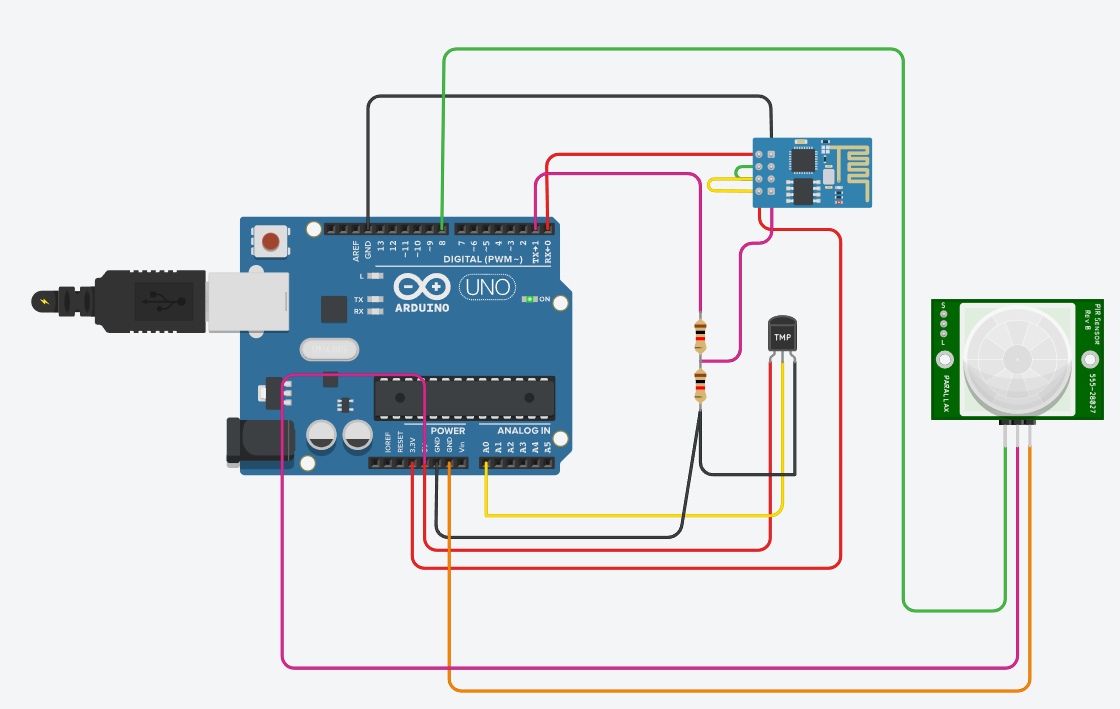
ESP8266 WiFi module. The solution must support continuous real-time data

transmission at regular intervals and provide visibility into environmental conditions

remotely. This model can serve as a foundational unit for smart automation,

occupancy detection, or health/safety monitoring applications.

**System Design:**

****

**Code:**

String ssid = &quot;Simulator Wifi&quot;;

String password = &quot;&quot;;

String host = &quot;api.thingspeak.com&quot;;

const int httpPort = 80;

// API key for your single ThingSpeak channel

String apiKey = &quot U1P21V9YTJ3HO7FG &quot;; // Channel 1

int pirPin = 8; // PIR motion sensor output pin

void sendToThingSpeak(String apiKey, int temp, int motion) {

// Send both fields in one request

String uri = &quot;/update?api\_key=&quot; + apiKey +

&quot;&amp;field1=&quot; + String(temp) +

&quot;&amp;field2=&quot; + String(motion);

String httpPacket = &quot;GET &quot; + uri + &quot; HTTP/1.1\r\nHost: &quot; + host + &quot;\r\n\r\n&quot;;

int length = httpPacket.length();

Serial.print(&quot;AT+CIPSEND=&quot;);

Serial.println(length);

delay(10);

Serial.print(httpPacket);

delay(10);

Serial.find(&quot;SEND OK\r\n&quot;);

}

int setupESP8266(void) {

Serial.begin(115200);

Serial.println(&quot;AT&quot;);

delay(10);

if (!Serial.find(&quot;OK&quot;)) return 1;

Serial.println(&quot;AT+CWJAP=\&quot;&quot; + ssid + &quot;\&quot;,\&quot;&quot; + password + &quot;\&quot;&quot;);

delay(10);

if (!Serial.find(&quot;OK&quot;)) return 2;

Serial.println(&quot;AT+CIPSTART=\&quot;TCP\&quot;,\&quot;&quot; + host + &quot;\&quot;,&quot; + httpPort);

delay(50);

if (!Serial.find(&quot;OK&quot;)) return 3;

return 0;

}

void anydata(void) {

int temp = map(analogRead(A0), 20, 358, -40, 125); // TMP36

int motion = digitalRead(pirPin); // PIR sensor (0 or 1)

sendToThingSpeak(apiKey, temp, motion);

}

void setup() {

pinMode(pirPin, INPUT);

setupESP8266();

}

void loop() {

anydata();

delay(10000);

}

**Sample Output:**

AT

AT+CWJAP=&quot;Simulator Wifi&quot;,&quot;&quot;

AT+CIPSTART=&quot;TCP&quot;,&quot;api.thingspeak.com&quot;,80

AT+CIPSEND=94

GET /update?api\_key=BAGOD0B0B245QWVN&amp;field1=24&amp;field2=0 HTTP/1.1

Host: api.thingspeak.com

AT+CIPSEND=94

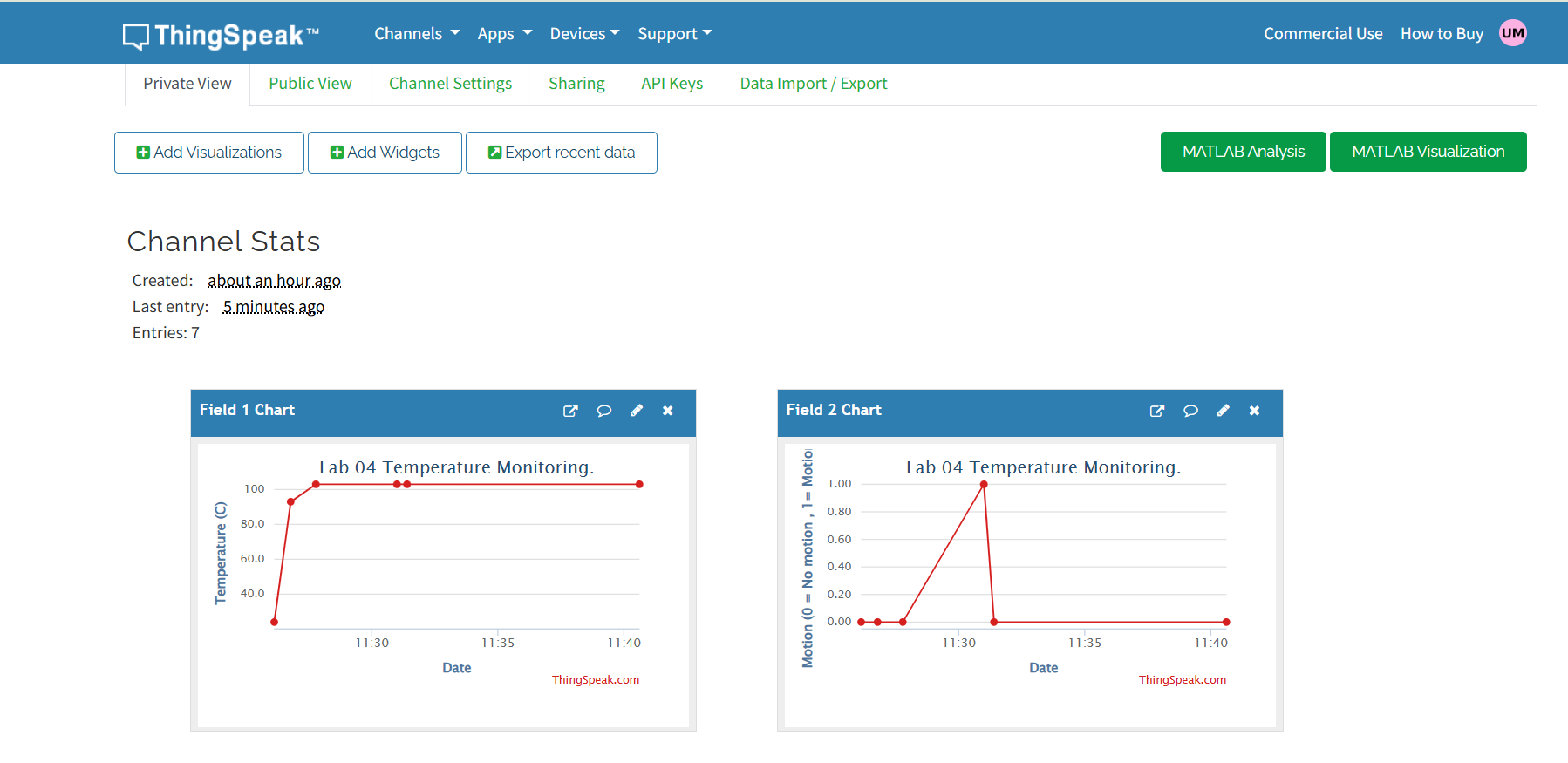
GET /update?api\_key=BAGOD0B0B245QWVN&amp;field1=83&amp;field2=0 HTTP/1.1

Host: api.thingspeak.com

AT+CIPSEND=94

GET /update?api\_key=BAGOD0B0B245QWVN&amp;field1=83&amp;field2=0 HTTP/1.1

Host: api.thingspeak.com

****

**Challenges:**

**1. Sensor Stability:** The temperature sensor readings can fluctuate due to noise;

proper scaling and calibration are required for accurate data.

**2. Motion Detection Sensitivity:** The PIR sensor may generate false positives

due to minor heat movements or environmental noise.

**3. WiFi Dependency:** Since the system relies on the ESP8266 module, it needs

a stable WiFi connection for successful data transmission.

**4. Data Transmission Delay:** Continuous 10-second data upload may not suit

real-time critical systems; needs careful delay management.

**5. Limited Feedback:** Without onboard display or buzzer alerts, all feedback

relies on cloud visualization (ThingSpeak) or Serial Monitor, limiting on-site

awareness.

**Application Design:**

**1. Components Used:**

* Arduino UNO
* ESP8266 WiFi Module
* PIR Motion Sensor (connected to Pin 8)
* TMP36 Temperature Sensor (connected to A0)

**2. Functionality:**

* Reads temperature using TMP36 sensor via analog pin A0.
* Reads motion status (0 or 1) using the PIR sensor.
* Sends both temperature and motion data to a ThingSpeak channel using ESP8266
* Data is sent every 10 seconds for continuous monitoring.
* Serial Monitor is used for sending AT commands and debugging.

**Reflection:**

This project introduced me to basic IoT concepts using Arduino and ESP8266. I

learned how to interface and read data from a PIR motion sensor and TMP36

temperature sensor, and how to upload this data to a cloud platform like ThingSpeak.

The experience deepened my understanding of real-time data monitoring and the

role of wireless communication in IoT systems. Although the system lacks local

alerts like buzzers or displays, it lays a strong foundation for smart home surveillance

or automation applications that require remote sensing and centralized monitoring.

Date: 8/08/25

**Lab 05**

**Aim and Objective:**

The aim of this lab is to develop an IoT-based smart bin monitoring system using

Arduino and the ESP8266 WiFi module. The objective is to measure the fill level of a

dustbin using an ultrasonic sensor and transmit this data to a ThingSpeak cloud

channel for remote observation. By interpreting the distance from the bin&#39;s lid to the

trash surface, the system determines whether the bin is full and triggers an LED alert

accordingly. Through the integration of WiFi-based communication, the system

enables real-time status updates of the bin, facilitating more efficient and intelligent

waste management practices suited for smart city environments.

Problem Statement:

With increasing urbanization, waste collection inefficiencies pose a challenge to city

management. To address this, the system designed in this lab detects the fill level of

a trash bin using an ultrasonic sensor interfaced with Arduino. The system processes

the measured distance to identify whether the bin is full and visually indicates this

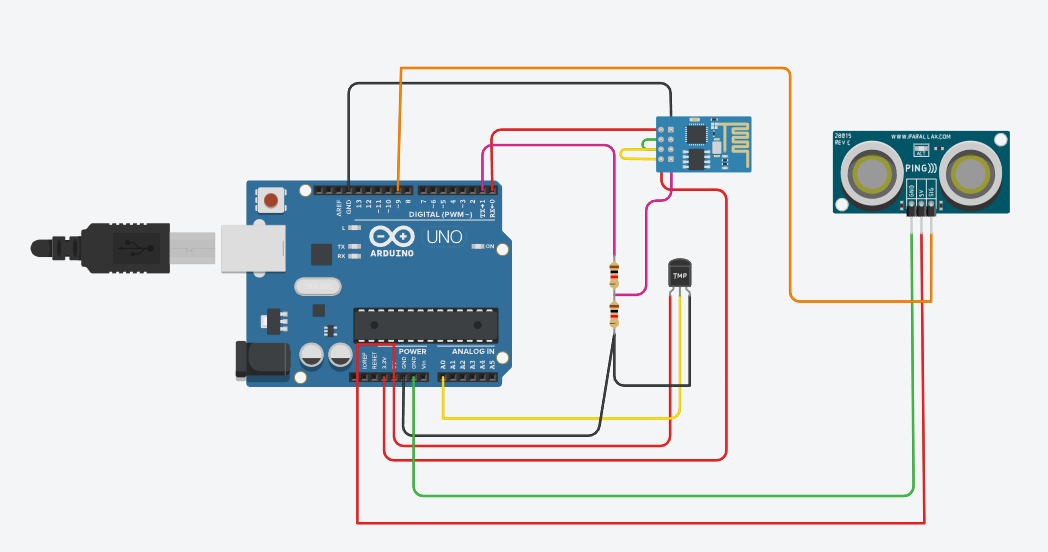
status using an LED. The collected data is sent to the cloud via the ESP8266

module, where it is logged and displayed on ThingSpeak. This setup provides real-

time monitoring of bin status and can be used to optimize garbage collection routes,

ultimately supporting smarter, cleaner urban infrastructure.

**System Design:**

****

**Code:**

// Using hardware Serial for ESP8266 communication just like previous lab

String ssid = &quot;Simulator Wifi&quot;;

String password = &quot;&quot;; // Empty for open network

String host = &quot;api.thingspeak.com&quot;;

const int httpPort = 80;

String apiKey = &quot; U1P21V9YTJ3HO7FG &quot;; // Your ThingSpeak key

const int pingPin = 9; // PING sensor single pin

const int ledPin = 7; // LED pin

void sendToThingSpeak(String apiKey, int distance, int binFull) {

String uri = &quot;/update?api\_key=&quot; + apiKey +

&quot;&amp;field1=&quot; + String(distance) +

&quot;&amp;field2=&quot; + String(binFull);

String httpPacket = &quot;GET &quot; + uri + &quot; HTTP/1.1\r\nHost: &quot; + host + &quot;\r\n\r\n&quot;;

Serial.print(&quot;AT+CIPSEND=&quot;);

Serial.println(httpPacket.length());

delay(100);

if (Serial.find(&quot;&gt;&quot;)) {

Serial.print(httpPacket);

delay(500);

Serial.find(&quot;SEND OK&quot;);

}

}

int setupESP8266() {

Serial.begin(115200); // Hardware Serial at 115200 baud

delay(2000);

Serial.println(&quot;AT&quot;);

delay(2000);

if (!Serial.find(&quot;OK&quot;)) return 1;

Serial.print(&quot;AT+CWJAP=\&quot;&quot;);

Serial.print(ssid);

Serial.print(&quot;\&quot;,\&quot;&quot;);

Serial.print(password);

Serial.println(&quot;\&quot;&quot;);

delay(6000);

if (!Serial.find(&quot;OK&quot;)) return 2;

Serial.println(&quot;AT+CIPSTART=\&quot;TCP\&quot;,\&quot;&quot; + host + &quot;\&quot;,&quot; + httpPort);

delay(2000);

if (!Serial.find(&quot;OK&quot;)) return 3;

return 0;

}

long getDistance() {

long duration, distance;

pinMode(pingPin, OUTPUT);

digitalWrite(pingPin, LOW);

delayMicroseconds(2);

digitalWrite(pingPin, HIGH);

delayMicroseconds(5);

digitalWrite(pingPin, LOW);

pinMode(pingPin, INPUT);

duration = pulseIn(pingPin, HIGH, 30000); // 30ms timeout

if(duration == 0) return -1;

distance = duration \* 0.034 / 2;

return distance;

}

void anydata() {

long distance = getDistance();

int binFull = (distance &gt;= 0 &amp;&amp; distance &lt; 50) ? 1 : 0;

digitalWrite(ledPin, binFull);

Serial.print(&quot;Distance: &quot;);

Serial.print(distance);

Serial.print(&quot; cm, Bin Full: &quot;);

Serial.println(binFull);

sendToThingSpeak(apiKey, distance, binFull);

}

void setup() {

pinMode(ledPin, OUTPUT);

digitalWrite(ledPin, LOW);

// Start Serial for ESP8266 communication and debugging

Serial.begin(115200);

int result = setupESP8266();

if(result != 0) {

Serial.print(&quot;ESP8266 setup failed, error code: &quot;);

Serial.println(result);

} else {

Serial.println(&quot;ESP8266 connected&quot;);

}

}

void loop() {

anydata();

delay(15000); // ThingSpeak rate limit

}

**Sample Output:**

AT

AT+CWJAP=&quot;Simulator Wifi&quot;,&quot;&quot;

AT+CIPSTART=&quot;TCP&quot;,&quot;api.thingspeak.com&quot;,80

ESP8266 connected

Distance: 41 cm, Bin Full: 1

AT+CIPSEND=94

GET /update?api\_key=ME5IE8XXR9W6VCB1&amp;field1=41&amp;field2=1 HTTP/1.1

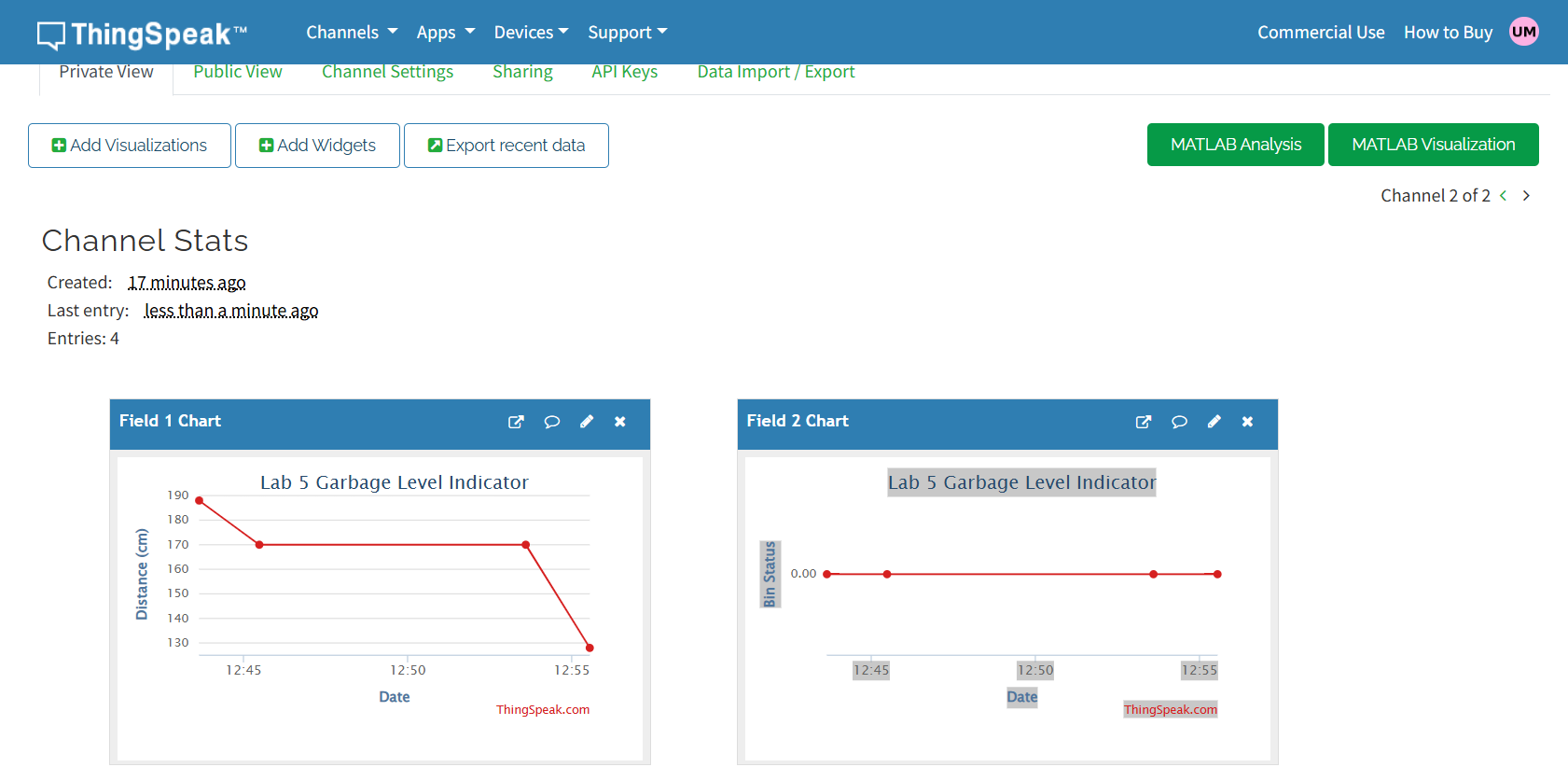
Host: api.thingspeak.com

Distance: 127 cm, Bin Full: 0

AT+CIPSEND=95

GET /update?api\_key=ME5IE8XXR9W6VCB1&amp;field1=127&amp;field2=0 HTTP/1.1

Host: api.thingspeak.com



**Challenges:**

* **Distance Accuracy:** Ultrasonic sensors are sensitive to surface angles and

ambient noise, which can affect distance readings and require filtering or

validation logic.

* **Bin Fill Threshold:** Setting the bin full threshold (e.g., 50 cm) needs calibration

based on bin dimensions and practical use cases**.**

* **ESP8266 Timing Sensitivity**: Communication with the ESP8266 using AT

commands demands precise timing, and improper delays can cause data

transmission failures.

* **Serial Communication Conflicts**: Since the ESP8266 and Serial Monitor share

the same hardware Serial interface, debugging outputs must be managed

carefully to avoid command conflicts.

* **Network Dependence**: Successful data transmission entirely relies on WiFi

availability and strength, making the system dependent on a stable network

connection.

**Application Design:**

**1. Components Used:**

* **Arduino UNO**
* **ESP8266 WiFi Module**
* **Ultrasonic Sensor (HC-SR04 compatible, connected via a single pin at digital 9)**
* **LED (connected to digital pin 7)**
* **USB Cable for Serial Communication**

**2. Functionality:**

The system begins by reading the distance to the trash using an ultrasonic sensor. If

the distance is less than 50 cm, the bin is considered full, and the LED is turned ON

as an alert. If the distance is 50 cm or more, the LED remains OFF. This bin status

along with the current distance is then sent to the ThingSpeak cloud using the

ESP8266 WiFi module. The data is updated every 15 seconds, respecting

ThingSpeak’s rate limits, and printed on the Serial Monitor for live feedback and

debugging purposes.

**1**

**Reflection:**

This lab provided valuable hands-on experience with integrating sensors and

wireless communication in embedded systems. By using the ultrasonic sensor, I

learned how to measure real-world distances and apply logical conditions to

determine physical status, such as whether a bin is full. The challenge of interfacing

the ESP8266 through AT commands over the hardware Serial port deepened my

understanding of timing issues and command sequencing. Sending real-time data to

ThingSpeak introduced me to cloud-based IoT applications and data logging.

Overall, this experiment served as a practical foundation for building smart systems

with real-time environmental awareness and wireless data reporting.

Date: 14/8/25

**Lab 06**

**Aim and Objective:**

To build a real-time data analytics pipeline that processes IoT sensor data, applies windowing techniques, performs multiple aggregation operations, and visualizes the results for monitoring and analysis.

1. To load and preprocess IoT sensor data from the Intel Berkeley Research Lab dataset.
2. To implement time-based windowing (tumbling and sliding windows) for streaming data simulation.
3. To perform at least five aggregation operations (mean, max, min, count, standard deviation) on the streaming data.
4. To store the aggregated results in a CSV file for further analysis.
5. To visualize the aggregated metrics using line plots for better interpretation.

**Problem Statement:**

IoT sensor networks generate continuous streams of data that need to be processed in real time for monitoring and decision-making. Traditional batch processing methods fail to capture time-sensitive changes effectively. This lab aims to simulate a real-time analytics pipeline by using historical IoT data from the Intel Berkeley Research Lab, applying windowing and aggregation operations to derive meaningful insights, and visualizing them for analysis.

**Code:**

import pandas as pd

import matplotlib.pyplot as plt

col\_names = ['date', 'time', 'epoch', 'moteid', 'temperature', 'humidity', 'light', 'voltage']

df = pd.read\_csv("intel\_lab\_data.txt", delim\_whitespace=True, names=col\_names)

df['timestamp'] = pd.to\_datetime(df['date'] + ' ' + df['time'], format = ‘mixed’)

df = df.sort\_values(['moteid', 'timestamp'])

agg\_df = (

df.set\_index('timestamp')

.groupby('moteid')

.resample('1min')

.agg(

count=('temperature', 'count'),

mean\_temp=('temperature', 'mean'),

max\_temp=('temperature', 'max'),

std\_light=('light', 'std'),

mean\_humidity=('humidity', 'mean')

)

.reset\_index()

)

agg\_df['rolling5\_mean\_temp'] = (

agg\_df.groupby('moteid')['mean\_temp']

.transform(lambda s: s.rolling(window=5, min\_periods=1).mean())

)

agg\_df.to\_csv("intel\_lab\_minute\_aggregations.csv", index=False)

plt.figure()

for mote in agg\_df['moteid'].unique()[:2]:

sub = agg\_df[agg\_df['moteid'] == mote]

plt.plot(sub['timestamp'], sub['mean\_temp'], label=f"Mote {mote}")

plt.xlabel("Time")

plt.ylabel("Mean Temp (°C)")

plt.title("Mean Temperature per Minute (Tumbling Window)")

plt.legend()

plt.xticks(rotation=45)

plt.tight\_layout()

plt.savefig("chart1\_mean\_temp.png")

plt.close()

plt.figure()

mote = agg\_df['moteid'].unique()[0]

sub = agg\_df[agg\_df['moteid'] == mote]

plt.plot(sub['timestamp'], sub['rolling5\_mean\_temp'])

plt.xlabel("Time")

plt.ylabel("5-min Rolling Mean Temp (°C)")

plt.title(f"5-min Rolling Mean Temp — Mote {mote}")

plt.xticks(rotation=45)

plt.tight\_layout()

plt.savefig("chart2\_rolling5\_temp.png")

plt.close()

plt.figure()

plt.bar(sub['timestamp'], sub['count'])

plt.xlabel("Time")

plt.ylabel("Readings Count")

plt.title(f"Readings per Minute — Mote {mote}")

plt.xticks(rotation=45)

plt.tight\_layout()

plt.savefig("chart3\_count.png")

plt.close()

plt.figure()

mote = agg\_df['moteid'].unique()[1]

sub = agg\_df[agg\_df['moteid'] == mote]

plt.plot(sub['timestamp'], sub['max\_temp'])

plt.xlabel("Time")

plt.ylabel("Max Temp (°C)")

plt.title(f"Max Temp per Minute — Mote {mote}")

plt.xticks(rotation=45)

plt.tight\_layout()

plt.savefig("chart4\_max\_temp.png")

plt.close()

plt.figure()

mote = agg\_df['moteid'].unique()[2]

sub = agg\_df[agg\_df['moteid'] == mote]

plt.plot(sub['timestamp'], sub['mean\_humidity'])

plt.xlabel("Time")

plt.ylabel("Mean Humidity (%)")

plt.title(f"Mean Humidity per Minute — Mote {mote}")

plt.xticks(rotation=45)

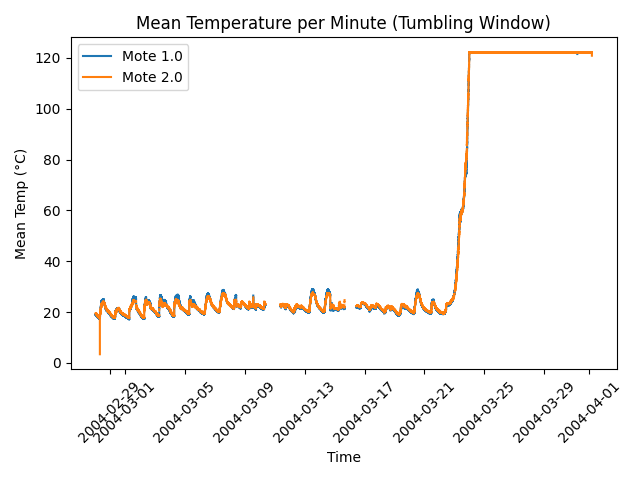
plt.tight\_layout()

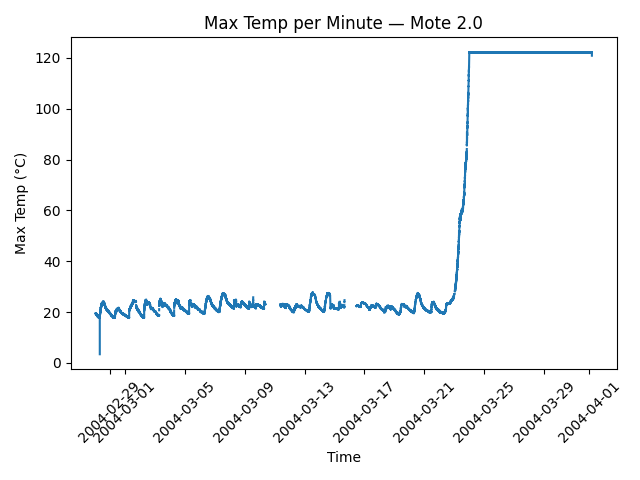
plt.savefig("chart5\_mean\_humidity.png")

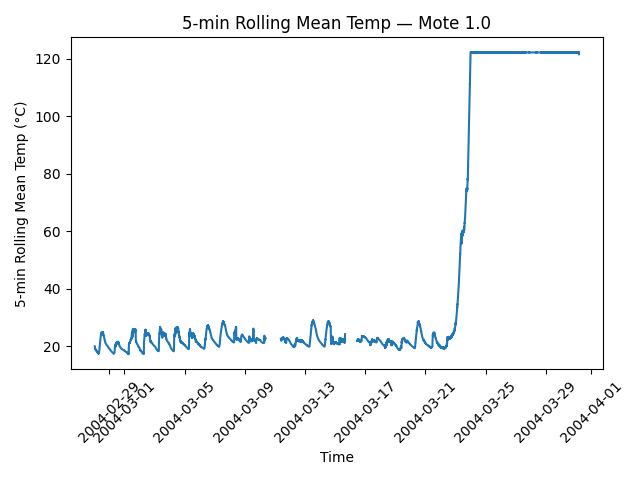
plt.close()

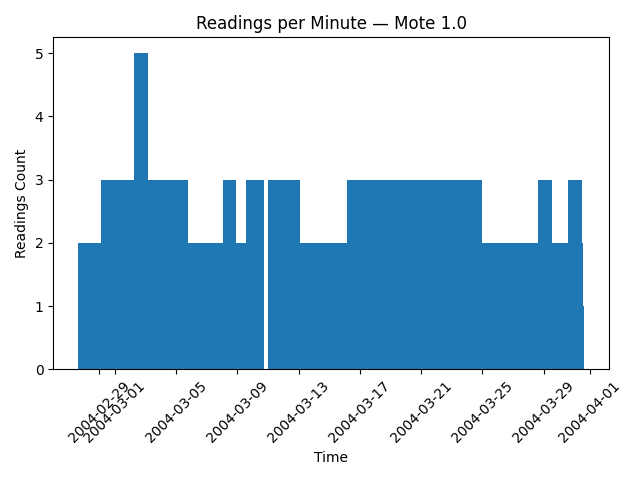
print(" Aggregations complete. CSV and charts saved.")

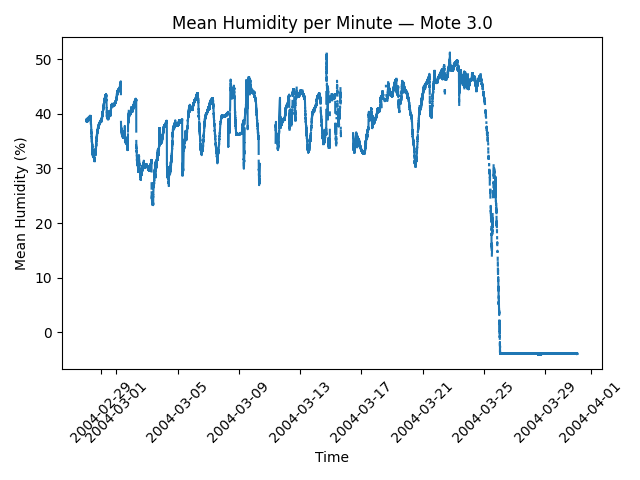
**Sample Output:**

****

****

****

****

****

**Reflection:**

In this lab, I learned how to preprocess IoT sensor data, handle mixed-format timestamps, and apply windowing techniques for real-time analytics simulation. I practiced performing multiple aggregation operations and visualizing results to identify trends and anomalies. This helped me understand how IoT data can be transformed into actionable insights for real-world applications.

Date: 22/8/25

**Lab 07**

**Aim and Objective:**

To find patterns and trends in IoT-based smart home energy consumption data and enhance future energy consumption forecasts by applying machine learning models and time series analysis techniques.

* To prepare data on IoT energy consumption for time series modeling by preprocessing it.
* To use machine learning models (Random Forest, LSTM) and traditional time series models (ARIMA) for forecasting.
* To examine the energy usage data for trends, patterns, and seasonality.
* To use metrics like MAE and RMSE to assess and contrast the performance of various models.
* To interpret the accuracy and usability of the model by visualizing actual versus predicted energy usage. Predictions for the future.

**Problem Statement:**

A lot of real-time data is produced as a result of the growing integration of IoT devices in smart homes (temperature, humidity, appliance usage, etc.). Accurate energy usage forecasting is necessary for energy management and optimization. Prediction is difficult, though, due to the dynamic nature of energy consumption, which is impacted by a variety of user-driven and environmental factors. In order to create precise predictive models that can aid in improving energy management in smart homes, this lab will use time series analysis and machine learning techniques.

**Code:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

rom statsmodels.tsa.seasonal import seasonal\_decompose

from statsmodels.tsa.arima.model import ARIMA

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

df = pd.read\_excel("/content/energydata\_complete.csv.xlsx")

df['date'] = pd.to\_datetime(df['date'])

df.set\_index('date', inplace=True)

data = df['Appliances'].resample('H').mean()

plt.figure(figsize=(12,5))

plt.plot(data, label="Appliances Energy Use")

plt.title("Hourly Appliances Energy Consumption")

plt.legend()

plt.show()

decomposition = seasonal\_decompose(data.dropna(), model='additive', period=24)

decomposition.plot()

plt.show()

train\_size = int(len(data) \* 0.8)

train, test = data[:train\_size], data[train\_size:]

arima\_train = train[-3000:]

arima\_model = ARIMA(arima\_train, order=(1,1,1))

arima\_fit = arima\_model.fit()

arima\_forecast = arima\_fit.forecast(steps=len(test))

def create\_lag\_features(series, lags=24):

    df\_lag = pd.DataFrame({'y': series})

    for i in range(1, lags+1):

        df\_lag[f'lag\_{i}'] = series.shift(i)

    return df\_lag.dropna()

rf\_data = create\_lag\_features(data, lags=24)

train\_size = int(len(rf\_data) \* 0.8)

train\_rf = rf\_data.iloc[:train\_size]

test\_rf = rf\_data.iloc[train\_size:]

X\_train, y\_train = train\_rf.drop('y', axis=1), train\_rf['y']

X\_test, y\_test = test\_rf.drop('y', axis=1), test\_rf['y']

rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf.fit(X\_train, y\_train)

rf\_pred = rf.predict(X\_test)

scaler = MinMaxScaler()

scaled\_data = scaler.fit\_transform(data.values.reshape(-1,1))

def create\_lstm\_data(series, look\_back=24):

    X, y = [], []

    for i in range(len(series) - look\_back):

        X.append(series[i:i+look\_back, 0])

        y.append(series[i+look\_back, 0])

    return np.array(X), np.array(y)

look\_back = 24

X\_lstm, y\_lstm = create\_lstm\_data(scaled\_data, look\_back)

X\_lstm = X\_lstm.reshape((X\_lstm.shape[0], X\_lstm.shape[1], 1))

split = int(len(X\_lstm)\*0.8)

X\_train\_lstm, X\_test\_lstm = X\_lstm[:split], X\_lstm[split:]

y\_train\_lstm, y\_test\_lstm = y\_lstm[:split], y\_lstm[split:]

model = Sequential([

    LSTM(50, return\_sequences=False, input\_shape=(look\_back,1)),

    Dense(1)

])

model.compile(optimizer='adam', loss='mse')

model.fit(X\_train\_lstm, y\_train\_lstm, epochs=5, batch\_size=32, verbose=1)

lstm\_pred\_scaled = model.predict(X\_test\_lstm)

lstm\_pred = scaler.inverse\_transform(lstm\_pred\_scaled.reshape(-1,1)).flatten()

y\_test\_lstm\_rescaled = scaler.inverse\_transform(y\_test\_lstm.reshape(-1,1)).flatten()

def evaluate(true, pred, model\_name):

    true, pred = np.array(true), np.array(pred)

    mask = ~np.isnan(true) & ~np.isnan(pred)

    true, pred = true[mask], pred[mask]

    mae = mean\_absolute\_error(true, pred)

    rmse = np.sqrt(mean\_squared\_error(true, pred))

    print(f"{model\_name} -> MAE: {mae:.2f}, RMSE: {rmse:.2f}")

plt.figure(figsize=(12,5))

plt.plot(test.index, test, label="Actual")

plt.plot(test.index, arima\_forecast, label="ARIMA Forecast")

plt.plot(y\_test.index, rf\_pred, label="Random Forest Prediction")

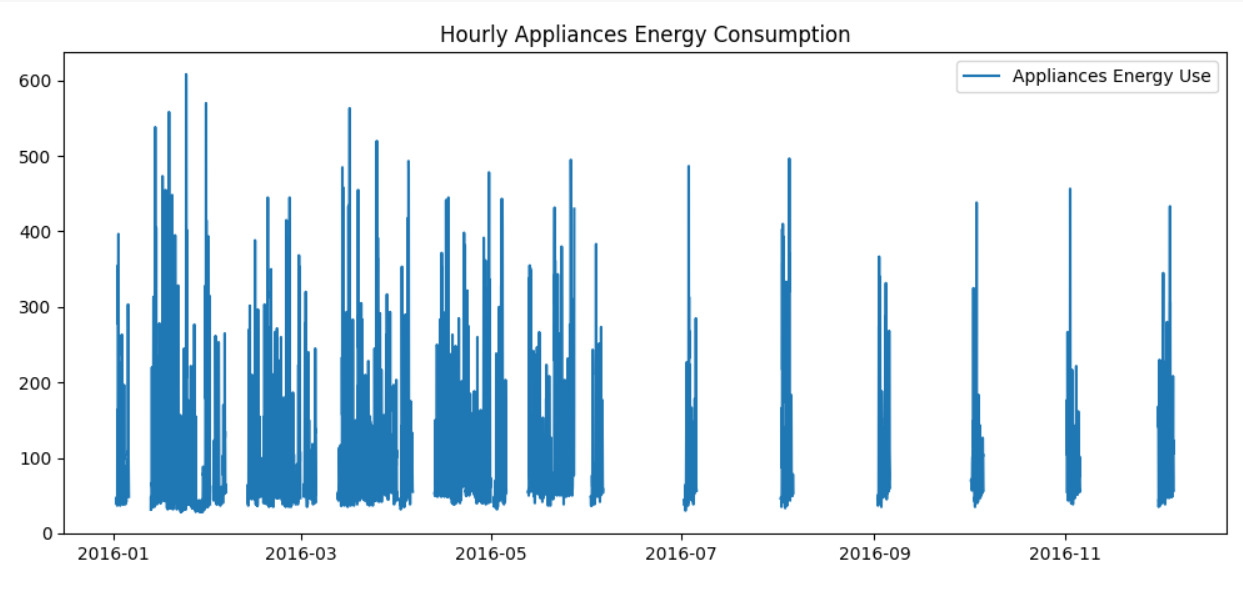
plt.plot(test.index[-len(lstm\_pred):], lstm\_pred, label="LSTM Prediction")

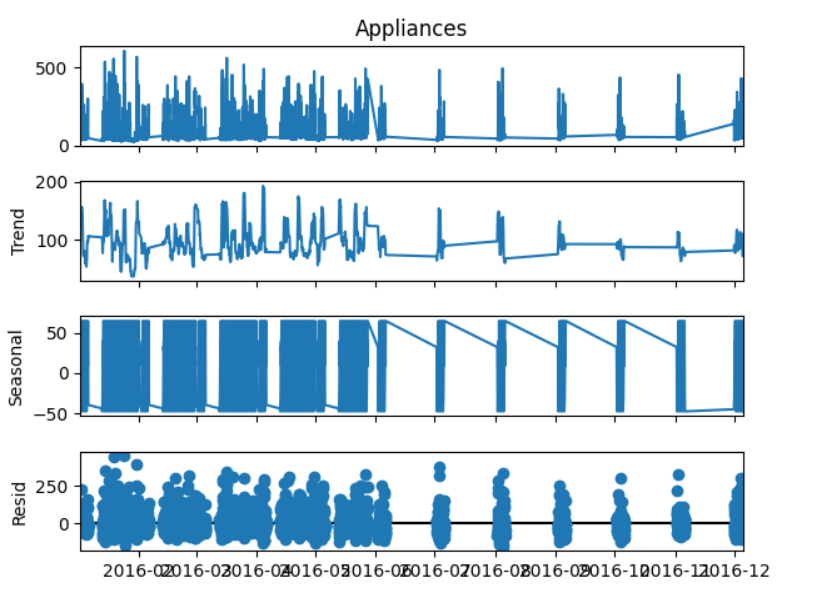
plt.title("Appliances Energy Consumption Prediction")

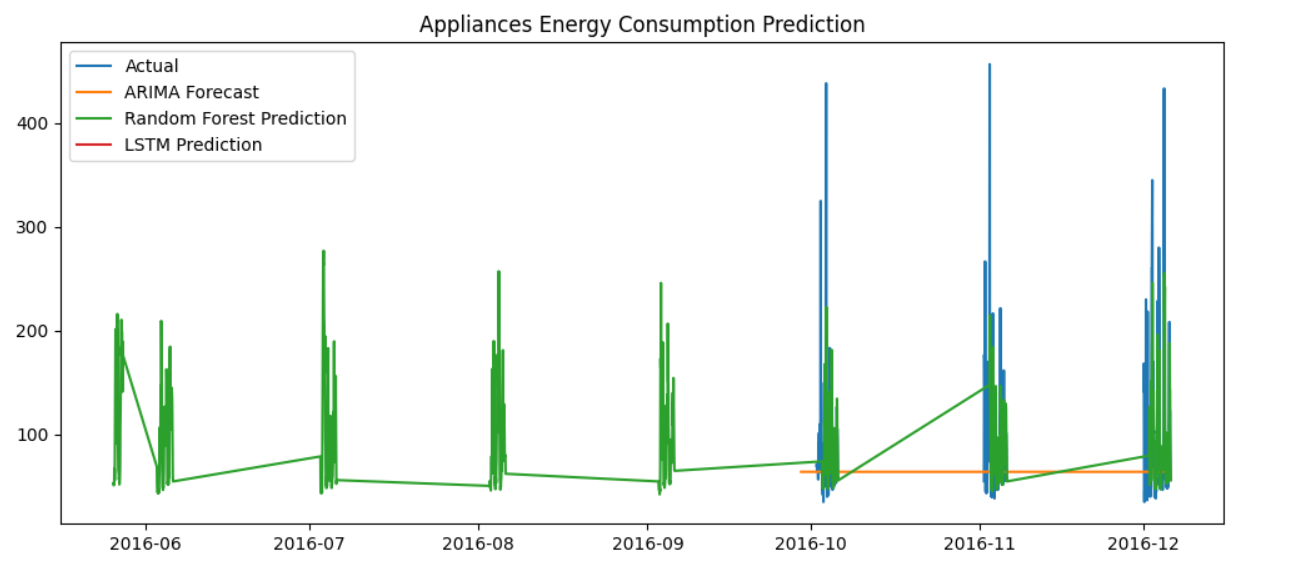
plt.legend()

plt.show()

**Sample Output:**

****

****

****

**Reflection:**

This lab gave participants hands-on experience using machine learning and time series analysis techniques on actual IoT data. Working with a smart home energy dataset taught me how to manage time-indexed data, spot seasonality and trends, and apply different predictive models like LSTM, Random Forest, and ARIMA.

Understanding the advantages and disadvantages of various strategies was crucial; machine learning and deep learning techniques were better at handling complex, non-linear relationships, while classical models like ARIMA were good at capturing short-term patterns.

The significance of appropriate data preprocessing (managing missing values, feature engineering with lags) and model evaluation (using RMSE/MAE and visual comparisons) was also emphasized in this lab. In general, the exercise improved my ability to use predictive analytics for Internet of Things applications and showed how data-driven insights can be used to optimize energy management in smart environments.