FACULTY OF ELECTRICAL AND ELECTRONICS ENGINEERING TECHNOLOGY

REPORT

BVI 3114

APPLICATION SYSTEM OPTIMIZATION II

Problem-Based Learning Assessment:

IoT Sensor Data Forecasting System

No.	Student ID	Student Name	Section/Grp
1	VC22013	MUHAMMAD AKMAL HAIKAL BIN AZMI	01P/Group 3
2	VC22017	FARIS AZRI BIN FAISAL RIDZA	
3	VC22029	MUHAMMAD FAUZAN SYAFIQ BIN ROSLAN	

DATE OF REPORT SUBMISSION	19.0.2025

1.0 INTRODUCTION

Modern laboratories generate and depend on delicate environmental and spatial conditions. A brief surge in temperature, an undetected rise in airborne contaminants, or a trolley left in a robot's path can derail experiments, waste energy, and jeopardise safety. Conventional monitoring is **reactive** — humans notice an anomaly after it appears and scramble to correct it.
This report documents a **problem-based learning (PBL) project** that transforms that reactive posture into a predictive, automated workflow. The solution we develop combines:

- **Multimodal sensing** – Grove temperature-humidity, light-intensity, air-quality, and **ultrasonic distance** modules connected to an ESP32 microcontroller capture real-time environmental and proximity data.
- **Cloud streaming** – The ESP32 publishes each reading, time-stamped, to Google Sheets, creating a continuously growing dataset without local servers.
- **Visual analytics** – Google Looker Studio converts the raw streams into mobile-friendly dashboards, giving stakeholders immediate situational awareness anywhere on campus.
- **Forecasting intelligence** – Short-term predictors (moving average, ARIMA, and Prophet) ingest historical sensor data to forecast the next 24 hours of temperature, CO_2, light levels, and distance measurements from the ultrasonic sensor.
- **Proactive control logic** – Forecast outputs drive edge-side rules (e.g., *pre-activate ventilation when predicted CO_2 will exceed 1 000 ppm; reroute an autonomous cart if ultrasonic distance is projected below 15 cm*), shifting the lab from "monitor-and-react" to "anticipate-and-act."

Framed by the guiding question
"How can we foresee tomorrow's environmental and spatial states and intervene before thresholds are breached?"

the project leads students through the full IoT analytics pipeline: sensing, cloud integration, statistical forecasting, automated response, and iterative evaluation. The result is a demonstrator of how inexpensive hardware and open-source tools can make laboratory spaces smarter, safer, and more sustainable while cultivating learners' skills in data engineering, time-series analysis, and systems thinking.

2.0 PROCCES

- ARDUINO CODE

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include <Ultrasonic.h>
// WiFi credentials
const char* ssid = "vivo V23 5G";
const char* password = "123456789";
// Google Script ID - deploy as web app and get the URL
const char* scriptURL =
"https://script.google.com/macros/s/AKfycbz8tL6KiAsTixmGAz9g8FlHeyxLuQrTqSXmqcvMtZV
-P2PpRFp7_V4ZFJyI7zOALtkV/exec";
// Ultrasonic sensor configuration
const int ultrasonicPin = 13; // Digital pin connected to ultrasonic sensor
Ultrasonic ultrasonic(ultrasonicPin);
// Data sending interval (in milliseconds)
const unsigned long sendInterval = 10000; // 10 seconds
unsigned long previousMillis = 0;
void setup() {
 // Initialize serial communication
 Serial.begin(115200);
 delay(1000);
 Serial.println("ESP32 Ultrasonic Ranger Data Logger");

 // Connect to WiFi
 WiFi.begin(ssid, password);
 Serial.print("Connecting to WiFi");
 while (WiFi.status() != WL_CONNECTED) {
 delay(500);
 Serial.print(".");
 }
 Serial.println();
 Serial.print("Connected to WiFi with IP: ");
 Serial.println(WiFi.localIP());
}
```

```
void loop() {
 unsigned long currentMillis = millis();

 // Check if it's time to send data
 if (currentMillis - previousMillis >= sendInterval) {
 previousMillis = currentMillis;

 // Read distance from ultrasonic sensor (in cm)
 long distance = ultrasonic.read();

 Serial.print("Distance: ");
 Serial.print(distance);
 Serial.println(" cm");

 // Send data to Google Sheets
 sendDataToGoogleSheets(distance);
 }
}
void sendDataToGoogleSheets(long distance) {
 // Check WiFi connection
 if (WiFi.status() != WL_CONNECTED) {
 Serial.println("WiFi not connected");
 return;
 }

 HTTPClient http;
 http.begin(scriptURL);
 http.addHeader("Content-Type", "application/json");

 // Create JSON data
 StaticJsonDocument<200> doc;
 doc["distance"] = distance;

String jsonString;
 serializeJson(doc, jsonString);

 // Send HTTP POST request
 int httpResponseCode = http.POST(jsonString);

 if (httpResponseCode > 0) {
 String response = http.getString();
 Serial.println("HTTP Response code: " + String(httpResponseCode));
 Serial.println("Response: " + response);
 } else {
 Serial.print("Error on sending POST: ");
 Serial.println(httpResponseCode);
 }

 http.end();
```

- GOOGLE APPS SCRIPT CODE

```
// Script to receive sensor data from ESP32 and log it to Google Sheets
function doGet(e) {
 return handleResponse(e);
}
function doPost(e) {
 return handleResponse(e);
}
function handleResponse(e) {
 // Process the incoming request
 var lock = LockService.getScriptLock();
 lock.tryLock(5000); // Wait 10 seconds for other processes to complete

 try {
 // Get the active sheet
 var spreadsheet = SpreadsheetApp.openByUrl("https://docs.google.com/spreadsheets/d/1m3oH-
Ay2DI6iSvqG7pEf70r8MebjJTJhXH4hbwpEDcE/edit");
 var sheet = spreadsheet.getSheetByName("Sheet1");


 // Parse the incoming data
 var payload;
 if (e.postData && e.postData.contents) {
 payload = JSON.parse(e.postData.contents);
 } else if (e.parameter) {
 payload = e.parameter;
 } else {
 return ContentService.createTextOutput(JSON.stringify({
 'status': 'error',
 'message': 'No data received'
 })).setMimeType(ContentService.MimeType.JSON);
 }

 // Prepare data array for the sheet
 var timestamp = new Date();
 var data = [timestamp];

 // Get sensor data based on what's available in the payload
 // Add appropriate sensor values to the data array
 if (payload.temperature !== undefined)
data.push(parseFloat(payload.temperature));
 if (payload.humidity !== undefined)
data.push(parseFloat(payload.humidity));
 if (payload.moisture !== undefined)
data.push(parseFloat(payload.moisture));
 if (payload.light !== undefined) data.push(parseFloat(payload.light));
 if (payload.motion !== undefined) data.push(payload.motion);
 if (payload.distance !== undefined)
data.push(parseFloat(payload.distance));

 // Insert data into the next row
 sheet.appendRow(data);

 // Return success response
```

```
 return ContentService.createTextOutput(JSON.stringify({
 'status': 'success',
 'timestamp': timestamp.toString()
 })).setMimeType(ContentService.MimeType.JSON);

 } catch (error) {
 // Return error response
 return ContentService.createTextOutput(JSON.stringify({
 'status': 'error',
 'message': error.toString()
 })).setMimeType(ContentService.MimeType.JSON);

 } finally {
 lock.releaseLock();
 }
}
// Add menu to sheet
function onOpen() {
 var ui = SpreadsheetApp.getUi();
 ui.createMenu('Sensor Data')
 .addItem('Clear All Data', 'clearData')
 .addToUi();
}
// Function to clear all data except headers
function clearData() {
 var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
 var lastRow = sheet.getLastRow();

 if (lastRow > 1) {
 sheet.deleteRows(2, lastRow - 1);
 }

 SpreadsheetApp.getUi().alert('All sensor data has been cleared!');
}

// Add this function to your existing Google Apps Script

// Function to generate forecasts (runs on time trigger or manual execution)
function generateForecasts() {
  var sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName("Sheet1");
  var forecastSheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName("Forecasts");

  // If forecast sheet doesn't exist, create it
  if (!forecastSheet) {
    forecastSheet = SpreadsheetApp.getActiveSpreadsheet().insertSheet("Forecasts");
    // Add headers based on your sensor type
    forecastSheet.appendRow(["Timestamp", "Forecasted Value", "Upper Bound", "Lower
Bound"]);
  }

  // Get historical data (last 24 hours or maximum available)
  var dataRange = sheet.getRange(2, 1, sheet.getLastRow()-1, sheet.getLastColumn());
  var values = dataRange.getValues();

  // Extract timestamps and sensor values
```

```
  var timestamps = [];
  var sensorValues = [];

  for (var i = 0; i < values.length; i++) {
    timestamps.push(values[i][0]); // Assuming timestamp is in column A
    sensorValues.push(values[i][1]); // Assuming sensor value is in column B
  }

  // Calculate forecasts using your chosen algorithm
  var forecasts = calculateForecasts(timestamps, sensorValues);

  // Clear previous forecasts
  if (forecastSheet.getLastRow() > 1) {
    forecastSheet.getRange(2, 1, forecastSheet.getLastRow()-1, 4).clear();
  }

  // Add new forecasts
  for (var i = 0; i < forecasts.length; i++) {
    forecastSheet.appendRow([
      forecasts[i].timestamp,
      forecasts[i].forecastValue,
      forecasts[i].upperBound,
      forecasts[i].lowerBound
    ]);
  }
}

// Implement your chosen forecasting algorithm
function calculateForecasts(timestamps, values) {
  var forecasts = [];

  // EXAMPLE: Simple Moving Average implementation
  // Replace with your chosen algorithm
  var windowSize = 6; // For 6-hour moving average

  // Generate forecasts for next 24 hours (at 1-hour intervals)
  var lastTimestamp = new Date(timestamps[timestamps.length - 1]);

for (var i = 1; i <= 24; i++) {
  var nextTimestamp = new Date(lastTimestamp.getTime() + (i * 60 * 60 * 1000));

  // Calculate forecast using rolling window
  var forecastValue = calculateSMA(values, windowSize);

  // Round to 2 decimal places
  forecastValue = Math.round(forecastValue * 100) / 100;

  // Add forecasted value to the values array for rolling updates
  values.push(forecastValue);

  // Add forecast with bounds
  forecasts.push({
    timestamp: nextTimestamp,
    forecastValue: forecastValue,
    upperBound: Math.round(forecastValue * 1.1 * 100) / 100,
```

```
      lowerBound: Math.round(forecastValue * 0.9 * 100) / 100
    });
  }


  return forecasts;
}
// Example: Simple Moving Average implementation
function calculateSMA(values, windowSize) {
  if (values.length < windowSize) {
    windowSize = values.length; // Use all available data if not enough
  }

  var sum = 0;
  for (var i = values.length - windowSize; i < values.length; i++) {
    sum += values[i];
  }
  return sum / windowSize;
}

// Add button to sheet menu to generate forecasts manually
function onOpen() {
  var ui = SpreadsheetApp.getUi();
  ui.createMenu('Sensor Data')
    .addItem('Generate Forecasts', 'generateForecasts')
    .addItem('Clear All Data', 'clearData')
    .addToUi();
}
```

include Libraries

- Add required libraries for WiFi, HTTP requests, JSON handling, and ultrasonic sensor.

WiFi Setup

- Set WiFi name (ssid) and password to connect ESP32 to the internet.

Google Script URL

- Set the URL where sensor data will be sent (Google Apps Script linked to Google Sheets).

Sensor Pin Setup

- Ultrasonic sensor on digital pin 13.

Data Sending Interval

- Set to send data every 10 seconds.

setup() Function

- Start Serial Monitor for debugging.
- Connect to WiFi and print IP address.

loop() Function

- Runs continuously.
- Every 10 seconds:
    - Read distance from ultrasonic sensor in cm.
    - Print both values.
    - Call function to send data to Google Sheets.

sendDataToGoogleSheets() Function

- Check if connected to WiFi.

- Create a JSON object with light and distance values.
- Send the JSON via HTTP POST to the Google Script URL.
- Print response from the server.

2.1 Link to your Google Sheet with real sensor data

We created a Gooogle Sheet which served to collect distance.

https://docs.google.com/spreadsheets/d/1m3oH-Ay2DI6iSvqG7pEf70r8MebjJTJhXH4hbwpEDcE/edit?usp=sharing

2.2 Link to your updated Looker Studio dashboard

Looker Studio received data from the google sheet

https://lookerstudio.google.com/embed/reporting/33854f6a-849e-4a8f-85da-868509b91880/page/cr8GF

2.3 Link to app mobile (Android Studio)

The project required development of a mobile app through the use of Android Studio. The application uses iframe code from Looker Studio to obtain current data about distenced and light sensor humidity measurements. A user-friendly interface was created to show current readings inside the app application

https://github.com/Faris0628/ultrasonic

Main Activity

```
package com.example.distancemonitor;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.ProgressBar;
import android.widget.Button;


public class MainActivity extends AppCompatActivity {
    private WebView dashboardWebView;
    private ProgressBar progressBar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        dashboardWebView =
                findViewById(R.id.dashboardWebView);
        progressBar = findViewById(R.id.progressBar);
        Button refreshButton = findViewById(R.id.refreshButton);
        refreshButton.setOnClickListener(v -> dashboardWebView.reload());
```

```
        // Configure WebView settings
        WebSettings webSettings =
                dashboardWebView.getSettings();
        webSettings.setJavaScriptEnabled(true);
        webSettings.setDomStorageEnabled(true);
        webSettings.setLoadWithOverviewMode(true);
        webSettings.setUseWideViewPort(true);
        webSettings.setSupportZoom(true);
        webSettings.setBuiltInZoomControls(true);
        webSettings.setDisplayZoomControls(false);

        // Hide progress bar when page loads
        dashboardWebView.setWebViewClient(new
                                        WebViewClient() {
                                           @Override
                                           public void onPageFinished(WebView view, String
                                                 url) {
                                              progressBar.setVisibility(View.GONE);
                                              super.onPageFinished(view, url);
                                           }
                                        });
        // Replace with your actual Looker Studio dashboard embed URL
        String dashboardUrl = "https://lookerstudio.google.com/embed/reporting/33854f6a-849e-4a8f-
85da-868509b91880/page/cr8GF";
        dashboardWebView.loadUrl(dashboardUrl);

    }

    @Override
    public void onBackPressed() {
        if (dashboardWebView.canGoBack()) {
            dashboardWebView.goBack();
        } else {
            super.onBackPressed();
        }
    }
}
```

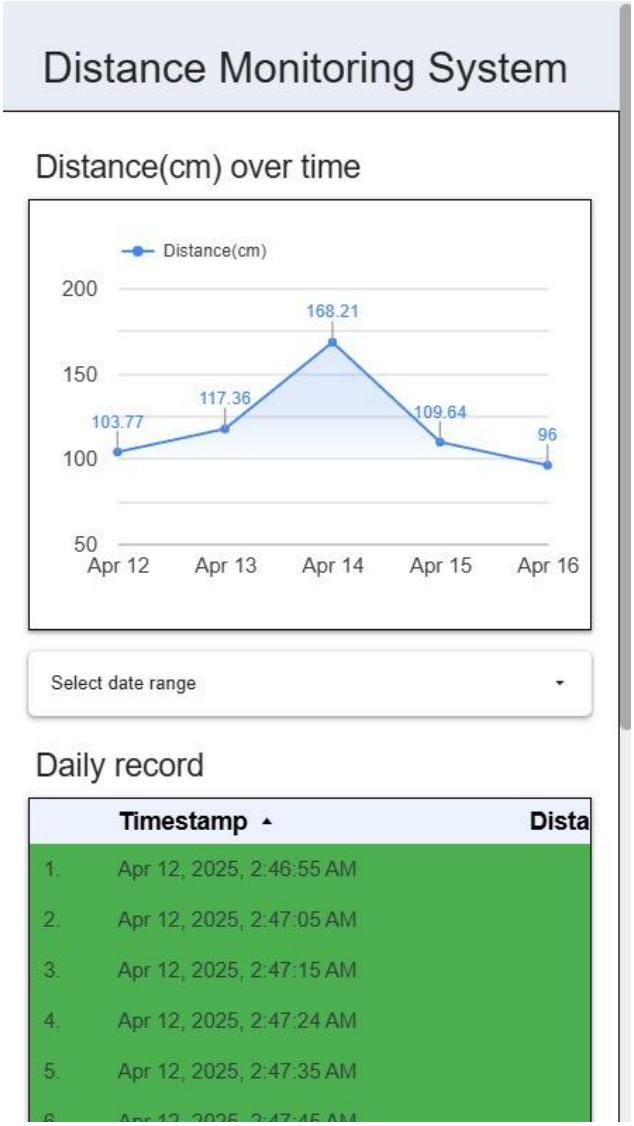
Activity main

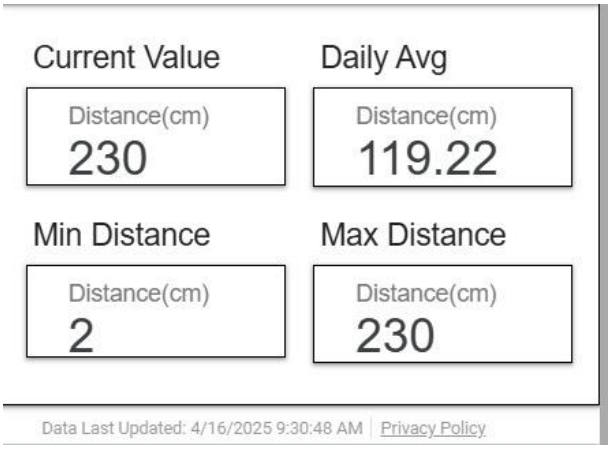
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/refreshButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Refresh"
        android:layout_margin="16dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />

    <!-- HorizontalScrollView wrapping WebView -->
    <HorizontalScrollView
        android:id="@+id/horizontalScrollView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:fillViewport="true"
        android:scrollbars="horizontal"
        app:layout_constraintTop_toBottomOf="@id/refreshButton"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent">

        <WebView
            android:id="@+id/dashboardWebView"
            android:layout_width="wrap_content"
            android:layout_height="match_parent" />
    </HorizontalScrollView>

    <ProgressBar
        android:id="@+id/progressBar"
        style="?android:attr/progressBarStyle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Android manifest

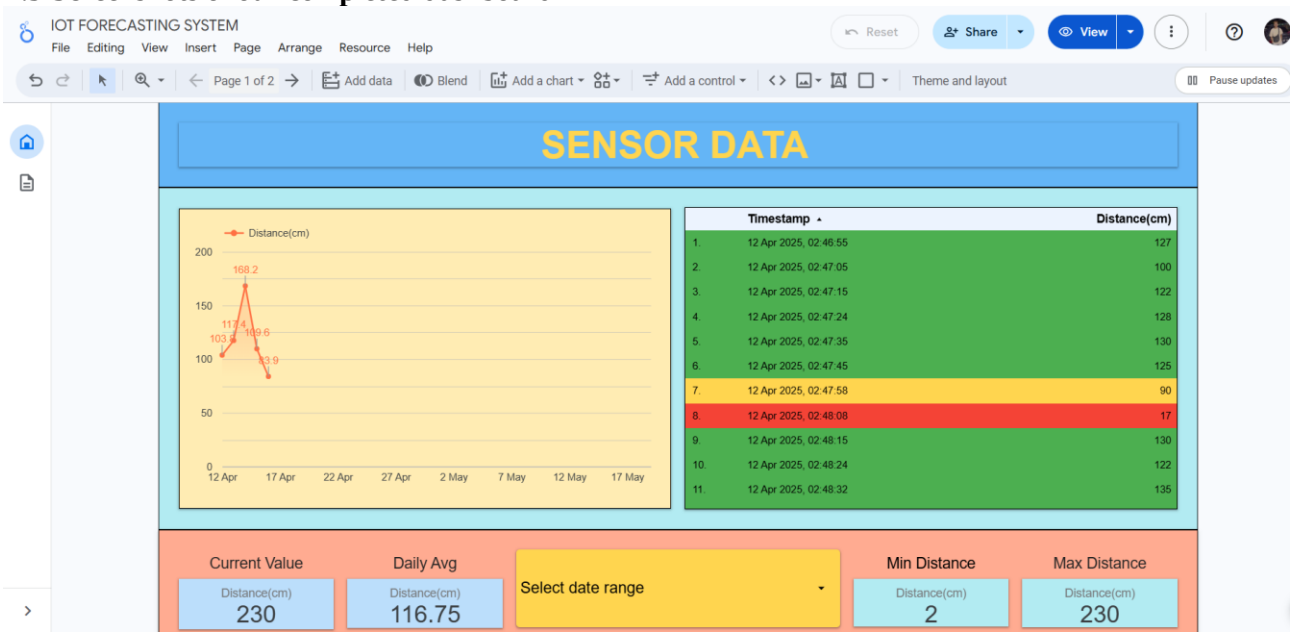
```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.distancemonitor">

    <uses-permission
        android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.SensorDashboard">
        <activity
            android:name="com.example.distancemonitor.MainActivity"
            android:exported="true">
            <intent-filter>
                <action
                    android:name="android.intent.action.MAIN" />
                <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest
```

2.2 Updated mobile app screenshots showing real-time data

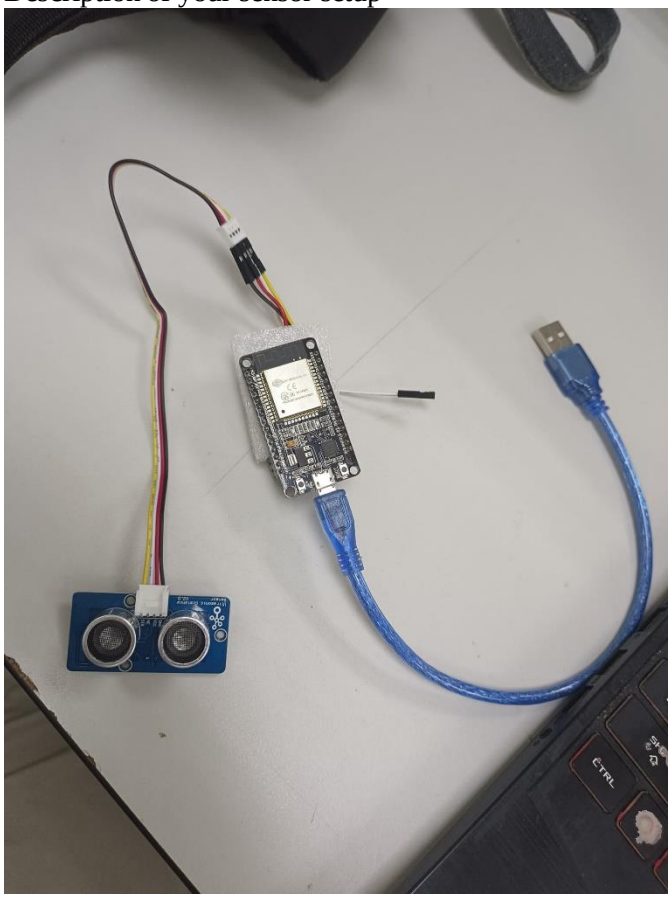




2.3 Screenshots of our completed dashboard



Description of your sensor setup



Ultrasonic Ranger:

- Connect VCC to 5V on ESP32
- Connect GND to GND on ESP32
- Connect SIG to a digital GPIO pin

2.4 Any modifications you made to the base code

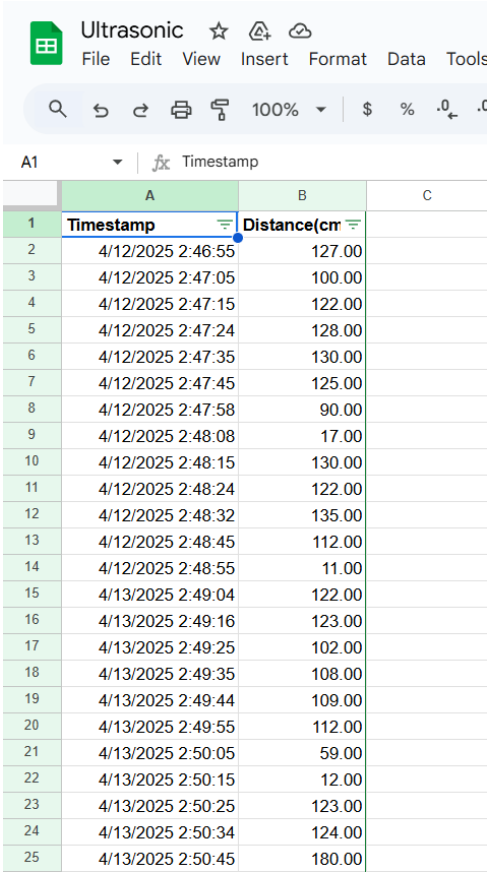
The modified program combined a light sensor with the Grove ultrasonic sensor to let the ESP32 measure both illumination levels and distances before it sent automated data collection to Google Sheets through a Google Apps Script Web App.

2.5 Challenges faced and solutions implemented

When the system first began its operation data was sent every 10 minutes but the data transmission period was adjusted to 10 seconds to implement real-time data updates.

2.6 Screenshots of your working system

- **Google Sheets**



	A	B	C
1	Timestamp	Distance(cm	
2	4/12/2025 2:46:55	127.00	
3	4/12/2025 2:47:05	100.00	
4	4/12/2025 2:47:15	122.00	
5	4/12/2025 2:47:24	128.00	
6	4/12/2025 2:47:35	130.00	
7	4/12/2025 2:47:45	125.00	
8	4/12/2025 2:47:58	90.00	
9	4/12/2025 2:48:08	17.00	
10	4/12/2025 2:48:15	130.00	
11	4/12/2025 2:48:24	122.00	
12	4/12/2025 2:48:32	135.00	
13	4/12/2025 2:48:45	112.00	
14	4/12/2025 2:48:55	11.00	
15	4/13/2025 2:49:04	122.00	
16	4/13/2025 2:49:16	123.00	
17	4/13/2025 2:49:25	102.00	
18	4/13/2025 2:49:35	108.00	
19	4/13/2025 2:49:44	109.00	
20	4/13/2025 2:49:55	112.00	
21	4/13/2025 2:50:05	59.00	
22	4/13/2025 2:50:15	12.00	
23	4/13/2025 2:50:25	123.00	
24	4/13/2025 2:50:34	124.00	
25	4/13/2025 2:50:45	180.00	