ECE LABORATORY

DREXEL UNIVERSITY

To: **Dr. Peters**

From: **Ehi Simon**

Re: **ECE 304 Lab 9 – Live-Time Plotting and Recording to Google Sheets**

PURPOSE:

The purpose of this week's lab is to implement live-time plotting of sensor readings using Charts.js, implement live-time plotting of sensor readings using the Google Visualization API, and record sensor readings in live time to Google Sheets using IFTTT.

Discussion:

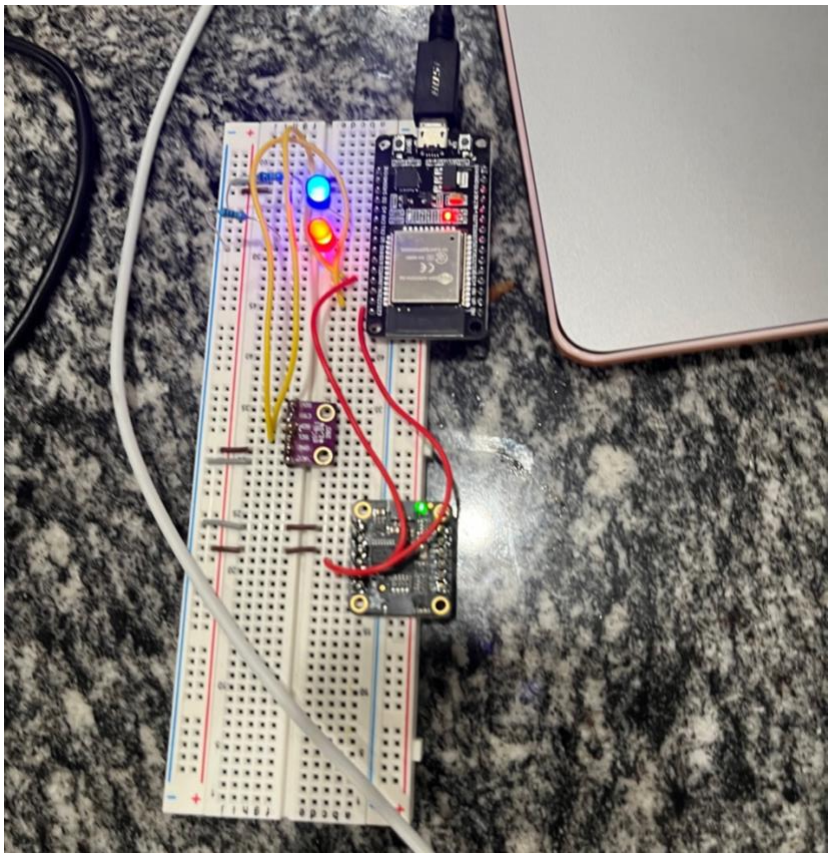


Fig. 1. Circuit Connection for Project 2

The circuit for the lab was built like the one above. It consists of 2 330Ω resistors, a red LED, a blue LED, a BME280 Environmental Sensor, an Adafruit BNO085 IMU, and an ESP32S microcontroller.

The setup of the project was simpler than previous projects. I had to create an IFTTT account and have a Google account to access Google Sheets.

Main.cpp

In my main.cpp file, I initialized the multiple libraries that were needed for the sensors to work and provide readings. I also included libraries to get the ESP32 to connect to STA, and libraries needed to interact with a server in JSON. I had to install the Adafruit JSON library prior. The LEDs are defined, the sea level reference pressure is defined and the BNO08X chip is reset. The BME object is created and the network SSID and password. The web server is then opened on port 80. This can all be found in the figure below:

Fig. 2. Figure Showing Initialization of Libraries and Variable Definitions

```
// Replace with your SSID and Password
const char* ssid = "Verizon_C4ZKVV";  // Enter SSID here
const char* password = "dna-blear7-bow";  //Enter Password here

// Replace with your unique IFTTT URL resource
const char* resource = "/trigger/YPR/with/key/cchmT2GVlCrLdRpIFh5xHT";

// Maker Webhooks IFTTT
const char* server = "maker.ifttt.com";

// Time to sleep
uint64_t uS_TO_S_FACTOR = 1000000;  // Conversion factor for micro seconds to seconds
// sleep for 30 minutes = 1800 seconds
uint64_t TIME_TO_SLEEP = 5;

// Create object for the BME
Adafruit_BME280 bme;

// Establish a Wi-Fi connection with your router
void initWifi() {
  Serial.print("Connecting to: ");
  Serial.print(ssid);
  WiFi.begin(ssid, password);

  int timeout = 10 * 4; // 10 seconds
  while(WiFi.status() != WL_CONNECTED  && (timeout-- > 0)) {
    delay(250);
    Serial.print(".");
  }
  Serial.println("");

  if(WiFi.status() != WL_CONNECTED) {
     Serial.println("Failed to connect, going back to sleep");
  }

  Serial.print("WiFi connected in: ");
  Serial.print(millis());
  Serial.print(", IP address: ");
  Serial.println(WiFi.localIP());
}
```

Fig. 3. Figure Showing Code Setting Up Wi-Fi Connection

In the figure above, the first two lines of code are setting up the Wi-Fi connection to my network at home. I then set the IFTTT website as the server to connect to. I also create an object for my BME sensor. The function initWifi() is made to establish a wi-fi connection with my router. I put in the trigger event as the resource and needed to make use of my API key.

```
void makeIFTTTRequest() {
  Serial.print("Connecting to ");
  Serial.print(server);

  WiFiClient client;
  int retries = 5;
  while(!!!client.connect(server, 80) && (retries-- > 0)) {
    Serial.print(".");
  }
  Serial.println();
  if(!!!client.connected()) {
    Serial.println("Failed to connect...");
  }

  Serial.print("Request resource: ");
  Serial.println(resource);
  float y = ypr.yaw;
  float p = ypr.pitch;
  float r = ypr.roll;
  float T = bme.readTemperature();
  float H = bme.readHumidity();
  float P = bme.readPressure()/1000;

  String jsonObject = String("{\"value1\":\"") + y +
                      "\",\"value2\":\"" + p +
                      "\",\"value3\":\"" + r + "\"}";
  Serial.println(jsonObject);

  client.println(String("POST ") + resource + " HTTP/1.1");
  client.println(String("Host: ") + server);
  client.println("Connection: close\r\nContent-Type: application/json");
  client.print("Content-Length: ");
  client.println(jsonObject.length());
  client.println();
  client.println(jsonObject);

  int timeout = 5 * 10; // 5 seconds
  while(!!!client.available() && (timeout-- > 0)){
    delay(100);
  }
  if(!!!client.available()) {
    Serial.println("No response...");
  }
  while(client.available()){
    Serial.write(client.read());
  }

  Serial.println("\nclosing connection");
  client.stop();
}
```

Fig. 4. Figure Showing makeIFTTTRequest Function

The figure above shows the setup to make the request to IFTTT. It also uploads the yaw, pitch, and roll values to the trigger event which then uses Webhooks to upload it to a google sheets file.

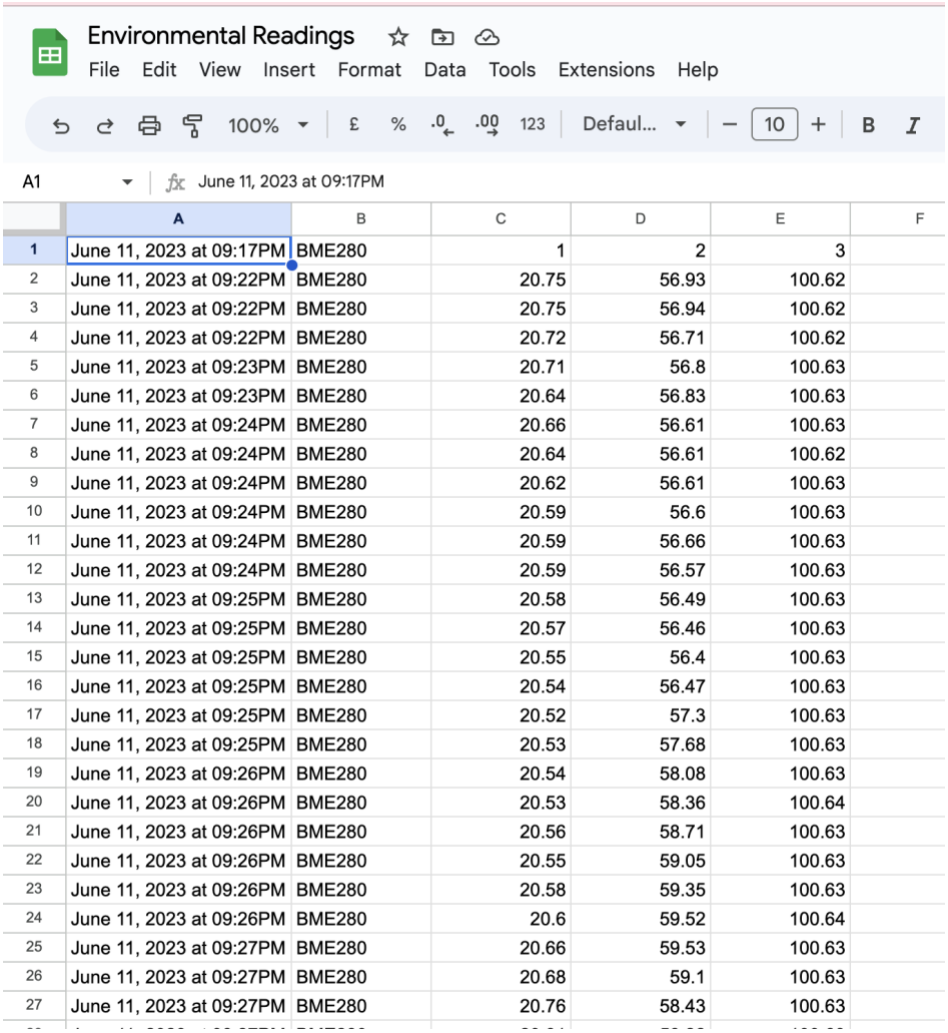
```
void setup() {
  Serial.begin(115200);
  delay(2000);
  if (!bno08x.begin_I2C()) {
    Serial.println("Failed to find BNO08x chip");
    while (1) { delay(10); }
  }
  Serial.println("BNO08x Found!");


  setReports(reportType, reportIntervalUs);

  if (bno08x.wasReset()) {
    Serial.print("sensor was reset ");
    setReports(reportType, reportIntervalUs);
  }
  if (bno08x.getSensorEvent(&sensorValue)) {
    quaternionToEulerRV(&sensorValue.un.arvrStabilizedRV, &ypr, true);

    static long last = 0;
    long now = micros();
    Serial.print(now - last);             Serial.print("\t");
    last = now;
    Serial.print(sensorValue.status);     Serial.print("\t");  // This is accuracy in the range of 0 to 3
    Serial.print(ypr.yaw);                Serial.print("\t");
    Serial.print(ypr.pitch);              Serial.print("\t");
    Serial.println(ypr.roll);
  }

  Serial.println("Reading events");
  delay(100);
  bool status;
  status = bme.begin(0x76);
  if (!status) {
      Serial.println("Could not find a valid BME280 sensor, check wiring!");
  while (1);
  }
  initWifi();
  makeIFTTTRequest();
  // enable timer deep sleep
  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
  Serial.println("Going to sleep now");
  // start deep sleep for 3600 seconds (60 minutes)
  esp_deep_sleep_start();
}

void loop() {
  // sleeping so wont get here
}
```

Fig.5. Figure Showing Setup and Loop Functions

The figure above shows the setup function of the file. This setup function does more than those in previous projects. It initializes the BME280 sensor and the BNO08X chip. It calls the initWifi and makeIFTTTRequest functions. The setup function makes it possible to open the webpage and it also gets the yaw, pitch, and roll values. The loop function remains empty because the ESP is sleeping when it gets to the loop function.

IFTTT

I had to create an event, I did for both the BME280 sensor readings and for the BNO08X chip. I named the trigger event and set it up such that once the website pulls readings from my ESP32 (after uploading my code), it updates a row in a google sheet it creates for the readings respectively. These figures can be found below.



	A	B	C	D	E	F
1	June 11, 2023 at 09:17PM	BME280	1	2	3	
2	June 11, 2023 at 09:22PM	BME280	20.75	56.93	100.62	
3	June 11, 2023 at 09:22PM	BME280	20.75	56.94	100.62	
4	June 11, 2023 at 09:22PM	BME280	20.72	56.71	100.62	
5	June 11, 2023 at 09:23PM	BME280	20.71	56.8	100.63	
6	June 11, 2023 at 09:23PM	BME280	20.64	56.83	100.63	
7	June 11, 2023 at 09:24PM	BME280	20.66	56.61	100.63	
8	June 11, 2023 at 09:24PM	BME280	20.64	56.61	100.62	
9	June 11, 2023 at 09:24PM	BME280	20.62	56.61	100.63	
10	June 11, 2023 at 09:24PM	BME280	20.59	56.6	100.63	
11	June 11, 2023 at 09:24PM	BME280	20.59	56.66	100.63	
12	June 11, 2023 at 09:24PM	BME280	20.59	56.57	100.63	
13	June 11, 2023 at 09:25PM	BME280	20.58	56.49	100.63	
14	June 11, 2023 at 09:25PM	BME280	20.57	56.46	100.63	
15	June 11, 2023 at 09:25PM	BME280	20.55	56.4	100.63	
16	June 11, 2023 at 09:25PM	BME280	20.54	56.47	100.63	
17	June 11, 2023 at 09:25PM	BME280	20.52	57.3	100.63	
18	June 11, 2023 at 09:25PM	BME280	20.53	57.68	100.63	
19	June 11, 2023 at 09:26PM	BME280	20.54	58.08	100.63	
20	June 11, 2023 at 09:26PM	BME280	20.53	58.36	100.64	
21	June 11, 2023 at 09:26PM	BME280	20.56	58.71	100.63	
22	June 11, 2023 at 09:26PM	BME280	20.55	59.05	100.63	
23	June 11, 2023 at 09:26PM	BME280	20.58	59.35	100.63	
24	June 11, 2023 at 09:26PM	BME280	20.6	59.52	100.64	
25	June 11, 2023 at 09:27PM	BME280	20.66	59.53	100.63	
26	June 11, 2023 at 09:27PM	BME280	20.68	59.1	100.63	
27	June 11, 2023 at 09:27PM	BME280	20.76	58.43	100.63	

Fig.6. BME Sensor Readings in Google Sheets

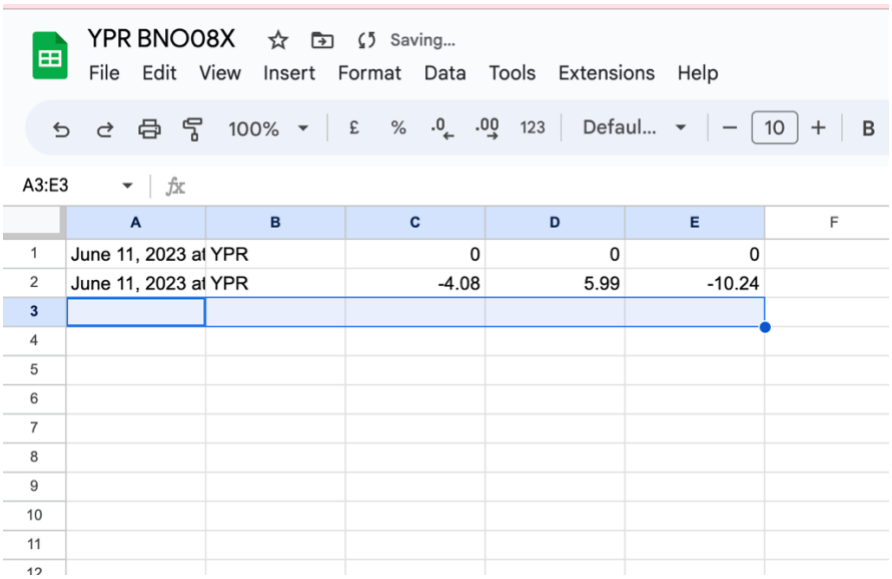


Fig.6. Yaw, Pitch, and Roll Uploaded to Google Sheets

Fig.7 Testing YPR Trigger Event

HTML

I also had to create a HTML webpage that could pull values from the ESP32 device live and plot it on a webpage. The code can be found on the page below and the result can be seen in a figure below as well.

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <title>Climate Monitoring</title>
  <script type="text/javascript"src="https://www.gstatic.com/charts/loader.js"></script>
 </head>

 <body>
  <h1 align=center>Climate Monitoring</h1>
  <!--Div that will hold the google chart-->
  <div id="chart_div" align='center'></div>

  <script type="text/javascript">
   // Load the Visualization API and the corechart package.
   google.charts.load('current', {'packages': ['corechart']});

   // Set a callback to run when the Google Visualization API is loaded.
   google.charts.setOnLoadCallback(drawChart);

   // Restrict the number of points we keep in the chart at any time
   const maxDatapoints = 20;


   // Set chart display options
   const chartOptions = {
    title: 'Temperature Profile',
    width: 800,
    height: 300,
    legend: 'none',
    vAxis: { title: 'Temperature (C)', viewWindowMode: "pretty", minValue: 0, maxValue: 40},
    hAxis: { title: 'Time' }
   };

   // Modified websocket code to update Google Chart on each message from ESP
   // This code will run at the end of the drawChart() callback
```

```
function initWebsocket(chart, dataTable) {
  const ws = new WebSocket("ws://192.168.1.94:80");

  ws.onmessage = (evt) => {
    // Code here runs every time a message is received
    const msg = JSON.parse(evt.data)
    const temp = msg['Temperature'];

    const now = new Date();
    dataTable.addRow([now, temp]);

      // dataTable.fg.length is the number of datapoints in foreground
    if (dataTable.getNumberOfRows() > maxDatapoints) {
      dataTable.removeRow(0);
    }
    chart.draw(dataTable, chartOptions);
  };

  // close websocket when leaving the page
  window.onbeforeunload = () => {
    ws.close();
  };
};

// Callback that creates and populates a data table,
// instantiates the pie chart, passes in the data and
// draws it.
function drawChart() {
  // Create the data table.
  const dataTable = new google.visualization.DataTable();
  dataTable.addColumn('date', 'Time');
  dataTable.addColumn('number', 'Temperature (C)');

  // Instantiate and draw our chart, passing in some options.
  const chart = new google.visualization.ScatterChart(document.getElementById('chart_div'));
  chart.draw(dataTable, chartOptions);
```

```
      // Set up connection with ESP and tell Javascript what to do with new messages
      initWebsocket(chart, dataTable);
    }
  </script>
 </body>
</html>
```

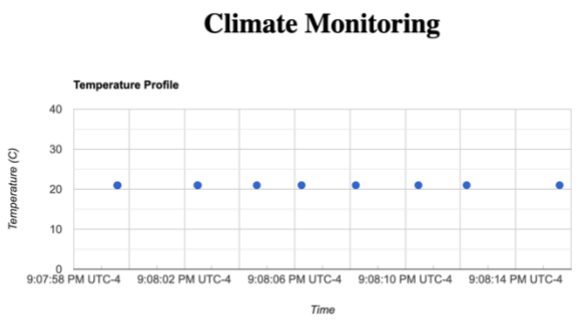


Fig.8. Figure Showing Webpage with Temperature Monitoring Live Plotted

Conclusion

In this lesson, I learned how to implement live-time data logging and plotting using different methods. We explored the use of Charts.js and Google API for real-time visualization of sensor data. I also learned how to send data from an ESP32 to Google Sheets for live updating. Additionally, I gained insights into the importance of live time plotting for identifying failing sensors, system casualties, and trends in real-time data. Overall, this lesson provided valuable hands-on experience in data visualization and analysis.