İZMİR KATİP ÇELEBİ UNIVERSITY

FACULTY OF ENGINEERING AND ARCHITECTURE DEPARTMENT OF MECHATRONICS ENGINEERING

MEE303 Sensor Systems Lab Project Report

Mehmet Berke Parlat 160412014

Hasan Ünlü 180412035

Metehan Sarıoğlan 190412063

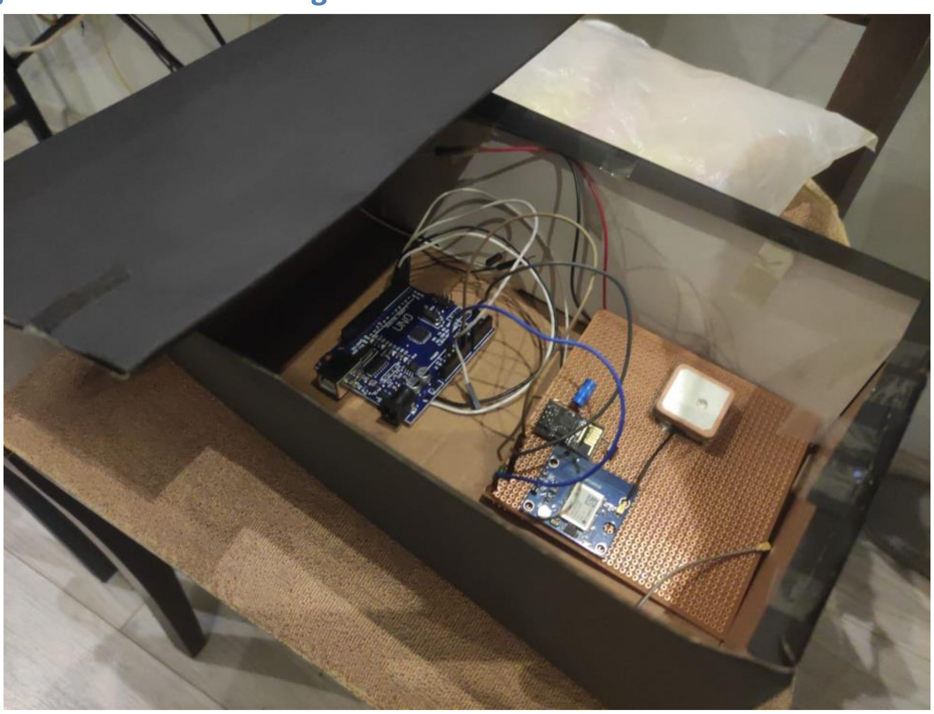
1. Introduction

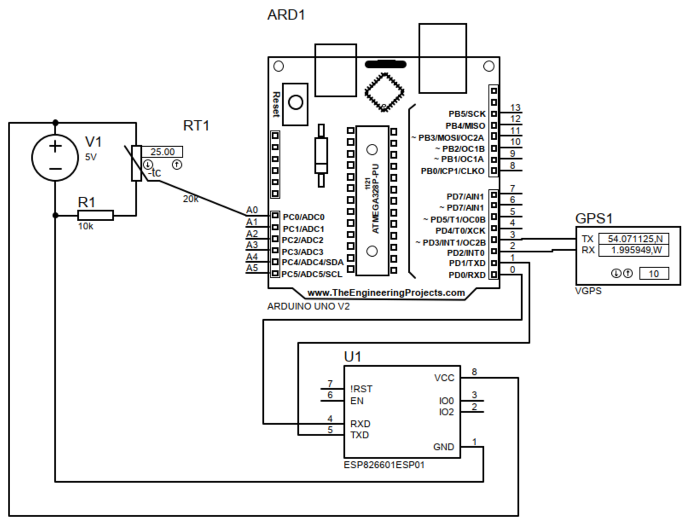
The project's main goal is to create a custom mobile device with integrated sensors for real-time environmental data collection, particularly temperature and location. Utilizing Arduino UNO, ESP 8266-01 Wi-Fi Module, and NEO-6M GPS Module, the device captures and broadcasts data over Wi-Fi for processing and visualization on a PC. This facilitates the creation of a dynamic temperature map, useful in environmental monitoring, urban planning, and climate studies. The report details the design, development, and technical aspects of this IoT-enabled device, highlighting its practicality and efficiency in live data collection and analysis.

2. Equipment and Software Requirements

1. Arduino UNO
2. ESP 8266-01 Wi-Fi Module
3. NEO-6M GPS Module
4. NTC (Negative Temperature Coefficient) Thermistor
5. 9 Volt Battery and its appropriate connector
6. 3.3 Voltage Regulator (LD1117V33)
7. On/Off Switch
8. Headers and terminal blocks
9. Resistors and capacitors
10. Perforated Board
11.Arduino IDE
12. Visual Studio
13. Python IDE

3. Design and Connection Diagram





4. Communication Protocols and Details

The Arduino Uno communicates with the NEO-6M GPS Module via UART, reading NMEA sentences for vital navigational data like longitude and latitude. Efficient data parsing is crucial to process this information in real-time. The Arduino Uno also interfaces with the ESP8266-01 Wi-Fi Module through serial communication. A stable connection for sensor data transfer is maintained using AT commands, with careful configuration of communication parameters.

Additionally, the ESP8266-01 transmits data to a PC over Wi-Fi. The PC software, using TCP/IP, receives temperature and GPS data, ensuring reliable data handling and secure network communication. This setup allows for the effective display of data in formats such as temperature maps.

5. Application and Functional Requirements

The application efficiently parses GPS data from the NEO-6M module, extracting key details like latitude, longitude, and altitude for real-time tracking. A core function involves using the ESP8266-01 module to broadcast sensor data, including temperature and GPS coordinates, over Wi-Fi. This process requires robust network management and secure data transmission.

Additionally, the system addresses the voltage disparity between the Arduino Uno (5V) and the ESP8266-01 (3.3V) by implementing logic level shifting to ensure safe and reliable data exchange. Communication between the PC and the electronic box is established via TCP/IP protocol, with the PC software handling data reception, parsing, and displaying it in an interactive format like a temperature map. This software is designed for user-friendliness, with features for real-time visualization and data analysis.

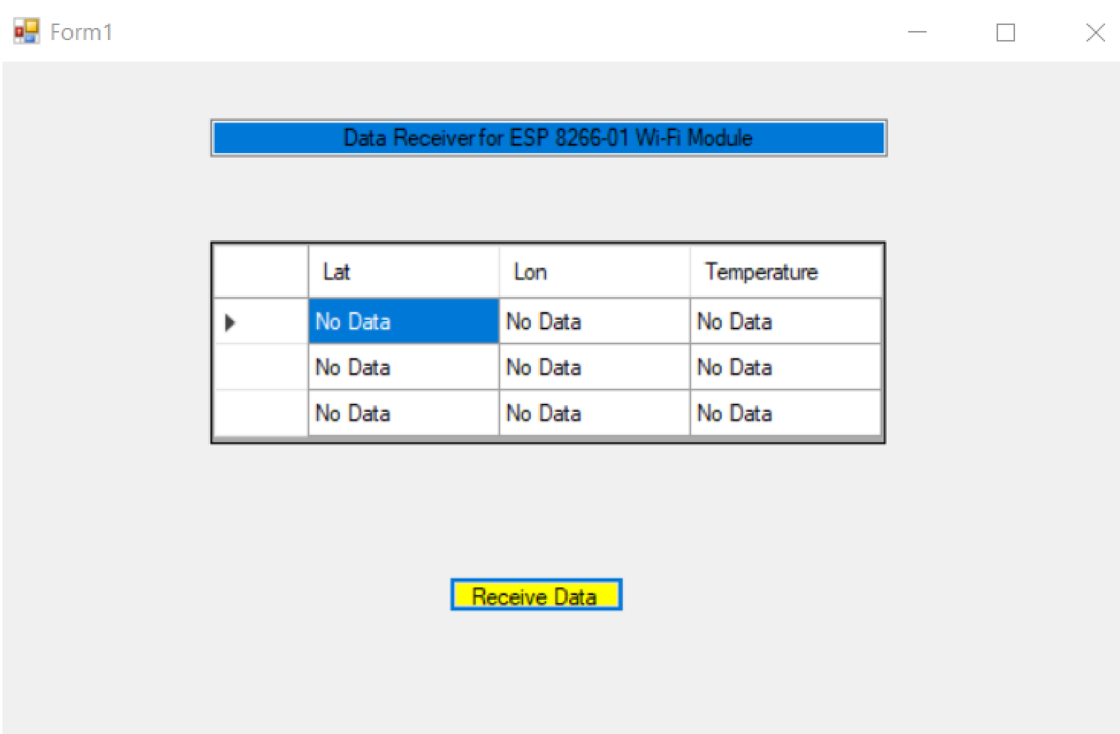
6. Implementation Details

The implementation process began with assembling the electronic box, housing the Arduino UNO, ESP 8266-01 Wi-Fi Module, NEO-6M GPS Module, and other components, ensuring efficient layout and minimal component interference. The Arduino was programmed for data reading, processing, and communication with the ESP8266-01. The ESP8266-01, configured for Wi-Fi connectivity, transmitted data to a PC. A custom software application, developed in C# and Python, was created on the PC to receive, parse, and visually display this data.

Testing and troubleshooting were integral, focusing on individual components, module integration, and overall device performance. Issues like data transmission errors and GPS inconsistencies were resolved through code refinement and communication optimization. The final outcome was a fully functional device, effectively gathering environmental data and interfacing with the PC to create a dynamic temperature map.

7. Results and Observations

Data Receiver Windows Forms



8. Conclusion

The project marks a major advancement in IoT and environmental data analysis, demonstrating a custom-built device's capability for efficient real-time data collection. Key achievements include the development of a reliable system for gathering and transmitting environmental data, using Arduino UNO, ESP 8266-01 Wi-Fi Module, and NEO-6M GPS Module. The project overcame challenges in hardware integration and software development, offering valuable lessons in sensor integration and data communication.

Future enhancements could focus on improving energy efficiency, Wi-Fi stability, and adding more sensors. The possibility of an advanced PC application for data analytics and cloud integration is also envisioned. Overall, this project exemplifies the potential of modern technology in environmental monitoring and opens new possibilities for future advancements.

9. Appendices

C# Code for data receiver

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Net.Sockets;
using System.Net;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;

namespace WindowsFormsApp1
{
    3 başvuru
    public partial class Form1 : Form
    {
        private string ipAddress = "192.168.1.33";  // Server IP Address (ESP8266-01)
        private int port = 5000;                    // Communication Port (Specified in ESP8266-01 code)
        private uint noReadCounter = 0;             // Counter to keep number of non-receiving data

        public TcpClient tcpclnt;                   // Instance of a TcpClient
        Stream stm;                                 // Instance of a Stream data

        public String tempMessage = "";             // Temporary variable to keep message in data package.
        public String wholeMessage = "";            // Variable to keep whole message in data package, once it is created properly.
        public bool startMessageFlag = false;       // Flag to keep the specified package is began or not.
        public bool stopMessageFlag = false;        // Flag to keep the specified package is ended or not.

        public bool socketReady = false;            // Flag to keep connection status

        public float lat;                           // Variable to keep latitude data
        public float lon;                           // Variable to keep longitude data
        public float temperature;

        4 başvuru
        class ClassTemperature
        {
            1 başvuru
            public string Lat { get; set; }
            1 başvuru
            public string Lon { get; set; }
            1 başvuru
            public string Temperature { get; set; }
```

```
            1 başvuru
            public ClassTemperature(string lat, string lon, string temperature)
            {
                Lat = lat;
                Lon = lon;
                Temperature = temperature;
            }
        }
        1 başvuru
        public Form1()
        {
            InitializeComponent();

            if(!socketReady)
            {

            }
            while (socketReady)
            {

            }
        }

        1 başvuru
        private void button1_Click(object sender, EventArgs e)
        {
            List<ClassTemperature> temperatures = new List<ClassTemperature>()
                {
                    new ClassTemperature("No Data","No Data", "No Data")
                };
            temperatures.Add(temperatures[0]);
            temperatures.Add(temperatures[0]);
            dataGridView1.DataSource = temperatures;
        }
        0 başvuru
        public void DataUsageLoop()
        {
            while (true)
            {
                Thread.Sleep(500);      // Delay loop for a prefered time.
            }
        }

        0 başvuru
        public void SetupSocket()
```

```
        {
            tcpclnt = new TcpClient();                  // Create a new instance of a TcpClient
            Console.WriteLine("Connecting...");
            try
            {
                tcpclnt.Connect(ipAddress, port);       // Trying to connect the ipaddress and port.
                stm = tcpclnt.GetStream();              // Getting the stream over WiFi.
                socketReady = true;                     // Change flag to true, means that the connection is alive.
            }
            catch (Exception e)
            {
                Console.WriteLine("Socket error:" + e);     // Connection is failed.
                socketReady = false;                        // Change flag to false, means that the connection is failed.
            }
        }

        0 başvuru
        public void ReadSocket()
        {
            byte[] bb = new byte[100];                              // Creating receiving bytes
            stm.ReadTimeout = 1000;                                 // Setting timeout for WiFi communication
            try
            {
                int k = stm.Read(bb, 0, 100);                       // Read stream over WiFi


                for (int i = 0; i < k; i++)
                {
                    char convertedChar = Convert.ToChar(bb[i]);     // Converting received byte element to char value

                    if (convertedChar.Equals('#'))
                    {
                        startMessageFlag = true;                    // Change flag to true, means that the specified package is began.
                        stopMessageFlag = false;                    // Change flag to false, means that the specified package is began; so the stop is non-true.

                    }
                    else if (convertedChar.Equals('*'))
                    {
                        stopMessageFlag = true;                     // Change flag to true, means that the specified package is ended.
                        startMessageFlag = false;                   // Change flag to false, means that the specified package is ended; so the start is non-true.
                        wholeMessage = String.Copy(tempMessage);    // Attend tempMessage to wholeMessage, which is the whole character collection of tempMessage.
                        tempMessage = "";                           // Clear tempMessage so that the next package is to be attended.
                        break;                                      // Break the loop
                    }
                    if (startMessageFlag && !stopMessageFlag)
```

```
                        break;                                      // Break the loop
                    }
                    if (startMessageFlag && !stopMessageFlag)
                    {
                        tempMessage += convertedChar;               // Add received characters together on tempMessage variable.
                    }
                }
            }
            catch
            {
                noReadCounter++;                                // Increase the counter by 1
                if (noReadCounter >= 10)
                {
                    CloseSocket();                              // Call CloseSocket() method to close the communication.
                    Console.WriteLine("Socket Closed!");
                    socketReady = false;                        // Change flag to false, means that the connection is lost.
                }
            }
        }

        1 başvuru
        public void CloseSocket()
        {
            Console.WriteLine("CONNECTION CLOSED");
            tcpclnt.Close();
        }

        1 başvuru
        private void dataGridView1_CellContentClick(object sender, DataGridViewCellEventArgs e)
        {

        }

        1 başvuru
        private void Form1_Load(object sender, EventArgs e)
        {

        }

        1 başvuru
        private void textBox1_TextChanged(object sender, EventArgs e)
        {

        }
    }
}
```

Arduino code

```cpp
#include <SoftwareSerial.h>
#include <math.h>

// Pin definitions
const int NTC_PIN = A0;  // Thermistor pin
const int RX_PIN = 10;   // RX pin for ESP8266
const int TX_PIN = 11;   // TX pin for ESP8266

// Software serial for ESP8266
SoftwareSerial esp8266(RX_PIN, TX_PIN);

void setup() {
  Serial.begin(9600);      // Start serial communication with GPS module
  esp8266.begin(9600);     // Start serial communication with ESP8266

  setupESP8266();          // Initialize the ESP8266 module
  pinMode(NTC_PIN, INPUT); // Setup NTC Thermistor pin
}

void loop() {
  String gpsData = readGPSData();
  float temperature = readTemperature();
  sendData(gpsData, temperature);
  delay(1000); // Wait for a second before next read
}

String readGPSData() {
  String data = "";
  while (Serial.available()) {
    char c = Serial.read();
    if (c == '\n') {
      String type = data.substring(0, 6);
      if (type == "$GPGGA") {
        // Parse data here
      }
      data = ""; // Reset the data string
    } else {
      data += c; // Build the data string
    }
  }
  return data;
}

float readTemperature() {
  int analogValue = analogRead(NTC_PIN);
  return convertToTemperature(analogValue);
```

```cpp
}

float convertToTemperature(int analogValue) {
  float resistance = (1023.0 / analogValue) - 1;
  resistance = 10000 / resistance;

  float B = 3950;
  float temperature0 = 273.15 + 25.0; // Reference temperature in Kelvin

  float temperature = temperature0 * B / (B + temperature0 * log(resistance / 10000)) - 273.15; // Convert to Celsius
  return temperature;
}

void setupESP8266() {
  esp8266.println("AT");
  delay(2000);
  esp8266.println("AT+CWMODE=1");
  delay(2000);
  esp8266.println("AT+CWJAP=\"RaspiSpot\",\"Mechatronics402\"");
  delay(5000);
}

void sendData(String gpsData, float temperature) {
  esp8266.println("AT+CIPSTART=\"TCP\",\"<192.168.1.35>\",<5000>");
  delay(2000);
  String dataToSend = "GPS Data: " + gpsData + "; Temperature: " + String(temperature);
  esp8266.println("AT+CIPSEND=" + dataToSend.length());
  delay(2000);
  esp8266.println(dataToSend);
  delay(2000);
  esp8266.println("AT+CIPCLOSE");
}

void parseGPGGA(String gpsData) {
  // Example: "$GPGGA,hhmmss.ss,latitude,N,longitude,E,fix,.."
  int firstCommaIndex = gpsData.indexOf(',');
  int secondCommaIndex = gpsData.indexOf(',', firstCommaIndex + 1);
  int thirdCommaIndex = gpsData.indexOf(',', secondCommaIndex + 1);
  int fourthCommaIndex = gpsData.indexOf(',', thirdCommaIndex + 1);

  // Extract latitude and longitude
  String latitude = gpsData.substring(secondCommaIndex + 1, thirdCommaIndex);
  String longitude = gpsData.substring(fourthCommaIndex + 1, gpsData.indexOf(',', fourthCommaIndex + 1));
  Serial.println("Latitude: " + latitude);
  Serial.println("Longitude: " + longitude);
}
```