IoT-Enabled Environmental Monitoring Technologies

Clayton Suell
University of South Alabama

IoT-Enabled Environmental Monitoring Technologies

By

Clayton Suell

A thesis submitted in partial fulfillment of the requirements of the Honors College at University of South Alabama and the Bachelor of Science in Computer Engineering in the Electrical and Computer Engineering Department

University of South Alabama

Mobile

May 2021

Approved by:

[illegible] Digitally signed by Jinhui Wang
DN: cn=Jinhui Wang, o=University of South Alabama, ou=Electrical and Computer Engineering, email=jwang@southalabama.edu, c=US
Date: 2021.04.24 20:16:41 -05'00'

Mentor: Jinhui Wang

Samuel H. Russ

Committee Member: Samuel Russ

[illegible]

Committee Member: Jingshan Huang

Kathy J. Cooke
Dean, Honors College

Acknowledgements

Firstly, I would like to thank my advisor Dr. Wang, if not for him I would never have gotten into IoT technologies. He saw potential in me early on and gave me the resources and support to complete this project. Without his support I would probably never have finished writing this thesis either as he pushed me to continue working even on the days I wanted to give up.

I would like to thank Tom Hammel and the other members of the IMPACT lab. Their input on this project was invaluable. When I began this project, I had very little understanding of using Raspberry pi and doing research and they helped me in understanding how everything worked together. They also provided great encouragement during late nights working on this project in the lab.

Last but not least, I would like to thank my friends and family for all their support going to college and getting involved in this project. If it were not for their support, I would not have taken the time or been willing to put in the effort to complete this project.

Abstract

Environmental monitoring is essential to protect human life and the environment due to issues such as global warming, pollution, and weather phenomena like hurricanes. To this end this thesis presents a prototype IoT (Internet of Things)-enabled environmental monitoring network. The network will include the ability to collect data from various locations that will be uploaded to a cloud so that the data can be accessed from any device with a network connection allowing for greater environmental awareness and a database that can be used for environmental analysis. The network will be flexible enough that every data collection point will either connect directly to the internet, send data through a gateway with an internet connection, or upload data separately. For communication between devices, a mesh network will be used to allow communication from end nodes to router nodes and finally to a central or coordinator node which will upload all the data from the mesh network to the cloud. This work will outline all the hardware, software, and setup required to prototype the proposed monitoring network and will conclude with some potential future improvements.

Table of Contents

List of Tables

List of Tables

Part 1 – Introduction

Today, the environment has become a major concern. Global warming continues to be an ongoing problem that has contributed to rising sea levels and changes in weather patterns to be more severe than any other time in recorded history. Pollution is causing environmental hazards that lead to unsafe food and water supplies. Many scientists have begun to address these issues but there is still much work to be done.

Scientists from across disciplines have begun working together to collect and analyze massive amounts of data about our environment [1]. One of the issues in this work is how to effectively collect data in a cost-effective manner. Over the past few years multiple scientists have proposed systems to directly address this issue. In 2016, a team of engineers proposed a system using a single board computer known as a Raspberry Pi, sensors, and the cloud service Adafruit IO to create an IoT-enabled sensor station [2]. In 2017, Forcella proposed a mesh sensor network that utilizes Raspberry Pi and Xbee communication modules [3]. In 2018, another team of engineers tested an Arduino with sensors to setup an IoT enabled environmental sensor station [4].

Previous works introduced approaches to creating environmental sensing devices that can be built upon. This work aims to combine using mesh-networks with IoT and provide a method for viewing the collected data in “real-time” from any device with a web browser. The purposed system will include arrays of nodes connected to the cloud through central nodes. The completed model provides a method to increase environmental awareness and make a database for environmentalists to use for environmental analysis. The model is tested using affordable sensors and can be altered as necessary by changing the sensors for a variety of applications.

1.1 Background

Today we face grave environmental issues such as global warming, pollution, and weather phenomena that continue to become more severe. For this reason, environmental monitoring is essential to protect not only our environment but also human life. Global warming is a primary issue that ninety percent of climate scientists agree has been affected by humans [5]. Climate change is a problem that is requiring scientists from across disciplines to work together to collect and analyze massive amounts of data [1]. Analyzing the data includes processing Petabytes worth of environmental data to create models of environmental change as well as linking this data to social and economic issues. The World Health Organization (WHO) agrees that both noise and air pollutants are human and environmental health risks that can disrupt our daily lives [6].

In 2015, a group at Quest International University Perak explored the idea of using a single board computer known as a Raspberry Pi in network applications. Raspberry Pi is a low-cost computer that runs a Linux operating system called Raspbian [7]. This work was done with the goal of finding that a Raspberry Pi can be used in IoT (Internet of Things) applications. IoT is a form of device-to-device communication over the internet that can allow things such as device commands and data to be sent between devices without the need for direct human interaction. In their research, they used a Raspberry Pi model B+ and tested its usage as a client-server using first Wi-Fi and then a mesh network protocol called Zigbee. A software called Samba was used to setup the Raspberry Pi as a server over Wi-Fi. To test the Zigbee application they used Xbee radio modules which were manufactured by a company called Digi. To configure and use the Xbee modules, they used a program called XCTU in windows and a program called minicom to run in Raspbian on the Raspberry Pi. In the end, their work concluded that the Pi is useful for smart applications and file transfer [7].

In 2016, Shete and Agrawal explored the idea of setting up a Raspberry Pi 2 model B to collecting environmental data [2]. They set out with the goal of designing a system that would promote a better quality of life by providing more data on the environment around us. To achieve this goal, they proposed an IoT sensor station that uses a Raspberry Pi to collect data from sensors and the Adafruit IO server to allow data to be viewed and commands to be sent to the sensor station from any device that has internet access. The Python programming language was chosen for this project because it is a primary language built into Raspbian and due to its wide range of available libraries that allow for a wide range of functionality. They setup a single station and did a short-term test to find that it could effectively collect environmental data [2].

As opposed to a single sensor stations, Forcella proposed a mesh sensor network that is intended to provide a cost-effective method to collect environmental data [3]. He did this with the intention of it being used for data collection on lakes. For each node he connected sensors to a Raspberry Pi 3, an Xbee module, and a self-sustaining power source made up of a rechargeable battery and a solar panel. The network he proposed uses Raspberry Pies as controllers at each node, Xbee modules to communicate between nodes, and a data station to save data in a MySQL database. The Xbee modules communicate between nodes using the ZigBee protocol and the Raspberry Pies use the Node.js coding language to run the nodes. In his network the data is saved in csv files that are sent from the nodes every 30 minutes. The network collects data and sends it to the data station over the internet. Forcella only got to do preliminary testing in a lab environment and did not get to testing the network outside of a lab. A serious issue he encountered was that the battery would die after a few hours of runtime. He did some testing to get data about the amount of voltage the Raspberry Pi used but did not get to finish fixing this issue [3].

In 2018, Parmar, Nagda, Palav, and Lopes did work that proposed an IoT sensor station that used Arduino as a low-cost alternative to Raspberry Pi [4]. They used sensors to monitor carbon monoxide and carbon dioxide levels, temperature and humidity, noise, and rainfall. They setup their station uploads data to a cloud service. From there the data is processed into graphs which can be viewed in a web browser. [4].

XBee modules have proven to be a useful, cost-effective method for device-to-device communication. The hardware reference manual provides data on the Xbee modules which is necessary to understand how they can be used [8]. The Xbee 3 modules operate using 2.1-3.6V and have an operating range of up to 60 meters in an indoor or urban environment and up to 1200 meters with line-of-sight [8]. XBee modules have support for network protocols based on the 802.15.4 network standard including Digi Mesh and ZigBee [8]. These mesh protocols allow network nodes to connect multiple devices to a single network that can all communicate with each other.

XBee devices have been used for multiple applications and over time new ways of using the devices have been developed. One of the more recent additions is an official Python library for setting up and communicating between devices [9]. The Python library works in Python 3.7 and allows data and commands to be sent using only a few lines of code. It sends data in a bitstream which does have to be encoded and decoded to get more useful data formats but built-in functions in Python make this a simple task [9].

In 2016, Nitin Naik looked into using Google Cloud, Google Drive, and some of the various tools offered by google to create a system of systems (SoS) [10]. This SoS allows for collected data to be stored and analyzed with minimal setup. The tools Nitin looked at using this can be used with little to no programming and technical knowledge. They can also be accessed by anyone who has access, at any time, from anywhere that they have an internet connection to access the tools over. These are some of the advantages that come with using a cloud service to store and analyze data. The paper also mentions that while these tools can be used at minimal cost there are limitations to how much you can do based on how much you are willing to pay. Nitin specifically looked at using Google Drive, Google Refine, Google Fusion Tables, Google Maps, Google Sheets, and Google Charts [10]. Each of these tools provide a way to store and/or analyze data from various types of files. The tools can also be used to visualize the data onto maps and charts.

1.2 Goals

This project aims to develop and test a model to setup a cost-effective IoT-enabled environmental monitoring network. This will also include an effective way to present the environmental data collected from the network. The implemented prototype will allow the data to be accessed from any location with an internet connection and a compatible browser via the cloud. The collected data will be presented on a map showing the location of each node along with environmental information. These data will be setup to update approximately every 5 minutes yielding “real time” results. The completed model will be under $200 per node using cost effective hardware options. The network and nodes will be of a modular design so that anyone with an understanding of the code and hardware could adapt the model to their needs.

Part 2 – Methodology

2.1 Hardware Components

Today there are several options to consider when choosing each hardware component. Given the many choices available, there are several potential approaches to reaching the goals of this project. Only one solution will be presented in this model, however an analysis of the hardware choices made will be discussed here.

2.1.1 Node Controller Raspberry Pi 3B+

When considering a controller for each node, there are two categories to choose between, a microcontroller, and an SBC (Single Board Computer). The distinct differences between these two categories are usually price, processing power, connectivity, and memory capacity. Depending on the device a microcontroller is typically cheaper than an SBC. For example, an Arduino Uno R3 (microcontroller) is typically priced at $30 and a Raspberry Pi 3B+ (SBC) costs $35 with no SD card for memory. SBCs are more powerful than microcontrollers as they are built to be able to run full fledge operating systems typically based on Linux. SBCs usually include more connectivity options such as Wi-Fi and Bluetooth. SBCs also typically have a larger RAM capacity and the choice of larger storage when compared to a microcontroller. There are valid reasons to choose either of these options, but it was decided to use an SBC to get access to Wi-Fi, memory for data backup, and access to greater processing power that will be used in future work to add things such as on node machine learning to perform initial data processing.

While there are many options for SBCs, Raspberry Pi is one of the most widely used and supported SBCs available. When looking at other SBC options it is either more expensive to get a more powerful board, or the board is a cheaper board that may leave out features such as wi-fi or have issues with software support and compatibility. That is not to say that there are not viable alternatives, but for this project a Raspberry Pi 3B+ was chosen. At the time of the start of this project the Raspberry Pi 4 had not released yet and thus the 3B+ was still the latest and most powerful version of a Raspberry Pi. The specs for the Raspberry Pi 3B+ are shown in table 2.1.

	Raspberry Pi 3B+
Price	$35
Processor	4-Cores @ 1.4GHz
Ram	1GB LPDDR2 SDRAM
Connectivity	Wi-Fi, Bluetooth, Ethernet
USB Ports	4x USB A
GPIO	40 Pin Header

Table 2.1 – Raspberry Pi 3B+ Specifications

2.1.2 Zigbee Network Modules – Xbee 3

When looking at short range network communications options, Zigbee was chosen because it offered an expandable mesh network option. Xbee modules provide a device that can be interfaced with a Raspberry Pi over a USB. For this project Xbee 3 modules were used because they offered the latest version of the Xbee series with increased and range and lower power consumption. These modules also offered compatibility with a new Python 3 library for simplified coding.

2.1.3 Sensors

Over the course of this project a variety of sensors were purchased, used, and tested. The devices used included Sense Hats, temperature and humidity sensors from the Sunfounder Sensor Kit v3.0, Grove Air Quality Sensor V1.3, and the DIYmall VK-162 GPS Module. Each of these could be setup to be read from using Python code, however some provided better results than others.

The Raspberry Pi Sense Hat is priced at $35. It connected to the 40-pin GPIO header and included temperature, humidity, barometric pressure, gyroscope, accelerometer, and magnetometer sensors. The Sense Hat also had an 8x8 RGB LED matrix and a joystick. For the purposes of this project however, the most useful features were the ability to read temperature, humidity, and barometric pressure. While each of these sensors did work, the results were usually highly inaccurate. There were two things that were done to help some with the accuracy, additional code was written to help correct the issue and GPIO extender cables were purchased to get the Sense Hat away from the heat produced by the Raspberry Pi processor. In the end however, due to the price point and the fact that only the environmental sensor functions were being used, it was determined that for the purpose of this project the Sense Hat is not an ideal device.

Individual sensors that come calibrated with a much higher accuracy can be purchased for $5-$10 apiece, however in the interest of having a variety of sensor options for other possible uses for the Raspberry Pi in the future, the Sunfounder Sensor Kit v3.0 was purchased. From this kit the sensors used were the Analog Temperature Sensor and the Humiture Sensor. Each of these sensors were plugged into the GPIO header as indicated by the kits wiring diagram and the python example code was used for the implementation. The Humiture Sensor was more useful as it offered temperature and humidity readings. The sensor used a DHT11 component which offers measurements with accuracies of ± 5% for humidity and ± 2% for temperature. The Analog Temperature Sensor offered similar accuracy for reading only temperature, making the Humiture sensor the better choice as it measured both temperature and humidity accurately. The DHT11 based Humiture sensor proved to be an accurate and affordable option coming in at under $10 per sensor.

Another data metric desired for this project was air quality. For this the Grove Air Quality Sensor v1.3 was chosen. Since it was a grove sensor it plugged directly into the development board along side the Xbee 3 module allowing data to be read through the input output lines on the Xbee 3 using the Python library. The sensor gives a value that represents the concentration it detects of a variety of gases including carbon monoxide, alcohol, acetone, thinner, and formaldehyde. While this sensor does not provide specific air metrics, it does provide a method to determine how fresh or polluted the air is.

The last module purchased was the DIYmall VK-162 GPS which is a simple GPS device that connects via a USB port. The module provided a necessary component when considering that someone could move one of the sensor nodes or could even steal one, it can allow for the location of the sensor node to be known each time the data is updated. It communicates with the Raspberry Pi using a standard serial protocol which could be read and then broken into the necessary components reading longitude and latitude using a Python script. The limitation to these modules was that they did not work inside of a building but given that the sensor nodes should be operating in an outdoor environment this should not be a major issue.

2.2 Network Model

This project combines using Zigbee with IoT to create a more flexible and more accessible network. It provides a way for the network to be accessed at anytime from anywhere with an internet connection and a compatible browser. The overall network uses Zigbee for short range communication between nodes to negate the need for a direct internet connection to all nodes. The central node in the Zigbee network is the only node that needs an internet connection to transmit data to the cloud storage, in this case Google Cloud. Multiple devices can upload to the cloud at the same time allowing for farther apart nodes to upload separately. This creates an overall network that all connects to Google cloud and is made up of subnetworks that use Zigbee.

2.2.1 Mesh Network - Zigbee

Zigbee is a mesh protocol that functions on the IEEE 802.15.4 standard and is useful for short range communication. Using the XBee modules in this project, connectivity has a maximum range of 60-1200 meters depending network interference and line-of-sight. The Zigbee protocol includes three types of devices: Coordinator, Routers, and End Devices. There is only one coordinator per Zigbee network, and it is the central device in the network where all communications within the network are sent to. Because all communications within a Zigbee network are sent to the Coordinator, bridges to other networks such as Wi-Fi and internet are set up at the Coordinator. Routers can communicate as well as pass data on from other devices in the network. End devices are strictly able to communicate with their parent node but unlike the other types of devices can go into a sleep mode to conserve power. Here is an example of the topology of Zigbee:

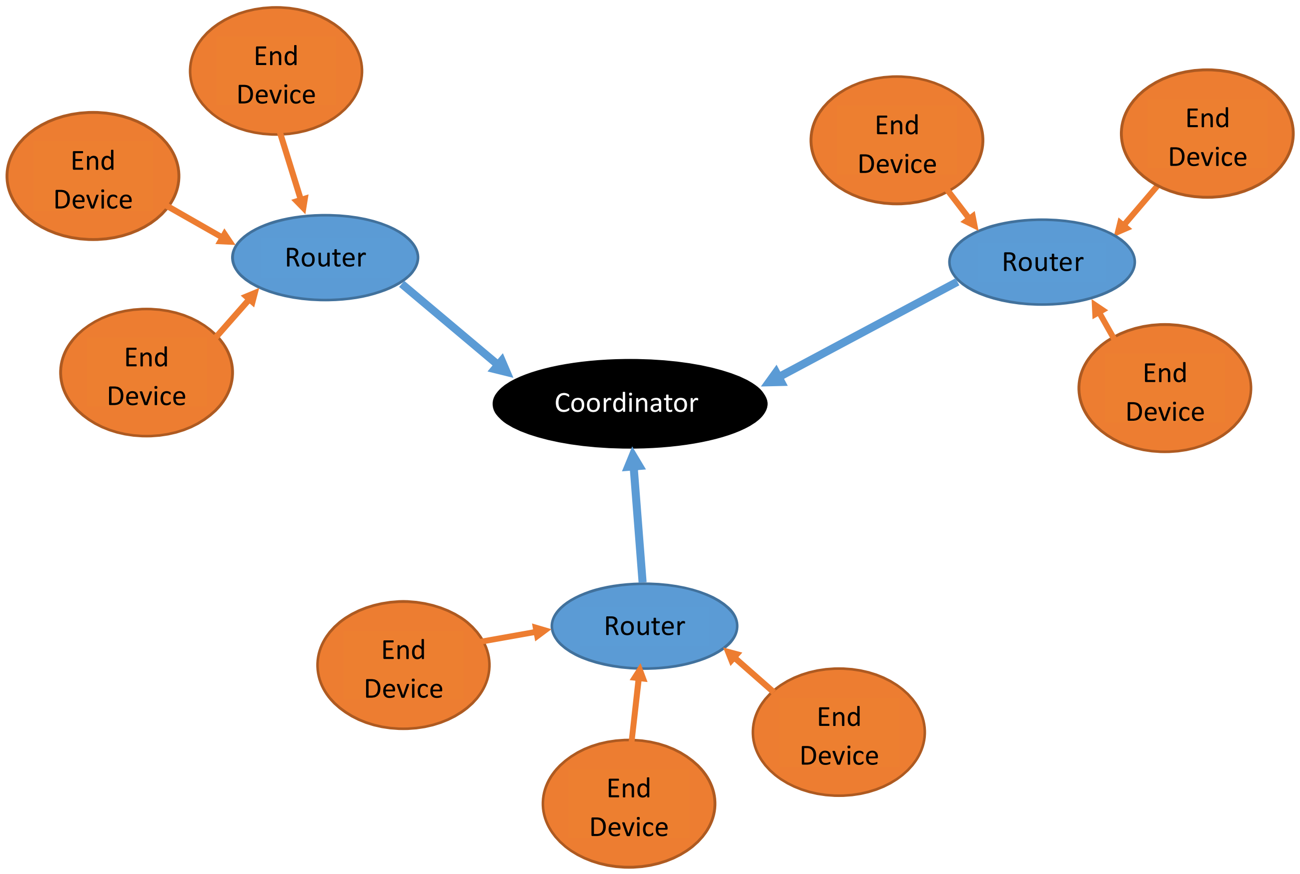


Figure 2.1 – Zigbee Network Topology

2.2.2 Cloud Storage Google Cloud

Google Cloud is a platform hosted on Google's servers that provides several API's for a variety of uses including data storage, data analysis, machine learning, web app hosting, and more. For the purposes of this project the Storage and App Engine features were used. The Storage feature was used to store the data in csv files where they could then be accessed by the App Engine which hosted a website to display the data on a map. The data was given as information attached to pins on the map showing the latest data uploaded. For the period of this project the Google Cloud trial was used which included a $200 account credit which allowed for some testing to be done.

2.2.3 Overall Network

As stated earlier the overall network combines Zigbee with IoT to create a flexible network. An example of the overall network topology is shown in Figure 2. A wi-fi connection is shown as a dashed line and a Zigbee connection is shown as a dotted line.

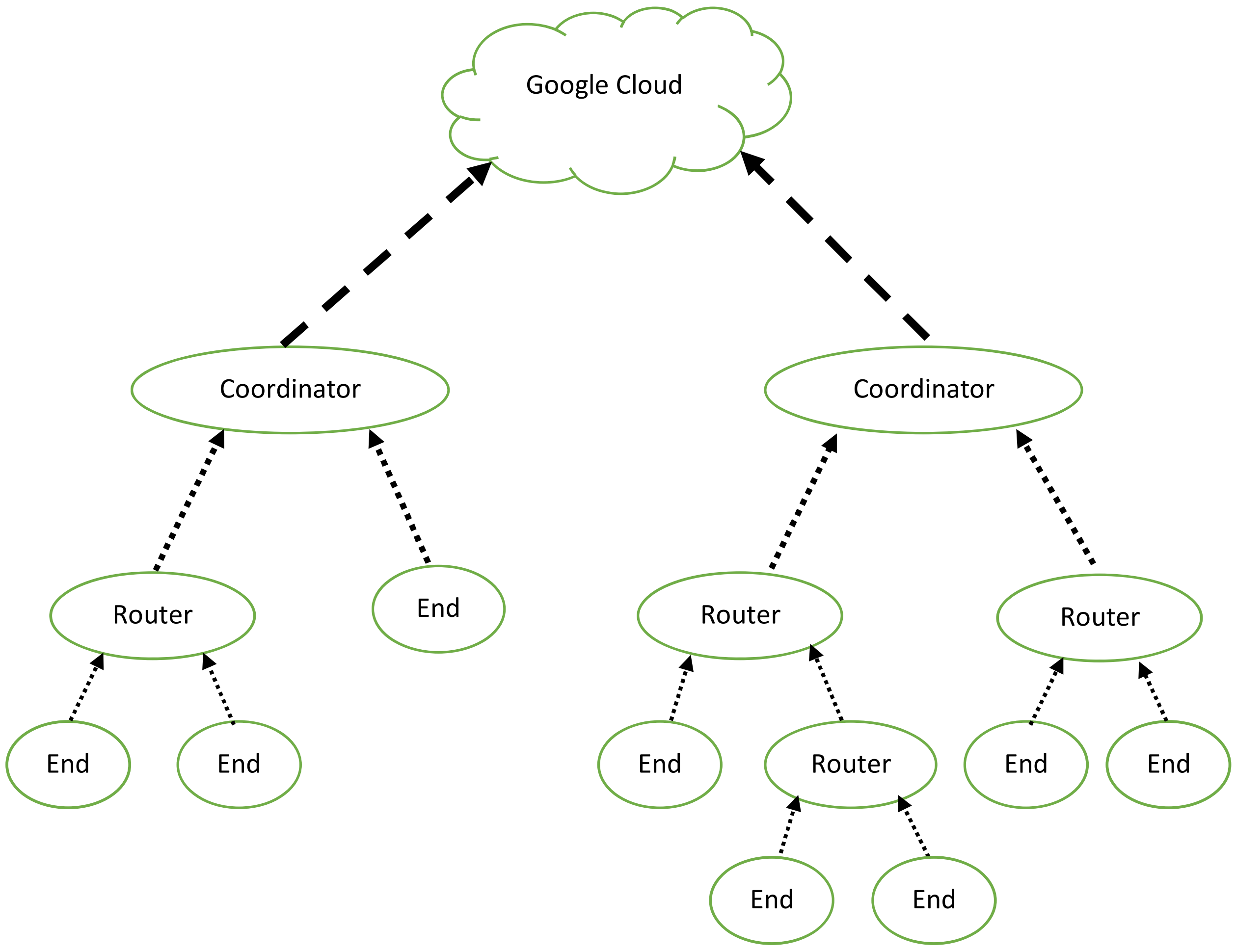


Figure 2.2 – Example Overall Network Topology

Part 3 – Setup and Programming

3.1 Node Setup

Over the course of this project multiple node setups were tested using different sensors. As discussed in 2.1.3 the Sense Hats were not ideal sensors so example setup discussed here will not include the Sense Hats but rather a setup including a Xbee module, air quality sensor, humiture sensor and a GPS module. The hardware connections are shown in Figure 3. The Xbee3 module was connected to the USB of the Raspberry Pi using the Grove Development Board. The Grove Air Quality Sensor v1.3 was plugged into the Grove Development Board allowing it to be connected through the Xbee3 module to the Raspberry Pi. The Sunfounder Humiture sensor was connected to the GPIO pins using 3 wires. The SIG wire was connected to GPIO 0, also known as GPIO 17 on the extension board, the VCC wire was wired to the 5V pin, and the GND wire was connected to a ground pin. The GPS module was connected via a USB port and power to the system is provided through the micro-USB port on the Raspberry Pi.

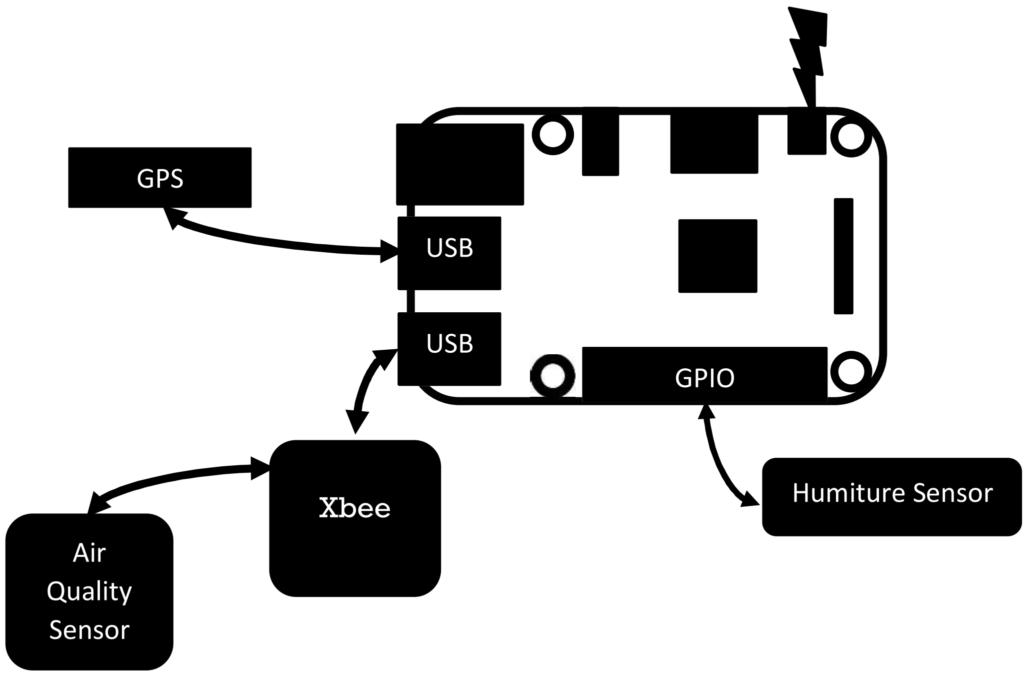


Figure 3.1 – Raspberry Pi Node Setup

3.1.1 Grove Air Quality Sensor Programming

Using the Xbee Python library, data could be read from the Grove Air Quality Sensor v1.3 in a few lines of code. The sensor is an analog sensor and as a result was plugged into an analog compatible IO port and the code put it in ADC mode. The function used in the final code to retrieve the value from the air quality sensor is shown below.

```
def air_quality(): #function reads data from air quality sensor
    #indicates the pin the sensor is connected to
    IOLINE_IN = IOLine.DIO1_AD1
    #defines the read mode of the connected device
    device.set_io_configuration(IOLINE_IN, IOMode.ADC)
    value = device.get_adc_value(IOLINE_IN)#reads data
    return value
```

3.1.2 Humiture Sensor Programming

After the humiture sensor is wired properly to the GPIO pins, it can be read from using the Adafruit_DHT python library. This kept the function for reading the sensor short and readable. The function used in the final code combined and read both the humidity and the temperature values at the same time, returning them in a format to be used in the csv file. The function used to read the humiture sensor values is shown below.

```
def read_Humiture():#reads humidity and temperature from DHT11
    while True:
        #using Adafruit_DHT library, reads data
        Temperature, humidity = Adafruit_DHT.read(Adafruit_DHT.DHT11,
17)

        #if statement checks for valid data

        if temperature is not None and humidity is not None:

            #returns valid data in csv compatible format

            return f"{temperature} C,{humidity} %"
```

3.1.3 GPS Module Programming

The DIYmall VK-162 GPS sent messages over serial that use the GPS standard NMEA. To access these messages from a Raspberry Pi, serial first had to be enabled in the terminal under the raspi-config. Once done the Raspberry Pi saved and rebooted. After the reboot, the GPS module was working, and a test was performed in the terminal to confirm its operation. To test it, the exact serial port was found, and a cat command was used so that raw data would then be listed out in the terminal. This confirmed the operation of the GPS modules. The function used to read from the GPS module is shown below. The code uses the serial library to read the NVMEA data for each line, splitting it to see each individual component which are separated by commas. The code searches for the $GPRMC line which contains the location data and then returns latitude and longitude components.

```
def get_Coord(): #function to retrieve GPS coordinates

    #define GPS port

    gps = serial.Serial("/dev/ttyACM0", baudrate = 9600)

    while True:
```

```
    line = gps.readline().decode("utf-8") #read line by line

    data = line.split(",") #split each line by its commas

    if data[0] == "$GPRMC": #finds the line starting with $GPRMC

        #filter latitude from line

        lat=data[3][0:2].lstrip('0')+"."+data[3][2:4]

        #filter longitude from line

        lng=data[5][0:3].lstrip('0')+"."+data[5][3:5]

        if data[4] == 'S': #add negatives based on direction

            lat = "-" + lat

        if data[6] == 'W':

            lng = "-" + lng

        coord = f"{lat},{lng}" #concatenate for csv format

        return coord
```

3.2 Network and Cloud Setup

As mentioned earlier this project used a combination of ZigBee and Wi-Fi to form a mesh network that could connect and upload information to google cloud. This setup required different code to setup example end, router, and coordinator devices. The coordinator includes code and a Wi-Fi connection to upload data to Google Cloud. Once the data made it to Storage on the cloud the data was accessed by the App Engine where additional coding read the data and displayed it on a map.

3.2.1 Configuring Xbee3 Modules

To begin using the Xbee3 modules they first had to be calibrated in the XCTU software. In XCTU, the Xbee3 modules can be set to one of two device roles, Join Network or Form network. Join Network is used for both router and end node configurations. The form network role is used only for the coordinator of a Zigbee network. To change a node from a router to an end node, the sleep settings must be changed to include a sleep mode. The final step in creating the network is to network identifier which is called the PAN ID. The PAN ID must be the same value for each Xbee3 module within a single ZigBee network. The Python 3 code for using the Xbee3 modules on the Raspberry Pi is covered in 3.2.2 and 3.2.3.

3.2.2 Router and Edge Node

For this project, the router and edge nodes used the same code. The difference in setup as mentioned in 3.2.1, is whether the sleep function is enabled for the Xbee3 module. To get a working router or edge node, the sensors were setup as shown in 3.1, and code was written using the Xbee Python library to handle delivering messages to the coordinator. The code below is the final code without the individual sensor functions to focus on the parts of the code dealing with the network communications.

```
#imports all used libraries
import time, serial
from datetime import datetime
from digi.xbee.devices import XBeeDevice, RemoteXBeeDevice,
XBee64BitAddress
```

```
from digi.xbee.io import IOLine, IOMode


device = XBeeDevice("/dev/ttyUSB0", 9600) #declare port used for Xbee
#set the coordinator address necessary for message destination
coord_address = XBee64BitAddress.from_hex_string("0013A200419819F8")
device.open() #open connection to Xbee
#declares the coordinator device used when sending messages
remote_device = RemoteXBeeDevice(device, coord_address)


while True:
     if datetime.now().minute%5 == 0: #send data every 5 minutes
        try: #try statement handles issues such as timeout errors
           # data is put in csv format
            data =
f"C1N2,{datetime.now()},{read_Humiture()},{air_quality()},{get_Coord()
}\n" #C1N2 functions as the ID and must be different for each node
            #send data to coordinator
            device.send_data(remote_device, data
        except:
            continue
```

3.2.3 Coordinator Node

The code for the coordinator node looks a bit different from the router and end node code shown in 3.2.2. This is because the coordinator node receives all the Zigbee messages. Puts them all into one csv file and then uploads them to the google cloud. It is important to note that for data to be uploaded to Google Cloud, the coordinator node must have a copy of the cloud credentials key. For more specifics on the code, the full coordinator code is given with comments below without the individual sensor functions to focus on the parts for Zigbee and Google cloud.

```
#general libraries
import os, time, serial, Adafruit_DHT
from datetime import datetime
#library for communicating with cloud
from google.cloud import storage
#libraries for operating xbee modules
from digi.xbee.devices import XBeeDevice, RemoteXBeeDevice,
XBee64BitAddress
from digi.xbee.io import IOLine, IOMode

#variables and declarations used for cloud
#set google cloud credentials
os.environ["GOOGLE_APPLICATION_CREDENTIALS"]=
"/home/pi/Documents/googlecredentials.json"
#create storage bucket variable
store = storage.Client()
buckets = list(store.list_buckets())
bucket = store.get_bucket("environmental-sensor-
network.appspot.com") #storage bucket declaration

#create the csv data file
```

```
csv_firstline = "ID,Date Time,Temperature,Humidity,Air
Quality,Latitude,Longitude\n"
csv = 'data.csv'
f = open(csv, 'w')
f.write(csv_firstline)
f.close()

#create Xbee device variable
device = XBeeDevice("/dev/ttyUSB0", 9600)
device.open() #open Xbee device connection

"""list for identifying other devices in Zigbee network, must
include all devices connected to the coordinator and each ID
must be unique"""
ID_list = ['C1N1', 'C1N2']

def recieve(ID): #function to get data from a specified device
    while True:
        try: #try statement ignores allows for timeout errors
            xbee_data = device.read_data(10) #recieve data
            xbee_data = xbee_data.data.decode("utf-8") #decode
            if(ID in xbee_data): #check to see if desired data
                return xbee_data #return desired data
            else: #otherwise start over and try again
```

```
                continue
        except:
            continue

#function to store data in the cloud at an alternative location
def archive():
    #creates new file on cloud with time stamp
    blob = bucket.blob(f"archive/{datetime.now()}{csv}")
    #uploads data from file
    blob.upload_from_filename("/home/pi/Documents/{csv}")
    #empty local csv file and write the first line headers to it
    f = open(csv, 'w')
    f.write(csv_firstline)
    f.close()

while True: #main part of the code
    if (datetime.now().minute%5) == 0:#runs code every 5 minutes
        #initialize data and put data from sensors in csv format
        data = ''
        data += f"C1,{datetime.now()},{read_Humiture()},
{air_quality()},{get_Coord()}\n"
        for i in ID_list: #gets data from each connected device
            data += recieve(i)#append all the data to one string
        f = open(csv, 'a')
        f.write(data) #appends the data string to the file
        f.close()
        #uploads the latest version of the data file
        blob = bucket.blob(f"data/{csv}")
        blob.upload_from_filename(f"/home/pi/Documents/{csv}")
        """Check the file size, if it is above ~5MB, then it
        will trigger the archive function to start over"""
        if os.path.getsize(f"/home/pi/Documents/{csv}") >
5000000:
            archive()
        time.sleep(60)
```

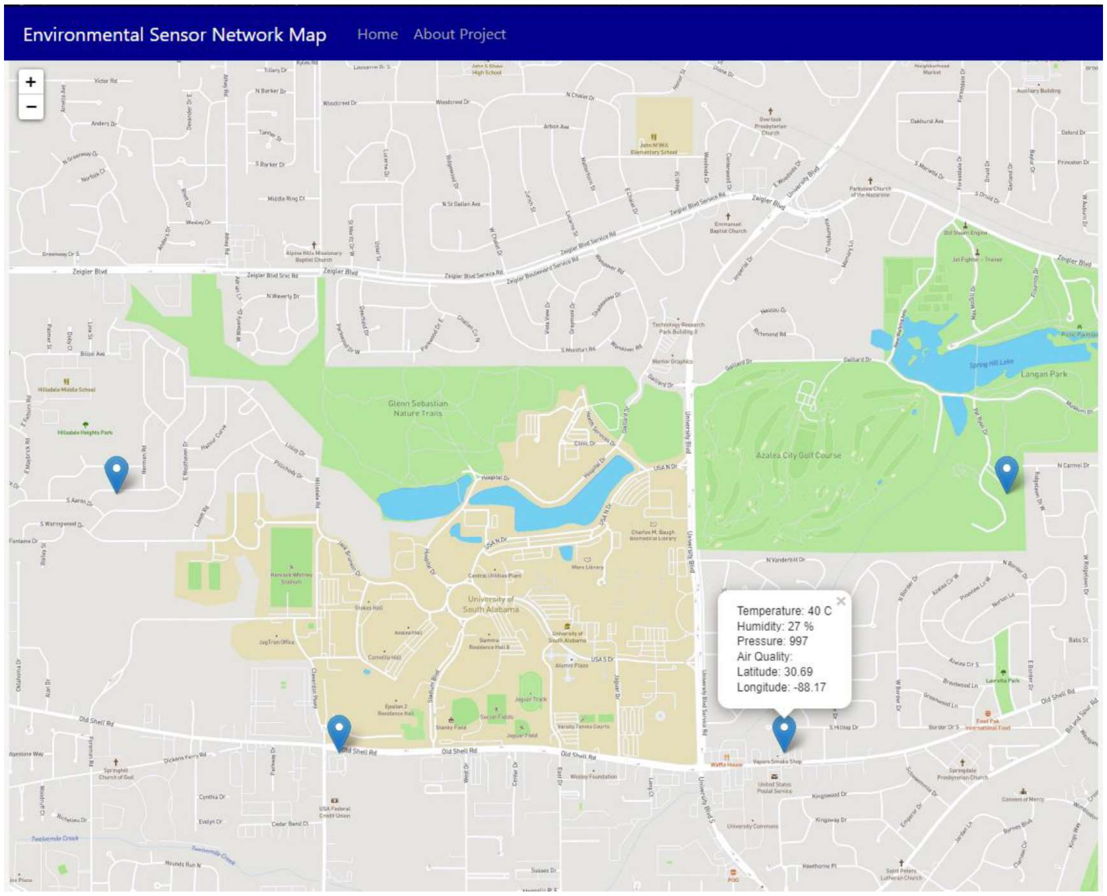
3.2.4 Cloud Setup

To setup the cloud, a Google account had to be used and linked, and as stated earlier, a trial account was made using Google Cloud to gain access to an account credit to help with learning their platform and testing the prototype system. After getting the account setup a project was created to start using all the features offered by Google Cloud. For this project, the only parts of Google cloud used were the Storage and the App Engine functionalities. The App Engine was first setup and then the default storage bucket from the App Engine was used. This was done as a cost saving measure since a free bucket is created with the App Engine. This also made the data more easily accessed by the code for the App Engine application which places the data on a map using pins. The App Engine itself acts to host a web-based application and in the case of this project functioned as a website.

To upload data to storage credentials first had to be created that would allow the Raspberry Pi access to the cloud storage bucket. This was done by creating new credentials using the API & Services section of google cloud. A service account was created here and given the role Storage Admin. This would allow any device with the credentials for this service account to edit, read, and create new files in Storage. After creating this role, a key was created and added to the Raspberry Pi where it could be easily accessed by the coordinator.py code. This file is only necessary on nodes connecting directly to the internet to upload to Google Cloud. The code sets the credential file as environment variable and uses the Google Cloud Storage python library to edit and upload files. This code can be found in Appendix A.

The App Engine required multiple files to get up and running, all of which can be found in Appendix B. The web application used Python and a library called Flask to run an html and css based website. The app.yaml and the requirements.txt files are necessary to get the app engine running as these specify the parameters of the runtime environment. The main.py file is the file ran by the runtime environment which using Flask renders the html files. There are a few html files to divide up the website and it uses css to help render the website. The layout.html file sets the basic layout of the website which is used by all the other html files. The home.html provides the home page which features a map with pins that can be clicked on to view the most recent data from each node. To get the interactive map, a free service called OpenStreetMap was utilized. The about.html is a very simple page that would include information about the project but was never completed and simply says "Content Coming Soon." In order to give the web application access to the data csv files, the files were changed to allow public read access. This simplified the process allowing the files to be read directly with no problem. Since these do not contain private information, the files having public read access was not a concern. The code for the web application can be run locally by running the main.py. This allowed testing to be done to ensure functionality. Once the web application was ready it was uploaded to App Engine using the Google Cloud SDK.

The final web application could be accessed from anywhere with a compatible browser and an internet connection using the web link provided by the App Engine. An example of what the home page looks like with the map and pins is shown in Figure 4. The pins in Figure 4 are located at dummy coordinates on the map meaning that these were just example coordinates in the Mobile area rather than locations where sensor nodes where actually deployed. Each pin can be clicked on to show the most recent data uploaded from the node at that location.



Temperature: 40 C
Humidity: 27 %
Pressure: 997
Air Quality:
Latitude: 30.69
Longitude: -88.17

Figure 3.2 – Example of Sensor Network Map

Part 4 – Testing and Results

4.1 Final Test Setup

To test the design of the Environmental Sensor Network, a setup using 4 nodes was used. The setup included 3 nodes all connected in a single ZigBee network and a node that was setup separated from the other 3. The 3 in the ZigBee network included one coordinator node and 2 router nodes. The router nodes reported data to the coordinator node and the coordinator node sent the data to the Google Cloud Storage. The 4th node was setup on its own to upload data separately to Google Cloud. It ran a modified version of the coordinator code that did not include receiving data from any other nodes. Each node was given a set of environmental sensors to report from. The network was then setup in the lab for an extended period of time, approximately 2 weeks, to ensure its functionality. During this test the nodes, cloud storage, and App Engine web application would be checked periodically (at least once every 2-3 days) to confirm that they were all working properly.

4.2 Final Results

The test successfully proved that the sensor network functioned as intended. Over the 2-week period the nodes all appeared to be operating properly. When checking their csv files and the files on the cloud, the data was being reported to the proper location every 5 minutes as intended. Each time 5 minutes hit the data would be updated in the cloud. When the current data file reached the approximately 5MB size limit, the file was promptly saved to the archive location and then a new empty file would start ion the main file location. It is important to note that the App Engine web application left some room for improvement, but it did function properly. There are 2 things to note about the web application. The first is that a disadvantage that could be fixed in future work is that the data does not update on the map without manually refreshing the page.

Sadly, the only real way to fix this is to still refresh the page, but an improvement could be to program the application to make it automatically refresh. The second disadvantage was the latency of the data upload to the data being available on the map, the data sometimes took up to a minute to become accessible on the map itself. This was disappointing but is also not something that can be improved as the reasons for this include the internet speed as well as the traffic on Google Cloud's services. Overall, the network functioned as intended, and this prototype provides a model than can be used for Environmental sensing.

Part 5 – Future Improvements and Applications

5.1 Hardware Improvements

While the testing of the prototype setup was successful there are still places for the design to be expanded upon and even improved. The setup was tested for a relatively short period of time in an indoor environment; thus, a focus of future work should be to perform long term analysis on the functionality of the hardware and to explore how to prepare the setup to be used in an outdoor environment. The current setup requires there to be an external wall outlet as a power source at each node. Another focus of future work should be to minimize power consumption at each node and explore alternative sources of self-sufficient power such as solar power. This would allow nodes to be setup at any appropriate location without concern for an external power source. The final thing that should be explored is the use of additional and different sensors. It has proved to be useful to apply pre-calibrated sensors, however due to the limited number of sensors used, it is still important to find and implement sensors for specific applications. Two ways to approach this is to either find and implement sensors based on what will be most useful or based on an application specific set of needs. After these improvements, the sensor nodes should prove to be more flexible and more useful in future applications.

5.2 Network Improvements

For testing, the network proved to work as intended, but there are still places that the network could be improved. There are 2 main areas of improvement that should happen. The first issue is that it is highly likely that at some point it will be desirable to place a sensor node where there is no immediate network connection via Wi-Fi or ZigBee. To address this situation, it would be ideal to add another connectivity options using a cellular modem or hotspot connection. This would increase the flexibility of the network by allowing nodes to be setup anywhere that a cellular internet connection is available. The second issue is that a form of network security should be added to the network. This becomes relevant in real world applications as data integrity is constantly being threatened by hackers. While this may not always be a concern when collecting basic data, protection is necessary to prevent any tampering with the collected data.

5.3 Data Analysis

The final place for future work and improvements is to focus on the data analysis. Being able to collect data using sensor nodes at key locations is useful but it would be even more useful to have conclusions drawn from the data collected. In the case of this environmental monitoring setup that might be drawing conclusions about pollution levels, changes in weather, or possibly a long-term analysis of climate change. To do these things though it is necessary to take the collected data and use it to make informed conclusions. A useful method to approach this would be to perform machine learning on the collected data. Machine learning can be used to quickly find trends in the collected data that can lead to informed conclusions. There are 3 possible ways to implement this method of data analysis.

The first method is using a Raspberry Pi and a trained algorithm to draw initial conclusions using machine learning prior to the data reaching the cloud. This method means that only the more useful data conclusions need to be sent to the cloud also providing a benefit of minimizing the data transmitted. This also allows for less work to have to be done later to draw any additional conclusions. The second method is to use the raw data on the cloud and use machine learning in the cloud to draw conclusions there. This can still prove useful as it allows conclusions to be drawn from all the data collected from the entire network at once.

The third and possibly most useful method would be to implement both previous methods at once. Implementing machine learning on the sensor nodes will allow for initial conclusions to be drawn minimizing the amount of data being transmitted to the server and reducing the workload on the cloud end. Combining this with the use of machine learning on the cloud end will allow for more useful conclusions to be drawn using all the data collected from each node. This proposed method requires more initial work on creating the machine learning algorithms but should allow for faster, more efficient analysis to be performed.

5.4 Ending Thoughts

The designed prototype IoT-enabled monitoring network proved to function as intended and provides a framework for future work to build from. While there is still much work to be done prior to a full-scale implementation of this setup, this initial work shows a lot of potential. There are still multiple directions this project can go in the future and additional work will continue to fine tune and make this technology more applicable. The hope is that some day this project will become a useful sensor network with nodes that can be setup at any desirable location, collect useful data, and provide informed conclusions.

Appendix A

Node Python Files

RouterandEdgeNode.py

```
#imports all used libraries
import time, serial
from datetime import datetime
from digi.xbee.devices import XBeeDevice, RemoteXBeeDevice, XBee64BitAddress
from digi.xbee.io import IOLine, IOMode

device = XBeeDevice("/dev/ttyUSB0", 9600) #declare port used for Xbee
#set the coordinator address necessary for message destination
coord_address = XBee64BitAddress.from_hex_string("0013A200419819F8")
device.open() #open connection to Xbee
#declares the coordinator device used when sending messages
remote_device = RemoteXBeeDevice(device, coord_address)

def air_quality(): #function reads data from air quality sensor
    #indicates the pin the sensor is connected to
    IOLINE_IN = IOLine.DIO1_AD1
    #defines the read mode of the connected device
    device.set_io_configuration(IOLINE_IN, IOMode.ADC)
    value = device.get_adc_value(IOLINE_IN)#reads data
    return value

def read_Humiture():#reads humidity and temperature from DHT11

    while True:

        #using Adafruit_DHT library, reads data

        Temperature, humidity = Adafruit_DHT.read(Adafruit_DHT.DHT11, 17)

        #if statement checks for valid data

        if temperature is not None and humidity is not None:

            #returns valid data in csv compatible format
            return f"{temperature} C,{humidity} %"


def get_Coord(): #function to retrieve GPS coordinates

    #define GPS port

    gps = serial.Serial("/dev/ttyACM0", baudrate = 9600)

    while True:

        line = gps.readline().decode("utf-8") #read line by line

        data = line.split(",") #split each line by its commas

        if data[0] == "$GPRMC": #finds the line starting with $GPRMC

            #filter latitude from line

            lat=data[3][0:2].lstrip('0')+"."+data[3][2:4]

            #filter longitude from line

            lng=data[5][0:3].lstrip('0')+"."+data[5][3:5]

            if data[4] == 'S': #add negatives based on direction

                lat = "-" + lat

            if data[6] == 'W':

                lng = "-" + lng

            coord = f"{lat},{lng}" #concatenate for csv format

            return coord


while True:
     if datetime.now().minute%5 == 0: #send data every 5 minutes
        try: #try statement handles issues such as timeout errors
               # data is put in csv format
            data = f"C1N2,{datetime.now()},{read_Humiture()},{air_quality()},{get_Coord()}\n"
#C1N2 functions as the ID and must be different for each node
            #send data to coordinator
            device.send_data(remote_device, data
        except:
            continue
```

Coordinator.py

```
#general libraries
import os, time, serial, Adafruit_DHT
from datetime import datetime
#library for communicating with cloud
from google.cloud import storage
#libraries for operating xbee modules
from digi.xbee.devices import XBeeDevice, RemoteXBeeDevice, XBee64BitAddress
from digi.xbee.io import IOLine, IOMode

#variables and declarations used for cloud
#set google cloud credentials
os.environ["GOOGLE_APPLICATION_CREDENTIALS"]= "/home/pi/Documents/googlecredentials.json"
#create storage bucket variable
store = storage.Client()
buckets = list(store.list_buckets())
bucket = store.get_bucket("insert name of storage bucket") #storage bucket declaration

#create the csv data file
csv_firstline = "ID,Date Time,Temperature,Humidity,Air Quality,Latitude,Longitude\n"
csv = 'data1.csv'
f = open(csv, 'w')
f.write(csv_firstline)
f.close()
```

```
#create Xbee device variable
device = XBeeDevice("/dev/ttyUSB0", 9600)
device.open() #open Xbee device connection

"""list for identifying other devices in Zigbee network, must include all devices connected to
the coordinator and each ID must be unique"""
ID_list = ['C1N1', 'C1N2']

def air_quality(): #function reads data from air quality sensor
    #indicates the pin the sensor is connected to
    IOLINE_IN = IOLine.DIO1_AD1
    #defines the read mode of the connected device
    device.set_io_configuration(IOLINE_IN, IOMode.ADC)
    value = device.get_adc_value(IOLINE_IN)#reads data
    return value

def read_Humiture():#reads humidity and temperature from DHT11

    while True:

        #using Adafruit_DHT library, reads data

        Temperature, humidity = Adafruit_DHT.read(Adafruit_DHT.DHT11, 17)

        #if statement checks for valid data

        if temperature is not None and humidity is not None:

            #returns valid data in csv compatible format

            return f"{temperature} C,{humidity} %"



def get_Coord(): #function to retrieve GPS coordinates

    #define GPS port

    gps = serial.Serial("/dev/ttyACM0", baudrate = 9600)

    while True:
        line = gps.readline().decode("utf-8") #read line by line

        data = line.split(",") #split each line by its commas

        if data[0] == "$GPRMC": #finds the line starting with $GPRMC

            #filter latitude from line

            lat=data[3][0:2].lstrip('0')+"."+data[3][2:4]

            #filter longitude from line

            lng=data[5][0:3].lstrip('0')+"."+data[5][3:5]

            if data[4] == 'S': #add negatives based on direction

                lat = "-" + lat

            if data[6] == 'W':

                lng = "-" + lng

            coord = f"{lat},{lng}" #concatenate for csv format

            return coord


def recieve(ID): #function to get data from a specified device
    while True:
        try: #try statement ignores allows for timeout errors
            xbee_data = device.read_data(10) #recieve data
            xbee_data = xbee_data.data.decode("utf-8") #decode
            if(ID in xbee_data): #check to see if desired data
                return xbee_data #return desired data
            else: #otherwise start over and try again
                continue
        except:
            continue
```

Appendix B

App Engine Code

app.yaml

```
runtime: python37


env_variables:
  Bucket_name: "insert name of storage bucket with data"


handlers:
- url: /data
  static_dir: public


- url: /.*
  secure: always
  redirect_http_response_code: 301
  script: auto
```

requirements.txt

```
Flask==1.1.1

google-cloud-storage

gunicorn
```

main.py

```
from flask import Flask, render_template

import csv

import urllib.request


app = Flask(__name__)

file_list = ['data1.csv','data2.csv']

url = 'http://storage.googleapis.com/environmental-sensor-network.appspot.com/data/'

@app.route('/')

def home():

    ID_list = []

    datalist = []

    for name in file_list:

        file = urllib.request.urlopen(url + name)

        data = file.read().decode('utf-8')

        file.close()

        csv_reader = csv.DictReader(data.split('\n'), delimiter = ',')

        for row in reversed(list(csv_reader)):

            if row['ID'] in ID_list:

                break

            else:

                ID_list.append(row['ID'])

                datalist.append(row)

    return render_template('home.html', nodes = datalist)

@app.route('/about')

def about():

    return render_template('about.html')
```

main.css

```
body {
  background: #F5F5F5;
  color: #333333;
  margin-top: 3.5rem;
}

h1, h2, h3, h4, h5, h6 {
  color: #000000;
}

.bg-steel {
  background-color: #00008B;
}

.site-header .navbar-nav .nav-link {
  color: #A9A9A9;
}

.site-header .navbar-nav .nav-link:hover {
  color: #ffffff;
}
```

```
***layout.html***

<!DOCTYPE html>

<html>

</head>

    <!-- Required meta tags -->

    <meta charset="utf-8">

       <meta name="viewport" content="width=device-width, initial-scale=1.0">


    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.6.0/dist/leaflet.css"

integrity="sha512-

xwE/Az9zrjBIphAcBb3F6JVqxf46+CDLwfLMHloNu6KEQCAWi6HcDUbeOfBIptF7tcCzusKFjFw2yuvEpDL9wQ=="

crossorigin=""/>

    <script src="https://unpkg.com/leaflet@1.6.0/dist/leaflet.js" integrity="sha512-

gZwIG9x3wUXg2hdXF6+rVkLF/0Vi9U8D2Ntg4Ga5I5BZpVkVxlJWbSQtXPSiUTtC0TjtGOmxa1AJPuV0CPthew=="

crossorigin=""></script>


    <link rel="stylesheet"

href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-

Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">


    <link rel="stylesheet" type="text/css" href="static/main.css">


    <title>ESense Map</title>

<body>

    <header class="site-header">

      <nav class="navbar navbar-expand-md navbar-dark bg-steel fixed-top">

        <div class="container">

          <a class="navbar-brand mr-4" href="/">Environmental Sensor Network Map</a>

          <button class="navbar-toggler" type="button" data-toggle="collapse" data-

target="#navbarToggle" aria-controls="navbarToggle" aria-expanded="false" aria-label="Toggle

navigation">

            <span class="navbar-toggler-icon"></span>

          </button>

          <div class="collapse navbar-collapse" id="navbarToggle">

            <div class="navbar-nav mr-auto">
```

```
              <a class="nav-item nav-link" href="/">Home</a>
              <a class="nav-item nav-link" href="/about">About Project</a>
            </div>
          </div>
        </div>
      </nav>
    </header>
    <main role="main" class="container">
      <div class="row">
        {% block content %}{% endblock %}
      </div>
    </main>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js"
integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"
crossorigin="anonymous"></script>
</body>
</html>
```

home.html

```
{% extends "layout.html" %}


{% block sidebar%}
<h3>Map Key</h3>
<p class='text'>
  <ul class="list-group">
    {% for key in mkey %}
      <li class="list-group-item list-group-item-light">{{ key }}</li>
    {% endfor %}
  </ul>
</p>
{% endblock sidebar %}


{% block content %}
<div id="mapid" style="width: 1920px; height: 1080px;"></div>
<script>


        var mymap = L.map('mapid').setView([30.69, -88.17], 13);


        L.tileLayer('https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token=pk.eyJ1
IjoibWFwYm94IiwiYSI6ImNpejY4NXVycTA2emYycXBndHRqcmZ3N3gifQ.rJcFIG214AriISLbB6B5aw', {
                maxZoom: 18,
                attribution: 'Map data &copy; <a
href="https://www.openstreetmap.org/">OpenStreetMap</a> contributors, ' +
                        '<a href="https://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>, '
+
                        'Imagery © <a href="https://www.mapbox.com/">Mapbox</a>',
                id: 'mapbox/streets-v11'
        }).addTo(mymap);


  {% for row in nodes %}
        L.marker([{{ row['Latitude'] }}, {{ row['Longitude'] }}]).addTo(mymap)
```

```
               .bindPopup("Temperature: {{ row['Temperature'] }}<br />Humidity: {{
row['Humidity'] }}<br />Air Quality: {{ row['Air Quality'] }}<br />Latitude: {{ row['Latitude']
}}<br />Longitude: {{ row['Longitude'] }}").openPopup();
  {% endfor %}

</script>
{% endblock content %}
```

about.html

```
{% extends "layout.html" %}

{% block content %}

    <h6>Content Coming Soon<h6>

{% endblock content %}
```

References

[1] Pickard, B., R., Baynes, J., Mehaffey, M., & Neale, A., C. (2015) *Translating Big Data into Big Climate Ideas.* The Solutions Journal, Volume 6, Issue 1.

[2] Shete, R., & Agrawal, S. (2016). *IoT based urban climate monitoring using Raspberry Pi* (Rep.). Melmaruvathur, IN: IEEE.

[3] Forcella, M. (2017). *Creating a mesh sensor network using Raspberry Pi and XBee radio modules* (Tech.). New Paltz, NY: State University of New York

[4] Parmar, J., Nagda, T., Palav, P., & Lopes, H. (2018) *IOT Based Weather Intelligence*. Mumbai, India: IEEE

[5] Cook, J., Oreskes, N., Doran, P., T., Anderegg, W. ,R., L., Verheggen, B., Maibach, E., W., Carlton, J., S., Lewandowsky, S., Skuce, A., G., & Green, S., A. (2016) *Consensus on consensus: a synthesis of consensus estimates on human-caused global warming.*: IOP

[6] Montes-González, D., Vílchez-Gómez, R., Barrigón-Morillas, J., M., Atanasio-Moraga, P.,   Rey-Gozalo, G., & Trujillo-Carmona, J. (2018) *Noise and Air Pollution Related to Health in Urban Environments.* Proceedings Vol. 2: MDPI

[7] Zhao, C. W., Jegatheesan, J., & Loon, S. C. (2015). Exploring IOT Application Using Raspberry Pi. *International Journal of Computer Networks and Applications Volume 2 Issue 1.*

[8] (2019) *Digi XBee3 RF Module Hardware Reference Manual* https://www.digi.com/resources/documentation/digidocs/pdfs/90001543.pdf : Digi International

[9] (July 9, 2019) *XBee Python Library Documentation* https://buildmedia.readthedocs.org/media/pdf/xbplib/latest/xbplib.pdf: Digi International

[10] Naik, N. (2016). *Connecting google cloud system with organizational systems for effortless data analysis by anyone, anytime, anywhere*. Edinburgh, UK IEEE.