

SCHOOL *of* BUSINESS AND TECHNOLOGY

Department of Engineering and Aviation Sciences

**The Design of a**

**Smart Fire Detection System**

**Israel Akinsoyinu**

**David Goslee**

**Dedrick McCoy**

Advisor: Dr. Zang

2/24/2019

The Design of a Smart Fire Detection System

By

Israel Akinsoyinu, David Goslee, Dedrick McCoy

Submitted to the Department of Engineering and Aviation Sciences in partial fulfillment of the requirements for the degree of Bachelor of Science in Engineering at the

UNIVERSITY OF MARYLAND EASTERN SHORE

Date: 2/24/2019

The author hereby grants the University of Maryland Eastern Shore permission to reproduce and distribute publicly, paper and or electronic copies of this document in whole or in parts.

Authors Israel Akinsoyinu, David Goslee, Dedrick McCoy

Signature

Date

Department of Engineering and Aviation Sciences

Certified by

**Table of Contents**

[1.1 Background/Motivation 9](#_Toc22542393)

[1.2 Objective: 12](#_Toc22542394)

[1.3 Design Requirements: 12](#_Toc22542395)

[1.4 Design Constraints 12](#_Toc22542396)

[1.5 Design Method (Approach) 12](#_Toc22542397)

[1.6 Standards 12](#_Toc22542398)

[2.1 System Description 13](#_Toc22542399)

[2.2 System Diagram 14](#_Toc22542400)

[2.3 System Functions 14](#_Toc22542401)

[3.1 Tasks 14](#_Toc22542402)

[3.2 Team Organization 16](#_Toc22542403)

[3.2.1. Responsibility of Team Member 1 (David Goslee). 16](#_Toc22542404)

[3.2.2. Responsibility of Team Member 2 (Israel Akinsoyinu). 16](#_Toc22542405)

[3.2.3. Responsibility of Team Member 3 (Dedrick McCoy). 16](#_Toc22542406)

[3.3 Timeline/Milestones/Delivery Plan 16](#_Toc22542407)

[4.1 Implementation of Task 1 Sensor Design. 18](#_Toc22542408)

[4.1.1. Implementation of Subtask 1.1 18](#_Toc22542409)

[*4.1.2.* *Implementation of Subtask 1.2* 18](#_Toc22542410)

[*4.1.3.* *Implementation of Subtask 1.3* 20](#_Toc22542411)

[*4.1.4.* *Implementation of Subtask 1.4* 20](#_Toc22542412)

[*4.1.5.* *Implementation of Subtask 1.5* 21](#_Toc22542413)

[*4.1.6.* *The Implementation of Subtask 1.6* 22](#_Toc22542414)

[*4.1.7.* *Implementation of Subtask 1.7* 23](#_Toc22542415)

[4.2 Implementation of Task 2 Wireless Sensor Network Design. 24](#_Toc22542416)

[*4.2.1.* *Implementation of Subtask 2.1* 24](#_Toc22542417)

[*4.2.2.* *Implementation of Subtask 2.2* 25](#_Toc22542418)

[*4.2.3.* *Implementation of Subtask 2.3* 25](#_Toc22542419)

[*4.2.4.* *Implementation of Subtask 2.4:* 26](#_Toc22542420)

[*4.2.5.* *Implementation of Subtask 2.5* 26](#_Toc22542421)

[*4.2.6.* *Implementation of Subtask 2.6* 28](#_Toc22542422)

[*4.2.7.* *Implementation of Subtask 2.7* 29](#_Toc22542423)

[4.3 Implementation of Task 3 Website Design 30](#_Toc22542424)

[*4.3.1.* *Implementation of subtask 3.1* 31](#_Toc22542425)

[*4.3.2.* *Implementation of subtask 3.2* 31](#_Toc22542426)

[4.4 Implementation of Task 4 Configure Raspberry Pi and Website 32](#_Toc22542427)

[4.5 Implementation of Task 5 Design Automated Emergency Phone Call 32](#_Toc22542428)

[4.6 Implementation of Task 6 Design SMS Message Data Base 32](#_Toc22542429)

[5. Conclusion (Discussion and Future Plans) 32](#_Toc22542430)

[Appendix 34](#_Toc22542431)

[A. Component Specs 34](#_Toc22542432)

[1. Specs of Arduino UNO 34](#_Toc22542433)

[2. Specs of Arduino NANO 34](#_Toc22542434)

[3. Specs of Raspberry Pi 3 34](#_Toc22542435)

[*4.* *Specs of MQ-2 Smoke Sensor* 34](#_Toc22542436)

[5. Specs of MQ-7 Carbon Monoxide Sensor 34](#_Toc22542437)

[6. Specs of DHT22 Temperature and Humidity Sensor 35](#_Toc22542438)

[7. Specs of 433 MHz RF Receiver 35](#_Toc22542439)

[8. Specs of 433 MHz RF transmitter 35](#_Toc22542440)

[9. Specs of 4-Channel Logic Level Converter 35](#_Toc22542441)

[10. Specs of Piezo Buzzer 35](#_Toc22542442)

[B. Source Code. 35](#_Toc22542443)

[1. Source Code of DHT22 Temperature and Humidity Sensor 36](#_Toc22542444)

[*2.* *Source Code of MQ-2 Smoke Sensor* 37](#_Toc22542445)

[*3.* *Source Code of MQ-7 Smoke Sensor* 38](#_Toc22542446)

[*4.* *Source Code of 433 MHz Transmitter.* 39](#_Toc22542447)

[*5.* *Source Code of 433 MHz Receiver.* 39](#_Toc22542448)

[*6.* *Source Code For Motion Detector.* 40](#_Toc22542449)

[C. *Project Code.* 42](#_Toc22542450)

[*1.* *Code of 433 MHz Transmitter and DHT22* 42](#_Toc22542451)

[*2.* *Code of 433 MHz Receiver and DHT22* 42](#_Toc22542452)

[*3.* *Code of MQ-7 and MQ-2 sensors.* 44](#_Toc22542453)

[*4.* *Code of 433 MHz Transmitter and All Sensors.* 45](#_Toc22542454)

[*5.* *Code of 433 MHz Receiver and All Sensors.* 47](#_Toc22542455)

**List of Figures**

[Figure 1: Simple Fire Detector. 9](#_Toc22542886)

[Figure 2: Fire Detection Mini Project. 10](#_Toc22542887)

[Figure 3: Nest Smart Detector. 10](#_Toc22542888)

[Figure 4: One link smart detector. 11](#_Toc22542889)

[Figure 5: System Flow Chart. 14](#_Toc22542890)

[Figure 6: Adafruit DHT22 breadboard schematic. 18](#_Toc22542891)

[Figure 7: Adafruit DHT22 circuit schematic. 18](#_Toc22542892)

[Figure 8: First Flying Fish MQ-7 breadboard schematic. 19](#_Toc22542893)

[Figure 9: Second Flying Fish MQ-7 breadboard schematic. 20](#_Toc22542894)

[Figure 10: Flying Fish MQ-2 breadboard schematic. 20](#_Toc22542895)

[Figure 11: Complete Node Breadboard Schematic. 21](#_Toc22542896)

[Figure 12: Complete Node Circuit Schematic. 21](#_Toc22542897)

[Figure 13: PBC design for transmitter 22](#_Toc22542898)

[Figure 14: PCB design for receiver. 23](#_Toc22542899)

[Figure 15: Motion Detector Breadboard Schematic. 23](#_Toc22542900)

[Figure 16: Motion Detector Circuit Schematic. 24](#_Toc22542901)

[Figure 17: Breadboard Schematic for Geekcreit RF Transmitter. 24](#_Toc22542902)

[Figure 18: Circuit Diagram for Geekcreit RF Transmitter. 25](#_Toc22542903)

[Figure 19: Breadboard Schematic for Geekcreit RF receiver. 26](#_Toc22542904)

[Figure 20: Circuit Diagram for Geekcreit RF Receiver. 26](#_Toc22542905)

[Figure 21: Geekcreit 433 MHz RF transmitter output. 26](#_Toc22542906)

[Figure 22: Complete RF transmitter breadboard schematic. 27](#_Toc22542907)

[Figure 23: Complete RF transmitter circuit schematic. 27](#_Toc22542908)

[Figure 24: Complete RF receiver breadboard schematic. 27](#_Toc22542909)

[Figure 25: Complete RF receiver circuit schematic. 28](#_Toc22542910)

[Figure 26: Configuration of Raspberry pi and Arduino with logic level converter. 28](#_Toc22542911)

[Figure 27: Configuration of Raspberry pi and Arduino using direct USB connection. 29](#_Toc22542912)

[Figure 28: Terminal output from raspberry pi. 29](#_Toc22542913)

[Figure 29: Website output from Raspberry Pi 30](#_Toc22542914)

[Figure 30: Python code that calls espeak 30](#_Toc22542915)

[Figure 31: Heroku Main Terminal 31](#_Toc22542916)

[Figure 32: ClearDB MySQL Database 31](#_Toc22542917)

[Figure 33: Building Layout with Grid Overlay 32](#_Toc22542918)

[Figure 34: Local Phone book on Raspberry Pi 32](#_Toc22542919)

**List of Table**

[Table 1. Project Timeline and Delivery Plan 16](#_Toc8036850)

**Abstract**

By the end of the project, summarize the project into short text and put here.

1. **Introduction**

This project will be the design, development, and implementation of a smart fire detection system. This system will utilize modern sensor and network technologies to help notify firefighters and building occupants of an impending or life threatening situation.

## Background/Motivation

This project is the design of a smart fire detection system, with the consideration that there are other systems that already exist. The goal is to make improvements to these devices and add features that set this product apart from the others. With this design the goal is to take out all the issues seen previous systems and to make it user friendly as possible. This will allow individuals to have a thorough understanding of the product and its functions. The design of this product will include a smoke sensor, carbon monoxide sensor, and a temperature sensor. The system will be connected to a webpage that will display the data that is read from each of the sensor nodes in real time. When a problem is detected the webpage will display what the cause of the alarm is and highlight the area of the building in which the problem is located. With this design the plan is to have an SMS message that will inform building occupants and dispatch of what the problem is and where it is located within the building. This will then activate the alarm system and the webpage will display possible escape routes for individuals that are in the structure at that time. This system will be controlled by a Raspberry Pi that will use a built in program call PIVLO that will generate and autonomous phone call to 911 operators that will let them know where the problem is located in the building and the exact nature of the problem at hand. The sensor data will be transmitted via a node network known as a wireless sensor network (WSN). This feature will allow the system to retain its functionality as an alarm based system even if the power is out and there is no WIFI connection.

Upon researching other designs that related to this particular idea, one found an idea for a mini fire detection system on a website called newton projects. It detected potential fire threats using a thermistor, with a simple led that flashes as an alert and a piezo buzzer for the alarm notification. This idea was a great starting point because it presented a good idea but left a lot to be desired. One looked at this mini project and thought of how to make this simple detection system more elaborate and something that can actually serve as a replacement for devices that are already installed.



Figure 1: Simple Fire Detector.



Figure 2: Fire Detection Mini Project.

Upon doing further research one found another system on Amazon called the nest smart detector. This smart detector uses a photoelectric sensor and a split-spectrum sensor to detect potential fire threats. This gave the project team the first idea to use a photoelectric sensor in our earlier design plan, which was later scrapped due to the dramatic increase in the cost of the overall system. The objective of this project is to keep the design within affordable price range. This means that one does not want to exceed the cost of existing detectors on the market. It was found that cheaper sensors exist that offer the same measurement precession for half the price. The Nest smart detector is equipped and offered to customers with its own app, which informs the individual who owns it if a problem is detected. This idea was very interesting and the team decided to add this as a feature for this particular project. However, it was realized that this was actually not needed and in fact was overkill for this particular project. In turn a decision was made to alert occupants in the form of an SMS message. The processes of the SMS message transmission will be done using a software called PLIVO. The nest system had a built in function within the app that allowed the user to shut the detector off in case of a false alarm. With this consideration taken into account it could be something this smart fire detection system could encounter. Thus, an executive decision was made afford the users of this system the ability to turn of the system if a false alarm occurs.



Figure 3: Nest Smart Detector.

One found another similar system on Amazon. It is the Alexa enabled smoke detector and CO monitor that uses a photoelectric sensor, compatible with both Apple and Android devices. The price range is more excessive than what this project plans to charge. That particular monitor is listed around 180 dollars, where this system will only cost around 100 dollars. This being one of the most expensive systems on the market, one compared this detector to the detector mentioned earlier which was the Nest Smart Detector to find a baseline for the price range of where this product should fall.



Figure 4: One link smart detector.

The inspiration behind this project stemmed from the background experience from two of its group members. Two out of the three group members are volunteer firefighters and have direct experience in the field. This led to the thought, what would a first responder want to see and what one believe could make the response time quicker, safer and more efficient. One also employed the ideas of other first responders to see what their thoughts on this device were. This product aims to limit casualties of civilians as well a fellow firemen. One has witnessed casualties caused by flashovers, and other dire situations caused by high temperatures. By having the system relay the temperature to the building occupants and dispatch, the goal is to greatly decrease casualties caused by flashovers. There are many signs of a flashover in which firefighters are thought to be aware of but inevitably due to human error these can be missed. With this product the objective is to have a drastic effect on that. This system will inform dispatch of the location of the threat in the building and floor it is located on. This will afford first responders the ability to pinpoint and isolate their plan of attack without having to assess the situation on scene.

When conducting research on this particular product idea one realized that people often do not have the same reaction to fire alarms, based on the simple premise that they think it is either system testing or a false alarm. This in turn causes individuals to sit idle and wait for the reactions of others. The issue with that particular situation is not the alarm system but the psychological component of the civilians themselves. The plan of this project is to eliminate this component by having a SMS message that will be sent to the occupants of the building. This SMS message will inform the building occupant’s whether or not the alarm is a test or an actual threat. Another hurdle is the problem that can arise when individuals are trying to exit the structure as soon as possible. This project plans to attach this problem by having a website that will show individuals possible escape routes, corresponding to their location within the building. The overall objective of our system is the safety and protection of life for both the civilians and first responders.

Another reason for the creation of this product is one realized the inherent need in fire detection and co monitors, between January 1, 2019 and March 4, 2019 a total of 427 civilian home fire fatalities were reported by the U.S. media, not to mention the fatalities that were not on record. One knows that they cannot completely eradicate the issue but with this system the overall goal is to have an impact on that number and have it decrease it in the future. In recent systems have improved and afforded a decrease in the number of casualties. However, the number of casualties are still too high and one must not be complacent with the systems currently in place. This is one of the main drivers for the idea for the design of a smart fire detection system, to have an impact as in the local and national community.

## Objective:

The project objective is to design a smart fire detection system that can be implemented for commercial and in home use this system will be able to determine the fires location, notify 911 operators of the fire location, and alert building occupants of the potential threats location.

## Design Requirements:

1. The system will have an autonomous 911 call that will notify dispatch of the fire and its location within the building.
2. System should monitor the temperature of the fire and determine if it is at flash point.
3. The system will upload sensor information onto webpage and highlight area of potential threat within the building.
4. The system will monitor carbon monoxide levels and alert occupants if level is above acceptable threshold.
5. Send out a SMS message to registered occupants of the structure of the fire and its location.
6. Must provide output signals for PPE systems.
7. In home fire detector should mount to existing detectors base plate.

## Design Constraints

1. A single detector must not cost more than 120 dollars.
2. Original system must have voice alarm notification for building occupants.
3. Wireless nodes must be no more than 60 feet apart.

## Design Method (Approach)

The first step for this design method is to select the appropriate sensors for smoke, carbon monoxide, and temperature detection. The second step is to program each of these sensors for their perspective measurement thresholds and if these thresholds are passed issues a warning signal. The third step is to interface these sensors with a microcontroller that will drive each of these sensors processes and transmit their data to the CPU. The fourth step in this project is to program the RF transmitter, RF receiver, and logic level converter for the interface between the microcontroller and raspberry pi. The fifth step is to download PIVLO onto the raspberry pi and program the pi to make autonomous phone calls to emergency services. The sixth step is to implement reference ID’s for each node and assign them to their proper autonomous emergency call. The seventh step is to create a website that will allow the building administrator to see a virtual map of the building and location of where a potential problem will be located. The eighth step will be to again use PIVLO to send SMS warning messages to all registered building occupants of potential problem. The tenth step is to create a printed circuit board that will house contain the sensors, alarm module, and microcontroller. The eleventh step is to design and print a 3D housing that will contain the PCB. The twelfth step is to connect raspberry pi with display and test the administrative web page. Finally the thirteenth step is to interface the entire project and test the system as a whole.

## Standards

1. The initiation function provides the input signal to the system. (NFPA 101, 9.6.1.7).
2. The notification function is the means by which the system advises that human action is required in response to a particular condition. (NFPA 101, 9.6.1.7).
3. The control function provides outputs to control building equipment to enhance protection of life. (NFPA 101, 9.6.1.7).
4. Occupant notification shall be by means of voice announcements. (NFPA 101, 9.6.1.7).
5. Fire alarm circuits shall be installed in a neat workmanlike manner. (NFPA 70 Art. 760.24).
6. Cables and conductors installed exposed on the surface of ceilings and sidewalls shall be supported by the building structure in such a manner that the cable will not be damaged by normal building use. (NFPA 70 Art. 760.24).
7. All fire alarm drawings shall use symbols described in NFPA 170, Standard for Fire Safety and Emergency Symbols.
8. With every new system, a documentation cabinet shall be installed at the system control unit or at another approved location at the protected premises.
9. Smoke detector must be replaced every ten years in accordance to Maryland Smoke Alarm Law.
10. The smoke alarm requirements for existing older homes are based upon when the house was built.
11. The Law heavily emphasizes the use of sealed smoke alarms with long life batteries and silence/hush buttons.
12. Any new home in Maryland constructed after January 1, 1989 required at least one hardwired electric smoke alarm on every level of the home, including the basement.
13. The units must be interconnected in order that activation of any one of the required smoke alarms resulted in the sounding all of the required smoke alarms.
14. The time from the detector’s issuing a fire alarm signal to the controller’s receiving should be controlled within the 10 seconds (GB4717-2005 Fire Alarm Control Units).
15. If any module in the system fails, fire alarm controller should detect the fault in 100 seconds (GB4717-2005 Fire Alarm Control Units).
16. **Project Description**

## System Description

The fire detector is going to consist of three core sensors, a temperature sensor, carbon monoxide sensor, and an ionization sensor. The sensor data will be transmitted over a wireless sensor network made up of a series of RF transmitters and receivers. The sensors. RF transmitter, and RF receivers will be controlled by a microcontroller (Arduino). The detector will transmit sensor data to a RF receiver that will be connected to a logic level converter. This logic level converter will step the microcontroller’s voltage up from 3.3 volts to 5 volts. This will allow communication between the microcontroller and microprocessor (raspberry pi). The raspberry pi will process the data contiguously and if a problem is detected will use a program called PLIVO to transmit an autonomous emergency call that is specific to the ID of the sensor that identified a problem. The raspberry pi will transmit a warning message to any numbers programmed in the occupant network alerting building occupants of the potential problem and its location. The microprocessor will also send a warning signal back through the system and activate an alarm in the detector.

## System Diagram

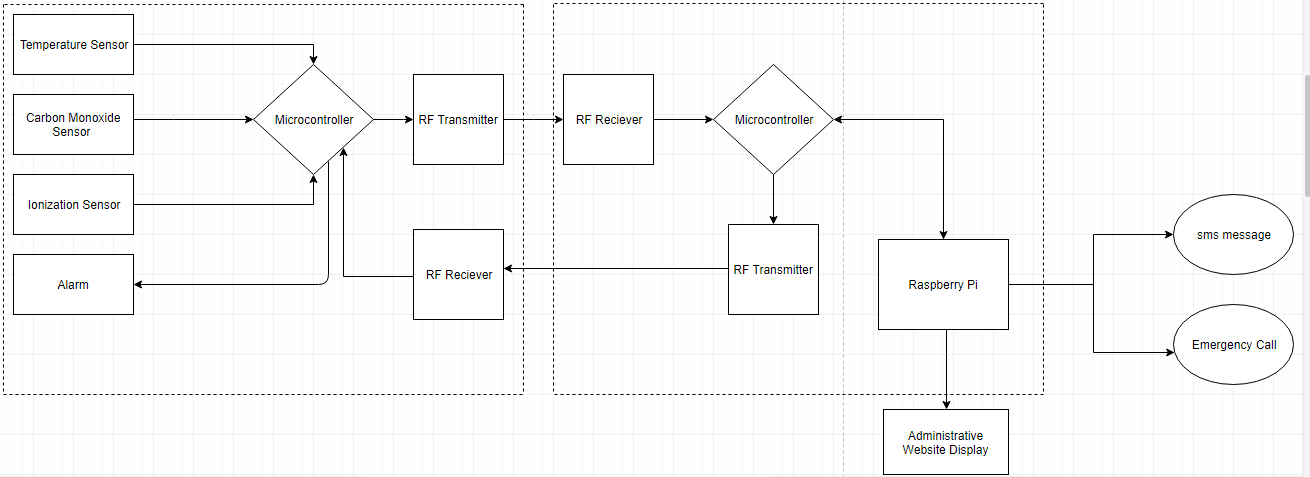


Figure 5: System Flow Chart.

## System Functions

1. When either of the sensor thresholds is tripped it will transmit a message signal that will be processed by the Arduino.
2. The Arduino will process this signal and transmit it via the RF transmitter module and also activate a Piezo buzzer as an instant alert notification.
3. This will signal will be received by the RF receiver that will pass this information through a second Arduino and then pass the signal through a logic level converter.
4. The signal will then be passed to the raspberry pi which will interoperate the data signal, determine its node ID, and determine the exact nature and origin of the problem within the building.
5. This will then instantiate the PLIVO program and transmit an autonomous phone call to emergency services notifying them of the nature and origin of the problem within the structure. While this process is occurring the raspberry pi will send out a SMS warning message to all registered occupant phone numbers within the database.
6. The potential threat will be updated to the administrative web page, its location within the building will be highlighted, and possible escape routes will be calculated and displayed.
7. **Implementation Plan**

## Tasks

* Task 1. Sensor Design
  + Subtask 1. Design program for Temperature Sensor.
  + Subtask 2. Design program for Ionization Sensor.
  + Subtask 3. Design program for Carbon Monoxide Sensor.
  + Subtask 4. Configure microcontroller with all three sensors.
  + Subtask 5. Design PCB for transmitter.
  + Subtask 6. Design PCB for receiver.
  + Subtask 7. Design and Configure Motion detector sensor.
* Task 2. Wireless Sensor Network Design
  + Subtask 1. Configure microcontroller and RF transmitter interface.
  + Subtask 2. Program microcontroller and RF transmitter.
  + Subtask 3. Configure microcontroller and RF receiver interface.
  + Subtask 4. Program microcontroller and RF receiver interface.
  + Subtask 5. Design program that will optimize bandwidth to hold all sensor data.
  + Subtask 6. Configure microcontroller and raspberry pi.
  + Subtask 7. Configure raspberry pi to retrieve, display, and store node data.
  + Subtask 8. Design a program that will interpret node data and transfer it to the webpage.
  + Subtask 9. Configure raspberry pi and website for node data processing and storage.
  + Subtask 10. Design voice alarm notification platform for building occupants.
* Task 3. Website Design
  + Subtask 1. Design background and layout of website.
  + Subtask 2. Design login for administrator and user access.
  + Subtask 3. Program a grid that will establish compartments for each room in building.
  + Subtask 4. Design a program that will interface the raspberry pi and website.
  + Subtask 5. Design a program that will pair node ID with corresponding room on the grid plan.
  + Subtask 6. Program grid to highlight room where potential threat arises.
* Task 4. Configure raspberry pi and website.
  + Subtask 1. Design program that will transmit and display WSN data on webpage.
  + Subtask 2. Design program that will display warning messages from pi on webpage.
  + Subtask 3. Configure node ID data from raspberry pi with website.
  + Subtask 4. Program alarm system shut off for administrative use.
  + Subtask 5. Program administrative link for admin to upload floor plan to website.
  + Subtask 6. Program website to receive and notify administrator when motion detector is tripped.
* Task 5. Design Automated Emergency Phone Call
  + Subtask 1. Setup Twilio account and install corresponding software on raspberry pi.
  + Subtask 2. Configure Twilio with wireless sensor network.
  + Subtask 3. Design a virtual autonomous alert message for each specific node ID.
  + Subtask 4. Design program that will allocate voice node ID numbers to specific automated dispatch message.
  + Subtask 5. Program each specific node ID for a specific SMS message transmission.
  + Subtask 6. Configure raspberry pi for SMS data transmission.
* Task 6. Design SMS Warning Message Database
  + Subtask 1. Design database for building occupant phone number storage.
  + Subtask 2. Design program to send link to the webpage.
  + Subtask 3. Configure SMS warning message with building occupant phone number database.
* Task 7. Design 3-D printed housings for transmitter and receiver modules
  + Subtask 1. Design cad model for transmitter housing.
  + Subtask 2. Design cad model for receiver housing.
  + Subtask 3. Print transmitter and receiver housings.
* Task 8. Complete System Assembly and Testing
  + Subtask 1. Assemble entire system.
  + Subtask 2. Test full platform with nodes placed in their desired positions.
  + Subtask 3. Test all warning message software and assure they work properly.

## Team Organization

### Responsibility of Team Member 1 (David Goslee).

Task 1, Subtask 1.1, 1.2, 1.3, 1.5

Task 2, Subtask 2.8, 2.9, 2.10

Task 4, Subtask 4.1, 4.2, 4.3, 4.4

Task 5, Subtask 5.3, 5.4

Task 8, Subtask 8.1, 8.2, 8.3

### Responsibility of Team Member 2 (Israel Akinsoyinu).

Task 1, Subtask 1.6, 1.7

Task 2, Subtask 2.1, 2.2, 2.3, 2.4, 2.6, 2.7

Task 5, Subtask 5.1, 5.2

Task 7, Subtask 7.1, 7.2, 7.3

Task 7, Subtask 8.1, 8.2, 8.3

### Responsibility of Team Member 3 (Dedrick McCoy).

Task 3, Subtask 3.1, 3.2, 3.3, 3.4, 3.5

Task 4, Subtask 4.1, 4.2, 4.3, 4.4, 4.5, 4.6

Task 6, Subtask 8.1, 8.2, 8.3

## Timeline/Milestones/Delivery Plan

**Table 1**. **Project Timeline and Delivery Plan**

|  |  |  |  |
| --- | --- | --- | --- |
| **Week Number** | ***David*** | ***Dedrick*** | ***Israel*** |
| **Week 1** | *Subtask 1.1-1.3:*  Design Program for DHT22, MQ-2, and MQ-7 | *Subtask 3.1-3.2:*  Design back ground and website logins for user and administrator. | *Subtask 2.1-2.4:*  Configure and Program RF transmitter and receiver with Arduino. |
| **Week 2** |
| **Week 3** | *Subtask 2.5:*  Design program to hold and transmit all sensor data. | *Subtask 2.6-2.7:*  Configure Arduino and Raspberry pi to receive and display node data. |
| **Week 4** | *Subtask 3.3:*  Program a grid that will establish compartments for each room. |
| **Week 5** | *Subtask 1.5:*  Design PCB for transmitter. | *Subtask 1.6:*  Design PCB for receiver. |
| **Week 6** | *Subtask 3.4:*  Design a program that will interface raspberry pi and website. |
| **Week 7** |
| **Week 8** | *Subtask 3.5-3.6:*  Design a program that will pair node ID’s with rooms and program gird to highlight potential threat. |
| **Week 9** | *Subtask 2.8-2.9:*  Design program that will interoperate node data and transfer it to webpage and configure raspberry pi and webpage. | *Subtask 5.1-5.2:*  Setup and Configure PLIVO with raspberry pi |
| **Week 10** |
| **Week 11** | *Subtask 5.3:*  Design program that will allocate node ID’s with specific automated calls. |
| **Week 12** | *Subtask 6.1:*  Design local database for occupant phone number storage. | *Subtask 4.1-4.2:*  Design program that will transmit and display node data on webpage and a program to display warning messages. |
| **Week 13** | *Subtask 6.2-6.3:*  Design a program to send building occupants the threat location within the building structure and a link to the webpage (cloud). |
| **Week 14** |
| **Week 15** | *Subtask 5.4-5.5:*  Configure Pi for SMS transmission and Program each node to specific SMS corresponding node location within the building (local). | *Subtask 4.3-4.5:*  Program administrative shut off for alarm system and link for floor plan upload. |
| **Week 16** |
| **Week 17** | *Subtask 2.10:*  Design Voice Alarm notification system for each node. | *Subtask 7.1-7.3:*  Design CAD models for transmitter and receiver housings. |
| **Week 18** | *Subtask 4.6:*  Program website to receive and notify administrator when motion detector is tripped. |
| **Week 19** | *Subtask 1.7:*  Configure and program motion detector for alarm detection nodes. |
| **Week 20** |
| **Week 21** | *Subtask 8.1-8.3.*  Full System assembly and testing. | *Subtask 8.1-8.3:*  Full System assembly and testing. | *Subtask 8.1-8.3:*  Full System assembly and testing. |
| **Week 22** |

1. **Implementation**

## Implementation of Task 1 Sensor Design.

### Implementation of Subtask 1.1

The implementation of subtask 1.1 is the design of program and circuit for a temperature sensor. The sensor that was chosen for this part of the project is the Adafruit DHT22 Temperature and Humidity Sensor. To implement this task first the Adafruit DHT source file must be downloaded and extracted into your project folder. It is imperative that this file be extracted into the proper folder so that the libraries can be utilized for the DHT22 program code. If this is not done properly all definition statements written for the DHT22 sensor will cause the Arduino IDE compiler to throw an error when attempting to upload the code to the board. If one is using a four pin DHT22 as exemplified in the figure 6 the far right pin is the ground pin, the far left pin is the 5V input pin, the pin directly to the right of the 5V pin is the data pin, and the last pin is a reference pin. If one is using a DHT22 with three pins. The far right pin will be the ground pin, the far right pin will be 5V input pin, and the middle pin will be the data pin. The DHT22 will be connected to digital pin seven on the Arduino board as exemplified below.

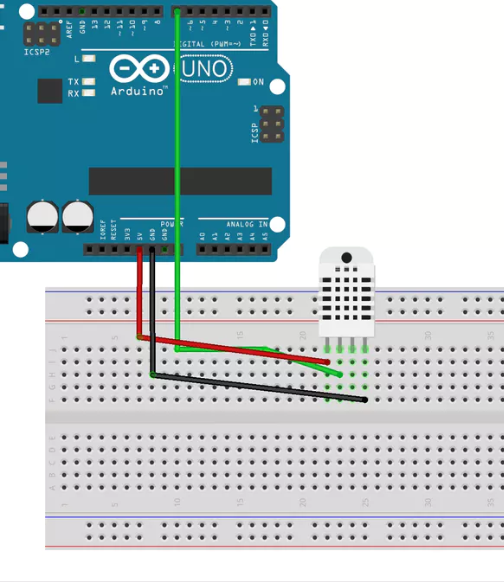


Figure 6: Adafruit DHT22 breadboard schematic.

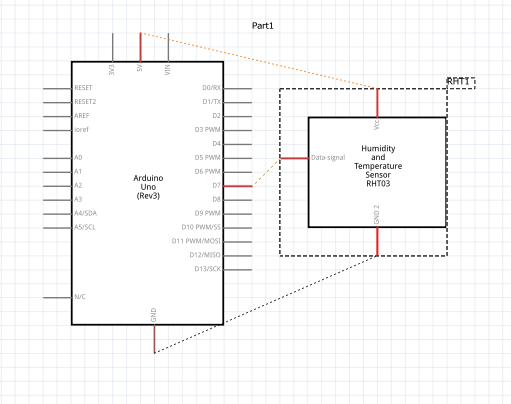


Figure 7: Adafruit DHT22 circuit schematic.

* + 1. ***Implementation of Subtask 1.2***

The implementation of subtask 1.2 is the design of a program and circuit for the flying fish MQ-7 carbon monoxide detector. This sensor was chosen for the implementation of this part of the project because of its precise calibration for the detection of harmful and poisonous gases. The MQ-7 fluctuates between a high and low voltage process that allows the device to clean its base plate before each reading thus affording a more accurate value for its user. No libraries are needed for this particular sensor only the program and circuit have to be built for complete operation. There are two ways to achieve this task the first is to build a complex circuit using capacitors, a NPN transistor, and multiple resistors to fluctuate the input voltage from 5V to 1.4V thus achieving the proper cleaning process for an accurate reading. The second is to hard code the voltage drop within the Arduino code itself. This will allow the microcontroller to fluctuate its own voltage over a specified range of time. The cleaning process needs to occur for a sixty second period and the reading must be taken directly after this process takes place to ensure that the device obtains an accurate reading. This sensor must be read using the analog pins of the Arduino because it does not contain a digital chip on its own board. The A0 pin on the Arduino must be utilized as a constant reference pin so that the device can differentiate between its high and low voltage fluctuations to achieve an accurate data reading. The complex circuit is exemplified by figure 8 and the simple circuit is exemplified by figure 9.

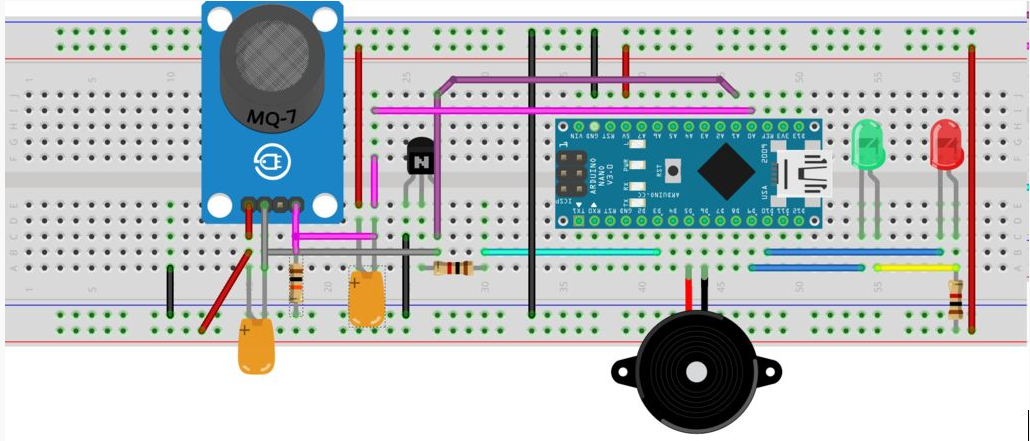


Figure 8: First Flying Fish MQ-7 breadboard schematic.

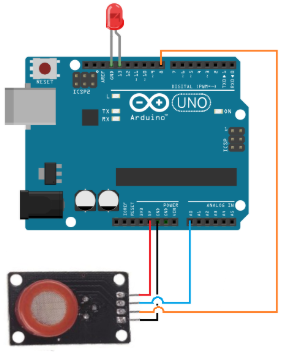


Figure 9: Second Flying Fish MQ-7 breadboard schematic.

* + 1. ***Implementation of Subtask 1.3***

The implementation of subtask 1.3 is the design of a program and circuit for the Flying Fish MQ-2 ionization sensor. This sensor was chosen because of its high accuracy for detecting air born particulate which is a major benefit for detecting smoldering fires. For this sensor no libraries are needed to ensure that the sensor functions properly only the code and a proper circuit need be implemented. The MQ-2 input must be read using the analog pins because there is no chip on its board to convert the reading into a digital signal. This sensor does not require any special calibration technique to achieve a proper measurement. To ensure that the code for this sensor is executing properly it is hardwired to a circuit with two LED’s and a Piezo buzzer. If the circuit is in normal operation the MQ-2 will begin reading and transmitting data and if no smoke is detected the Piezo buzzer will be silent and the green LED will stay lit. If the MQ-2 sensor threshold is tripped then the red LED will light and the Piezo buzzer will initiate a high frequency ring until the sensor gains a reading that is below the set threshold value. In the circuit exemplified in figure 10 the MQ-2 has four pins. The far left pin is the 5V input pin, the next pin directly to the right is the ground pin, the next pin directly to the right is the digital data pin, and the last pin is the analog data pin. For this particular project as mention in the above text the analog pin is being utilized for the MQ-2 and it is connected to Analog pin 5 on the Arduino board. The 5V input of the Piezo buzzer is connected to the Digital 10 pin, and the ground pin is connected to ground via a 110 ohm resistor. Likewise, the green LED is connected to Digital 11 pin and the Red LED is connected to the digital 12 pin of the Arduino board. Again both LED’s are connected to ground via 110 ohm resistors to stabilize their input signals.

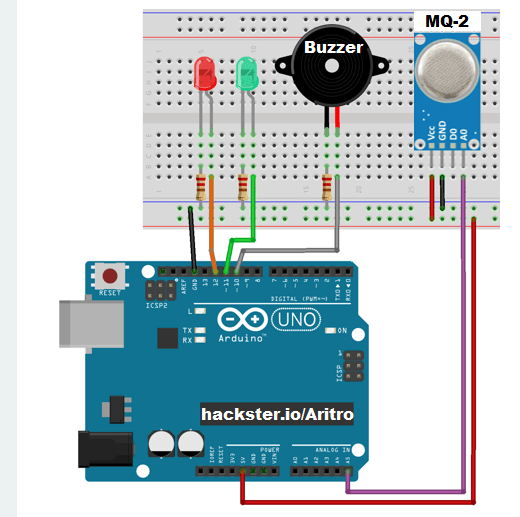


Figure 10: Flying Fish MQ-2 breadboard schematic.

* + 1. ***Implementation of Subtask 1.4***

The implementation of subtask 1.4 is the complete integration of all three sensors with the alarm circuit. The most critical component of this task is the proper integration of all code for each of these sensors into one contiguous block of code that functions for them all. For this the Adafruit libraries must be present in the project directory to ensure the definition statements for the DHT22 temperature and humidity sensor do not cause a compilation error. Careful consideration is needed for the integration of this circuit and it is imperative that all wires are connected to their proper input pins. As exemplified in the figure 11 the Piezo buzzer is connected to the digital 6 pin on the Arduino, and is grounded across a 110 ohm resistor. The MQ-2 is directly beside the DHT22 temperature and humidity sensor and is connected to the analog pin A2 on the Arduino board. The MQ-7 carbon monoxide sensor is connected to analog pin A5 on the Arduino board. Finally the DHT22 remains connected to the digital input pin 7 on the Arduino board.

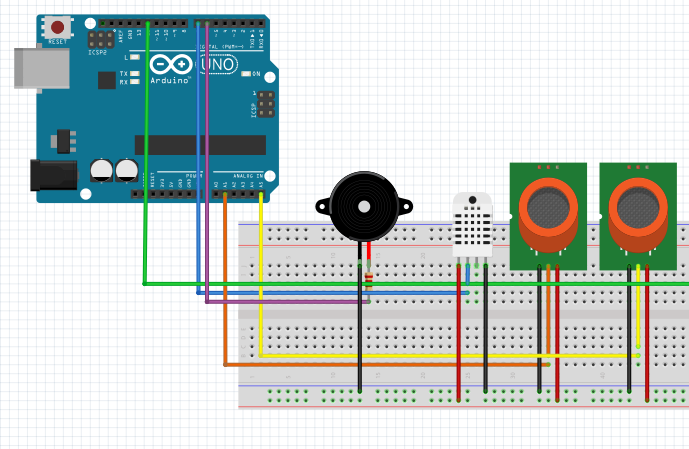


Figure 11: Complete Node Breadboard Schematic.

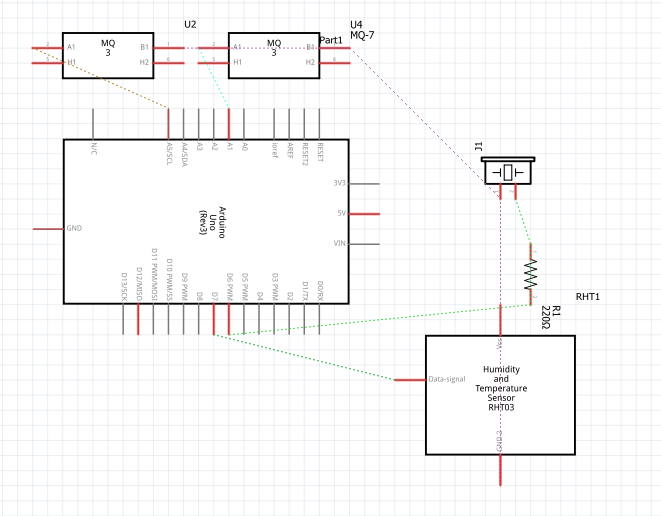


Figure 12: Complete Node Circuit Schematic.

* + 1. ***Implementation of Subtask 1.5***

The implementation of subtask 1.5 is the design of a printed circuit board for the transmitter for this project. This board will contain an Arduino Nano, MQ-7, MQ-2, DHT22, piezo buzzer, two LED’s, and a RF transmitter module. There are a multitude of free programs that one can use when designing a PCB. The top three choices are Fritzing, KiCad, and Eagle. For this particular project KiCad was chosen for as it is a more efficient platform for the implementation of custom built parts. The first thing to do is to find the official KiCad sit and initiate the download for your particular device (either MAC OS or Windows). Once this has been completed open the program and create a new project file and name it transmitter. Then click on the schematic view and start by adding the simplest elements that already exist within KiCad’s built in libraries. Those components would be the 110 ohm resistor, the piezo buzzer, Arduino Nano, and the LED’s. Once these have been inserted into the schematic it is time to create the custom sensors needed for this particular PCB. To achieve this click on the new symbol link in the schematic view. Once this is done one can create as many new symbols as they want. Use the data sheet and pin layout to create each corresponding sensor and then save them all in a new folder titled my sensors. After all custom sensors are created simply insert them into the schematic and wire the circuit accordingly. Now for the second step it is time to assign footprints to each of the elements within the transmitter schematic. To do this click the foot notes button in the schematic view and a dialog box will show up. Match each device to its corresponding foot note and then save the schematic. Now it is finally time to import the wiring schematic into the PCB view within KiCad. Do not worry that all the elements are jumbled and out of order simply hit ctrl ‘M’ and move the pieces into their proper orientation or the orientation of choice. Finally inspect each layer of the PCB and ensure all connections are where they are supposed to be and save the PCB to the transmitter project file. The routed PCB in the PCB window of KiCad can be clearly seen in figure 13.

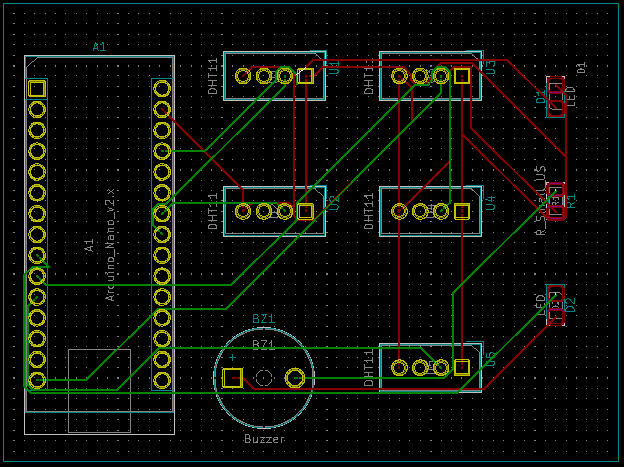


Figure 13: PBC design for transmitter

* + 1. ***The Implementation of Subtask 1.6***

The implementation of subtask 1.6 is the design of a printed circuit board for the receiver module of this project. To implement this task create a new project folder in KiCad and save the project name as Receiver. First open the schematic view, place the cursor over the place button, and select add a symbol from the drop down menu. After the window opens type in Arduino Nano in the search bar and press enter. Select the Arduino Nano and insert it in the schematic window. This same process will be followed for inserting the LED, RF receiver, and the Antenna modules. Once all symbols have been inserted into the schematic place them in a logical manner and begin the wiring process. To do this select the place wire button on the right hand too bar and connect the modules to their respective outputs and inputs on the Arduino Nano. Once this step is completed it is time to add footprints for each of the schematic pieces so they can be transferred to the PCB view. To do this select the assign PCB footnotes button on the tool bar and wait for the dialog box to appear. Then link each corresponding part to its proper footnote and it is time to start the implementation. Select the generate netlist box on the upper tool bar and this will push all assigned footnotes to the PCB view. Then select the PCB view and properly orient the newly inserted parts into view. The second to last step is to select the routing tool from the right hand tool bar and route each of the wires to their respective ports. Finally once this is accomplished select then add a graphic tool on the right hand tool bar and draw a proper boarder around the newly created PCB. The completed PCB design for the receiver can be seen in figure 14.

A screenshot of a computer

Description automatically generated

Figure 14: PCB design for receiver.

* + 1. ***Implementation of Subtask 1.7***

The implementation of subtask 1.7 is the design of a program and configuration of the Arduino and Motion Detector. To do this one must first acquire all components necessary for this portion of the project. Those components are an Ilagoo motion detection unit, Arduino Nano, jumper wires, and a Piezo buzzer. No source files will need to be downloaded for this particular portion of the project. As exemplified in the figure 15 directly below the motion detector has three pins the far right pin is the data pin, the far left pin is the ground pin, and the middle pin is the 5V pin. On the Piezo buzzer the pin that corresponds with the ‘+’ mark on the device is the 5V pin and the adjacent pin is the ground pin. Connect the data pin of the motion detection module to digital pin two on the Arduino Nano, the 5V pin to 5V output, and the ground pin to ground respectively. Repeat the same process with the piezo buzzer with the exception that the 5V input pin from the buzzer will be attached to digital output pin nine on the Arduino Nano. Once this is completed simply copy the source code into Arduino IDE and upload it to the board for complete operation. To see the output on the terminal screen when the motion detector is tripped, select the tools option, and then serial monitor. One should wave their hand in front of the detector to see that the circuit and devices are working properly.

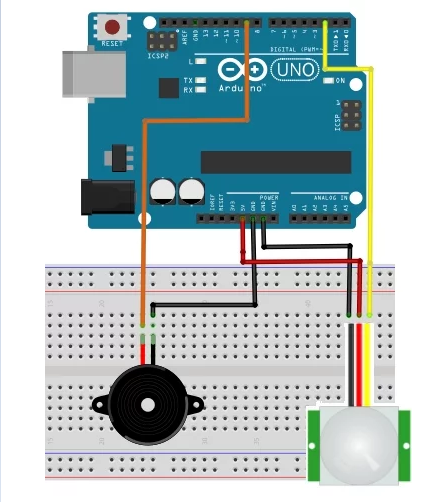


Figure 15: Motion Detector Breadboard Schematic.

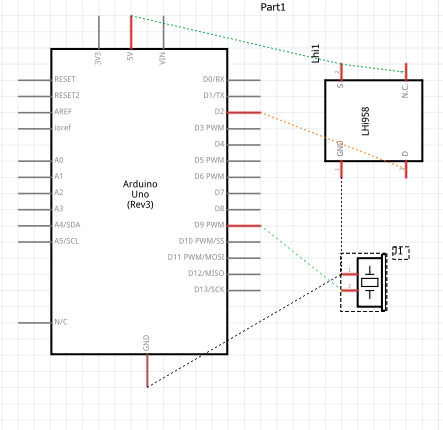


Figure 16: Motion Detector Circuit Schematic.

## Implementation of Task 2 Wireless Sensor Network Design.

* + 1. ***Implementation of Subtask 2.1***

The implementation of subtask 2.1 is the configuration of the circuit for the Arduino and RF transmitter. This is a relatively simple process as there are only two components that are needed for this portion of the project. Those two components are a 433 MHz transmitter and an Arduino UNO. As exemplified in Figure 6 below the 433 MHz transmitter has three pins. The far left pin on the transmitter is the data input pin, the middle pin is the ground pin, and the pin on the far right of the transmitter is the 5V input pin. The receiver input data pin is connected to the digital output pin of the Arduino board and the 5V input and ground pins are connected to their corresponding pins on the Arduino. This can be clearly visualized by the circuit diagram in figure 17 directly below.

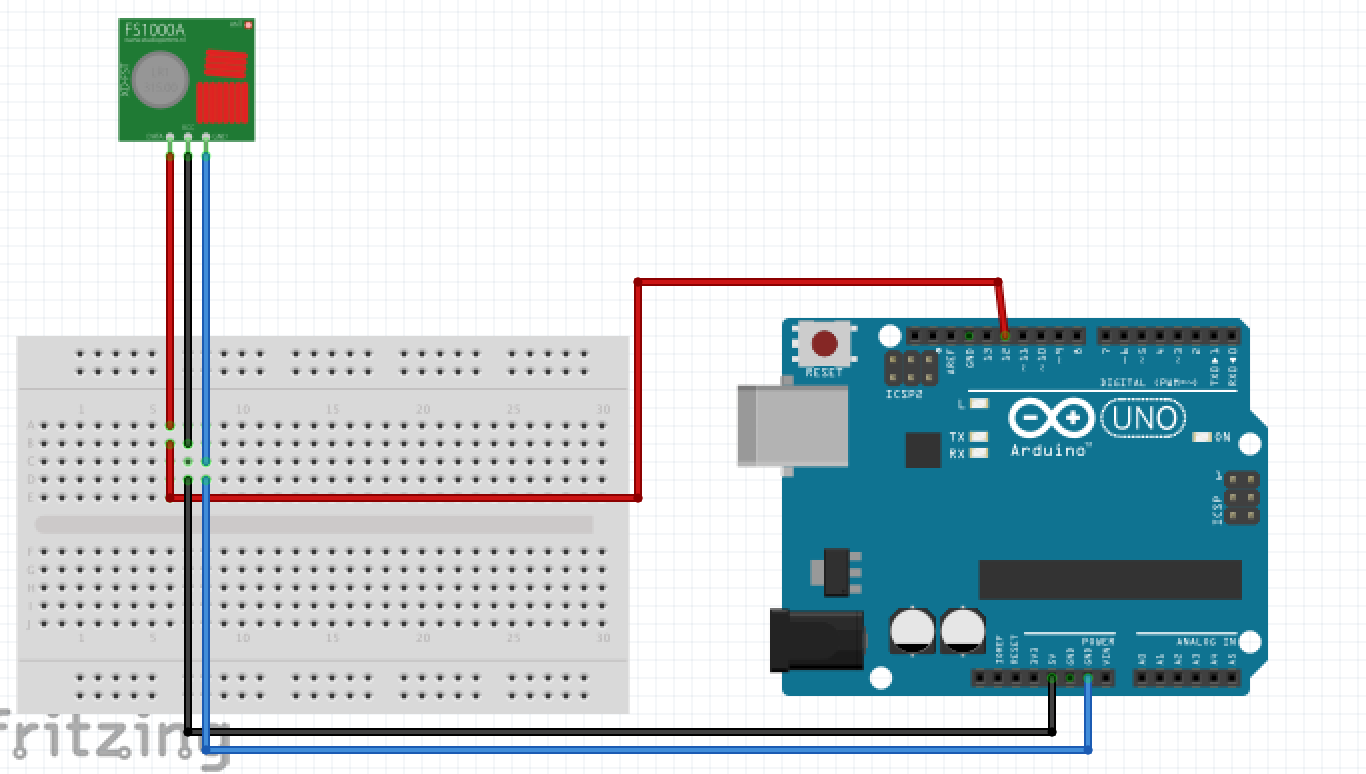


Figure 17: Breadboard Schematic for Geekcreit RF Transmitter.

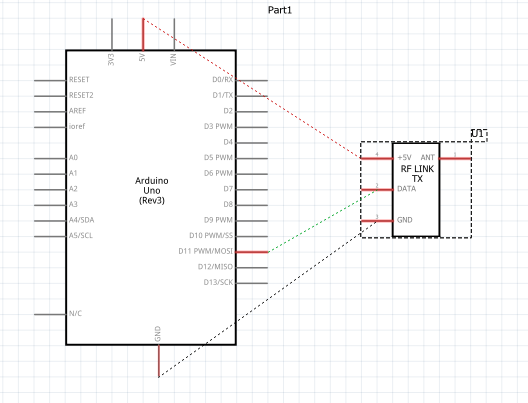


Figure 18: Circuit Diagram for Geekcreit RF Transmitter.

* + 1. ***Implementation of Subtask 2.2***

The implementation of subtask 2.2 is the design of a program for the Arduino and RF transmitter. For this part of the project the Virtual Wire and Radio Head zip file libraries need to be downloaded and extracted into the project folder. These two libraries will be crucial for instantiating the transmission and acquisition pins on the Arduino board. Once these libraries have been successfully extracted into the project folder one can start writing the actual code for the transmitter. Be sure to include the *<VirtualWire.h>* declaration at the beginning of the program and set your reference pin to digital pin 13. Once this is done set the TX pin on the Arduino for pin 12 and the bandpass rate for data transmission to two thousand bps. Once this is done simply build the code that is needed for proper data transmission and upload it to the Arduino board.

* + 1. ***Implementation of Subtask 2.3***

The implementation of subtask 2.3 is the configuration of the 433 MHz RF receiver and Arduino circuit. This task is again relatively short as there are only two components needed for the succession of this task. Those two components again being the Arduino UNO and the 433 MHz RF receiver. As exemplified by the circuit in Figure 7 below the 433 MHz receiver has four pins on its board. The far left pin is the 5V power input pin, the two pins to the right of the power pin are the digital and analog ground pins, and the final pin on the far right is the data input pin. The data pin is connected to digital pin 11 on the Arduino board, the ground pin is connected to digital ground, and the power pin is connected to the 5V output pin respectively. This can be clearly visualized in figure 19 directly below.

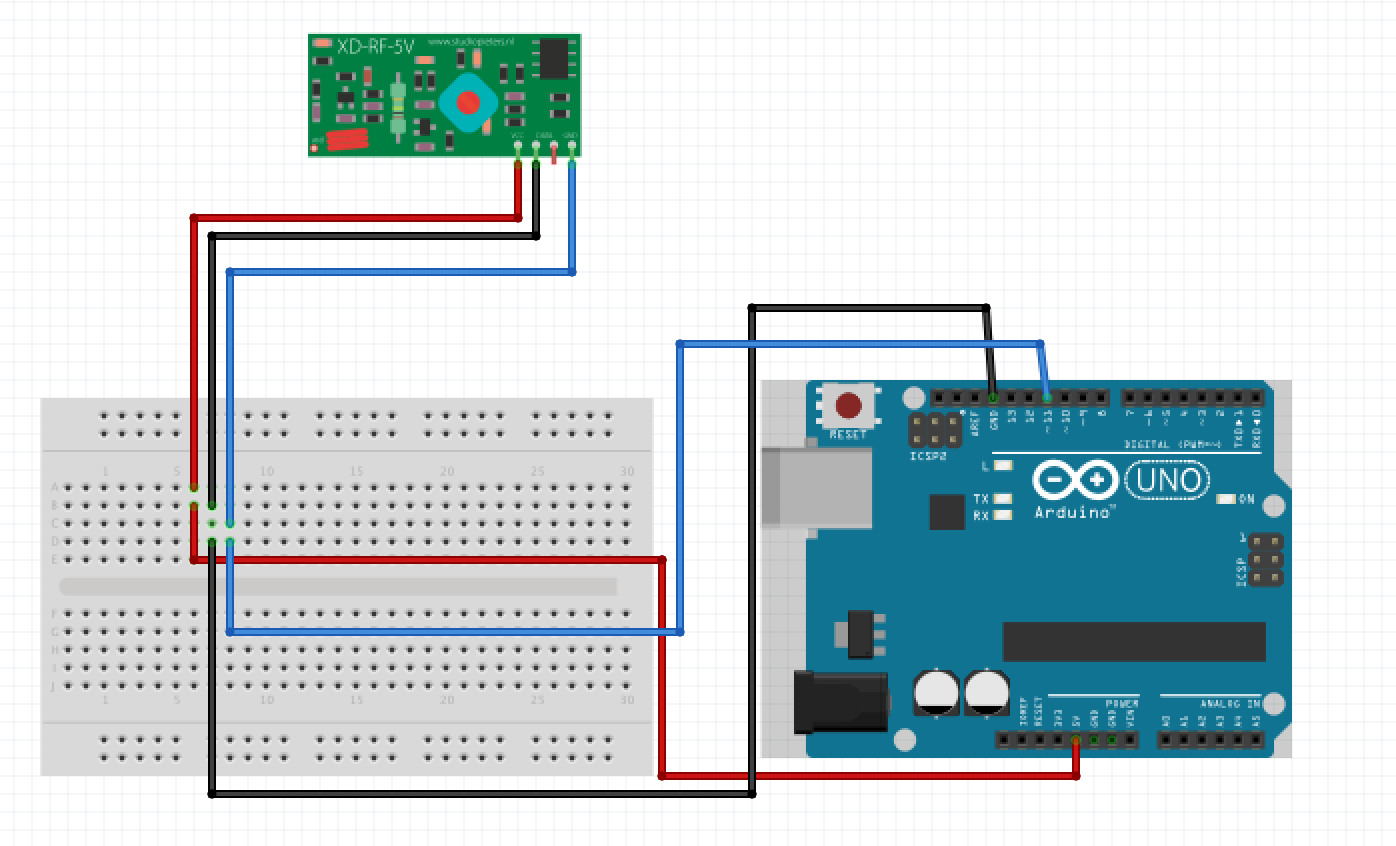


Figure 19: Breadboard Schematic for Geekcreit RF receiver.

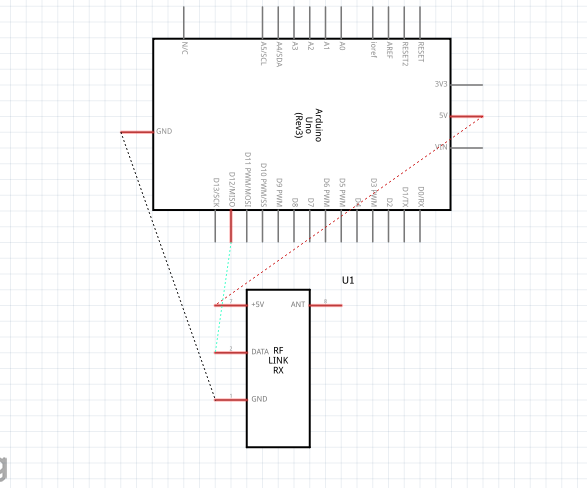


Figure 20: Circuit Diagram for Geekcreit RF Receiver.

* + 1. ***Implementation of Subtask 2.4:***

The implementation of subtask 2.4 is the design of a program for the 433 MHz RF receiver and Arduino. Once again for this part of the project the Virtual Wire and Radio Head zip file libraries need to be downloaded and extracted into the project folder. These two libraries will be crucial for instantiating the transmission and receiving pins on the Arduino board. For this program again the *<VirtualWire.h>* library must be included and the reference pin will remain the constant as pin 13. The data input pin from the transmitter to the receiver will be set at 11 and the bps acquisition rate for the data should be set to begin at 9600 bps. Once these constants have been set the program can be built. Set the RX pin to data pin 11 in the program code and be sure to start the RX program. Once this is done move on to the void loop of the program code and insert the buffer length and an *if-else* statement to see if the data has been transmitted and acquired. Finally once this is done upload the program code to the Arduino board and test the complete system to see if data is being transmitted. To do this go to the tools drop down menu on in Arduino IDE and select the serial monitor tab. The communication port should open and you should see the message displayed in Figure 21.

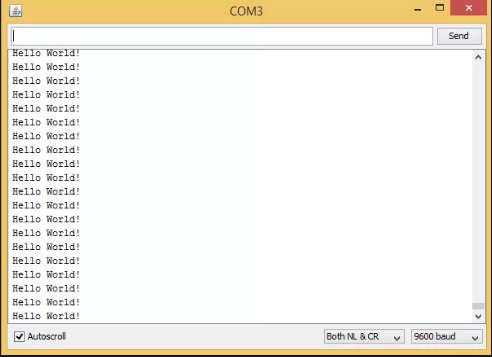


Figure 21: Geekcreit 433 MHz RF transmitter output.

* + 1. ***Implementation of Subtask 2.5***

The implementation of subtask 2.5 is the configuration of all sensors with the RF receiver and transmitter. To accomplish this task two separate breadboards and Arduinos will be utilized. The bread board with all of the interfaced sensors will be utilized for the transmitter. The only thing that needs to be added to the circuit is the RF transmitter itself. It will need to be grounded and will require a 5V power input from the Arduino board. The data pin will be connected to the digital input pin 11 on the Arduino board. The circuit is exemplified by Figure 22 and all components and connections can clearly be visualized. For the second board nothing need be changed from its original circuit layout in subtask 4.2.4. The circuit diagram for the receiver is exemplified by Figure 24 and all connections are cleared represented.

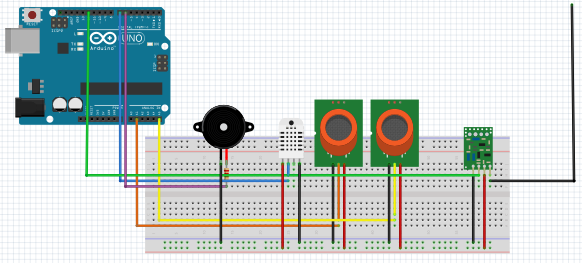


Figure 22: Complete RF transmitter breadboard schematic.

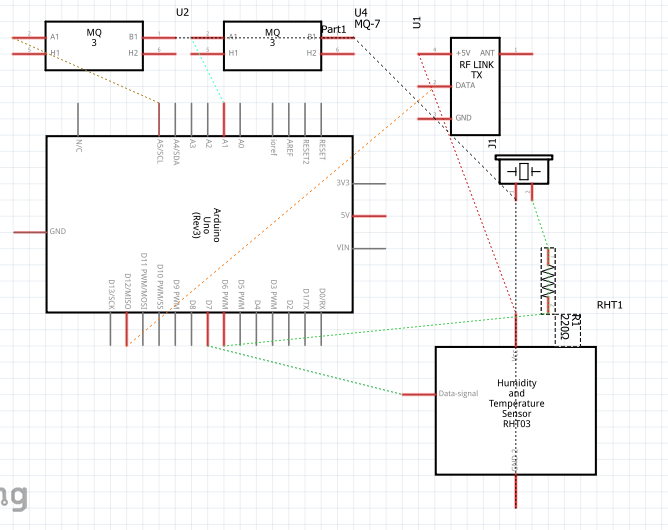


Figure 23: Complete RF transmitter circuit schematic.

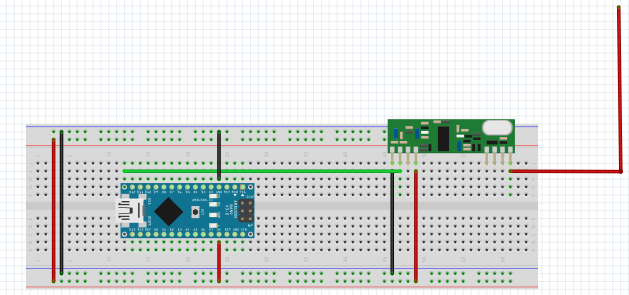


Figure 24: Complete RF receiver breadboard schematic.

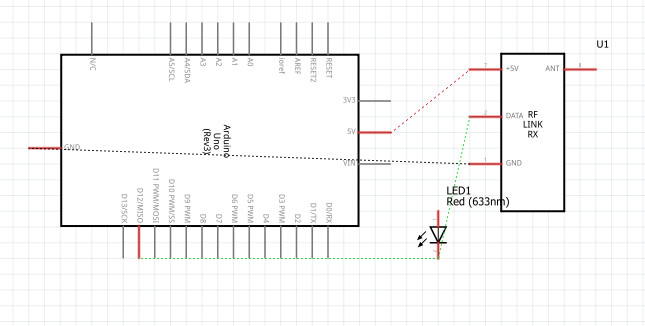


Figure 25: Complete RF receiver circuit schematic.

* + 1. ***Implementation of Subtask 2.6***

The implementation of subtask 2.6 is the configuration of the raspberry pi and Arduino. There are two ways of implementing this particular part of the project. If one choses to use an IoT platform then a logic level converter is needed for the communication of the raspberry pi and Arduino. The logic level converter steps the 5V output voltage of the Arduino down to 3.3V so the raspberry pi can process the data being transferred over the TX and RX pins of each device. When wiring the logic level converter the TX pin from the Arduino will go to the TX input on the converter and the RX pin will go to the RX input respectively. However, for the raspberry pi the RX output from the converter will go to the TX pin of the pi and the TX output from the converter will go to the RX pin of the pi. The 5V output voltage from the Arduino will be wired to the input voltage V1 on the converter and the 3.3V output voltage from the pi will be wired into V2 respectively. Each ground will be wired to their corresponding grounds on either side of the logic level converter. For the second option no logic level converter is needed and the Arduino is connected directly to one of the four USB ports located on the back of the pi itself. No other wiring is needed for that portion. However, if one chooses this option there is back end work that needs to be completed to ensure that the pi and microcontroller are configured properly. First one needs to SSH into the pi and make sure the device is up to date, and I2C and serial port commination are enabled. Once this is done download the Arduino library from the Raspberian website and install it on the device. Once these steps have been completed plug in the Arduino to the raspberry pi and type the following commands */dev/ttyAM0\** a grid of data will appear and find the *ttyAM (‘your ID number’)* corresponding to your particular Arduino (each Arduino ID is unique). Once this has been found the pi and Arduino will have been successfully configured.

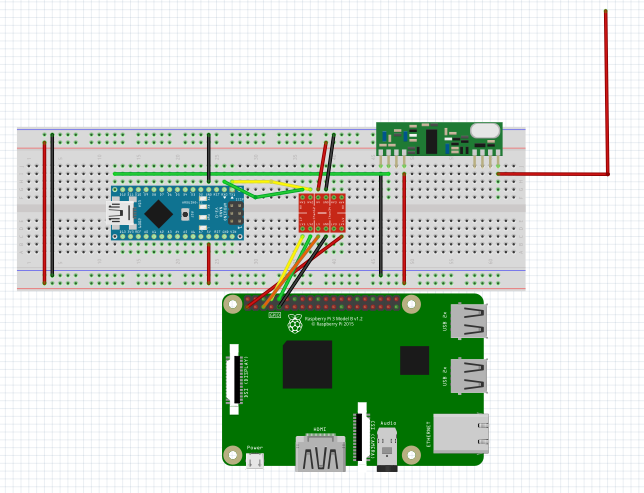


Figure 26: Configuration of Raspberry pi and Arduino with logic level converter.

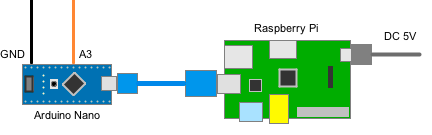


Figure 27: Configuration of Raspberry pi and Arduino using direct USB connection.

* + 1. ***Implementation of Subtask 2.7***

The implantation of Subtask 2.7 is to configure raspberry pi to retrieve, display, and store node data. For this portion of the project the Arduino and Raspberry Pi must be configured properly in subtask 2.6 or data retrieval will be impossible. To make this possible there are a multitude of different approaches to take. The approach that this project takes is by directly reading in the transmitted data from the Arduino to the Raspberry Pi via the USB connection and then printing that data to the Raspberry Pi’s serial monitor. To do this one will need to create a python file called *transmitter.py* inside of the *usr/bin/home* directory of the raspberry pi. Once this has been completed one can start designing the program that will interoperate and store the data. To accomplish this first use the import command to import serial which will instantiate the USB connection between the Raspberry Pi and the Arduino. After this is accomplished use the *serial.read(‘your device info’)* command to read in the data. Then use the command to output that data to a text file so it can be stored for further operations later in the project. The output should look similar to the figure below.

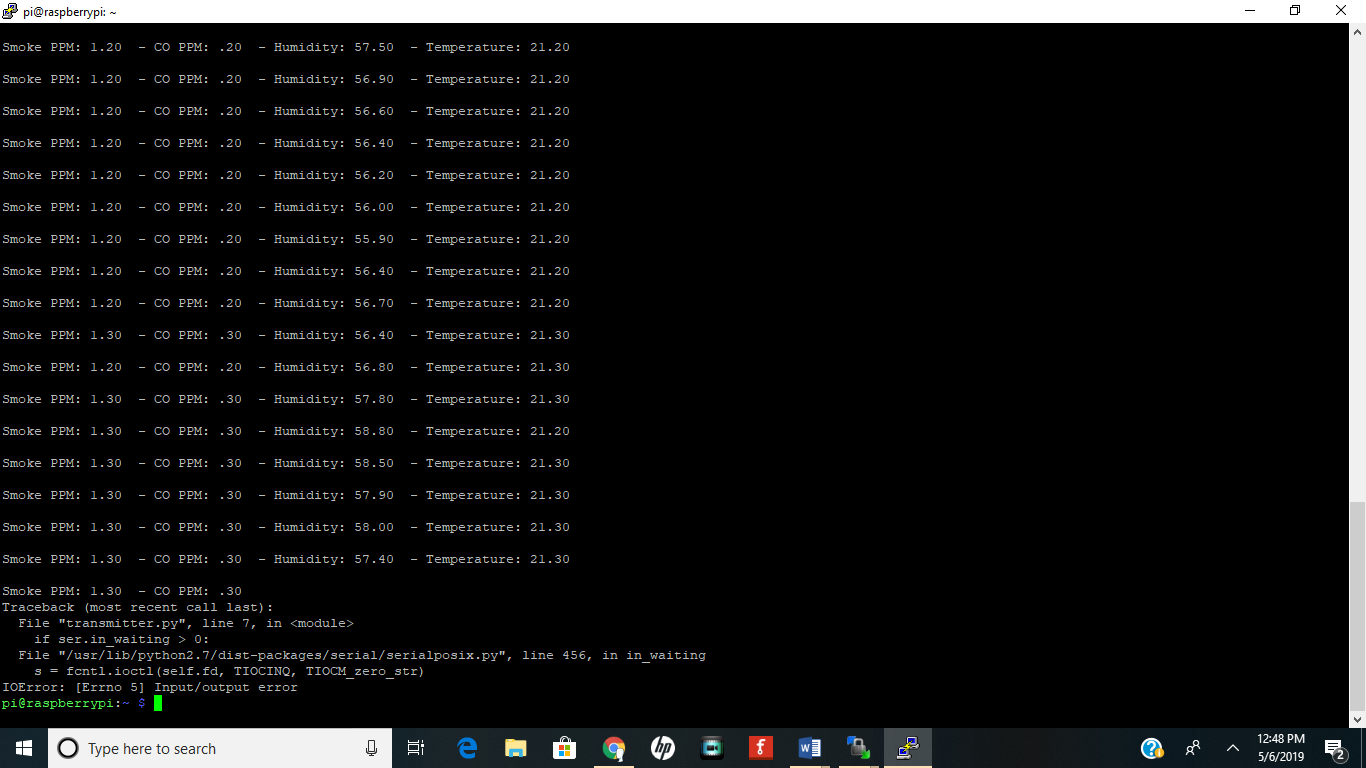


Figure 28: Terminal output from raspberry pi.

* + 1. ***Implementation of Subtask 2.8***

Subtask 2.8 is the design of a program that will interoperate node data and transfer it to a webpage. The first step in this portion of the project is to install a program called apache on the raspberry pi. To do this simply type *sudo apt-get install acpache* and hit enter in the putty terminal window. Apache is a web service that affords the user an ability to push data to an actual web page hosted on the Raspberry Pi’s IP address. Once the apache installation has finished create two new files. It is essential to place these files *cgi-bin* folder otherwise apache will not be able to access them. Call the first file *console.cgi* and the second *serial.html*. The cgi file will be used as the driving file where the python code will be written, stored, and saved. The html file will be the host for the web service to display node data and contiguously update the data stream every five seconds. Once the node is plugged in and python code is ran the data should appear on the webpage hosted on the Raspberry Pi. To view the webpage use google chrome or fire fox and simply type in the IP address of your pi followed by */cgi-bin/console.cgi*. The code for both programs can be found in the index of this report. The output should look similar to the data displayed in figure 29 below.

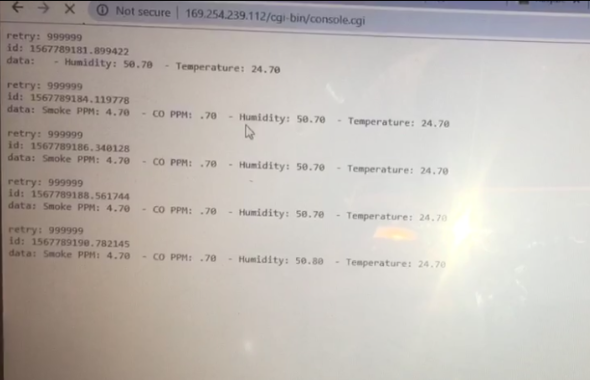


Figure 29: Website output from Raspberry Pi

* + 1. ***Implementation of subtask 2.9***

Subtask 2.9 is the configuration of the website and the raspberry pi to receive, interoperate, and store node data. The implementation of this step has changed many times over the course of the project. The best way to achieve this step is to store the node data in a text file detailed in subtask 2.7. The webpage itself has to be hosted on the pi and can’t be directly connected to the main website of this project. To circumvent this issue place a redirect link in the data section of the main web page. Then use the echo command in the code prompt to echo the dynamic IP address of the raspberry pi this will pull up the webpage that displays the node data that is updated every five seconds.

* + 1. ***Implementation of subtask 2.10***

Subtask 2.10 is the design of a voice notification platform for building occupants. For this segment of the project four main components are needed. The first is a working node, the second is the Raspberry pi, the third is an aux cord, and the last is a speaker of any form. The aux cord will be connected to the speaker and then to the aux port of the Raspberry pi. Once this is completed turn on the raspberry pi and SSH into the Pi. Now it is time to give the Pi its very own voice. First use sudo apt-get install espeak. This will download a Linux platform that will allow the raspberry pi to translate text files into automated speech. Once this is completed create a text file called speakAll.txt. This will be the file that will contain all of the virtual alert messages for the device platform. Once that is completed create a python file called *speakAll.py* by using the command sudo nano *speakAll.py*. To access espeak one will have to instantiate a call for the platform within the python file. This is needed because espeak is a Linux platform and thus can’t be directly imported for use in python 3. Once this step is completed the file can be automatically called in the cgi file to transmit a specific alert message for each node in the system. The python code is depicted in the figure below.

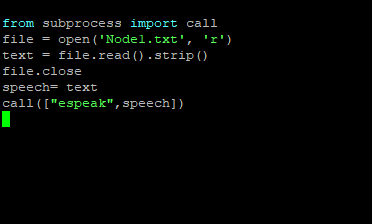


Figure 30: Python code that calls espeak

## Implementation of Task 3 Website Design

* + 1. ***Implementation of subtask 3.1***

Subtask 3.1 is the design of the background and layout of the website. The purpose of the website is so the owner or manager has access to the layout of the building, readings from the sensors and to have a place to store employee information. Along with the layout, the building diagram will have a grid programmed over it. This grid will allow whatever room, hallway or other area to flash red when the detector for that corresponding area is going off. The employee information stores the name and phone number of the employees. Once the numbers are stored, the raspberry pi phone will be able to access these numbers and alert the employees via SMS messaging.

* + 1. ***Implementation of subtask 3.2***

While submitting the login and registration information locally, the information from the form was stored into a local database. Since the webpage became global, there was no way to access the local database. The information submitted in the form could not be saved into the database it was originally coded to access. Heroku has its own database so that anything submitted could be stored and accessed by this database. Heroku’s ClearDB MySQL allows a new database to be made similar to the database used originally. ClearDB comes with its own host, database name, username and password. To change what database the login and registration pages are associated with, the old database information was replaced with Heroku’s database information. The tables and elements of the tables were created with MySQL Workbench. The table and elements have the exact same name as the table and elements in the local database to prevent any extra changes in the code. Below is the Heroku database and the content of the database.

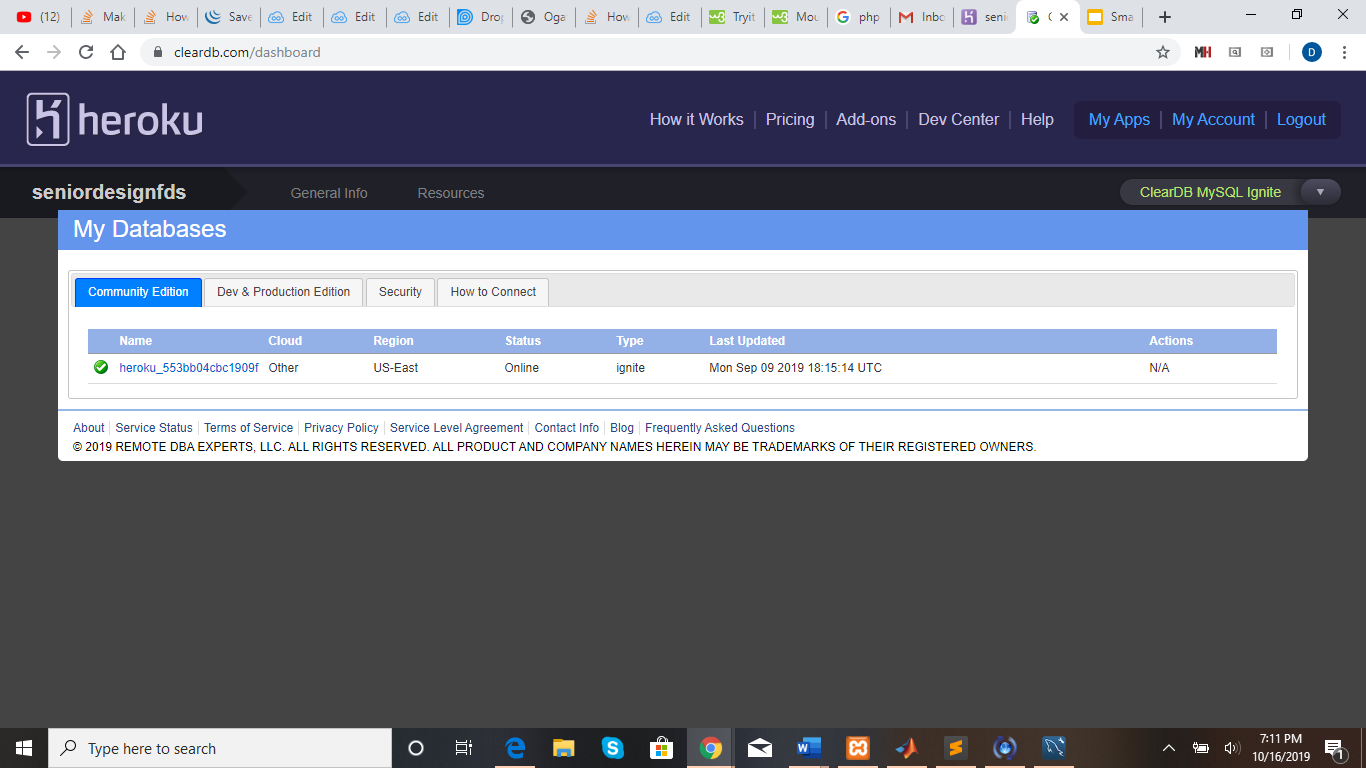


Figure 31: Heroku Main Terminal

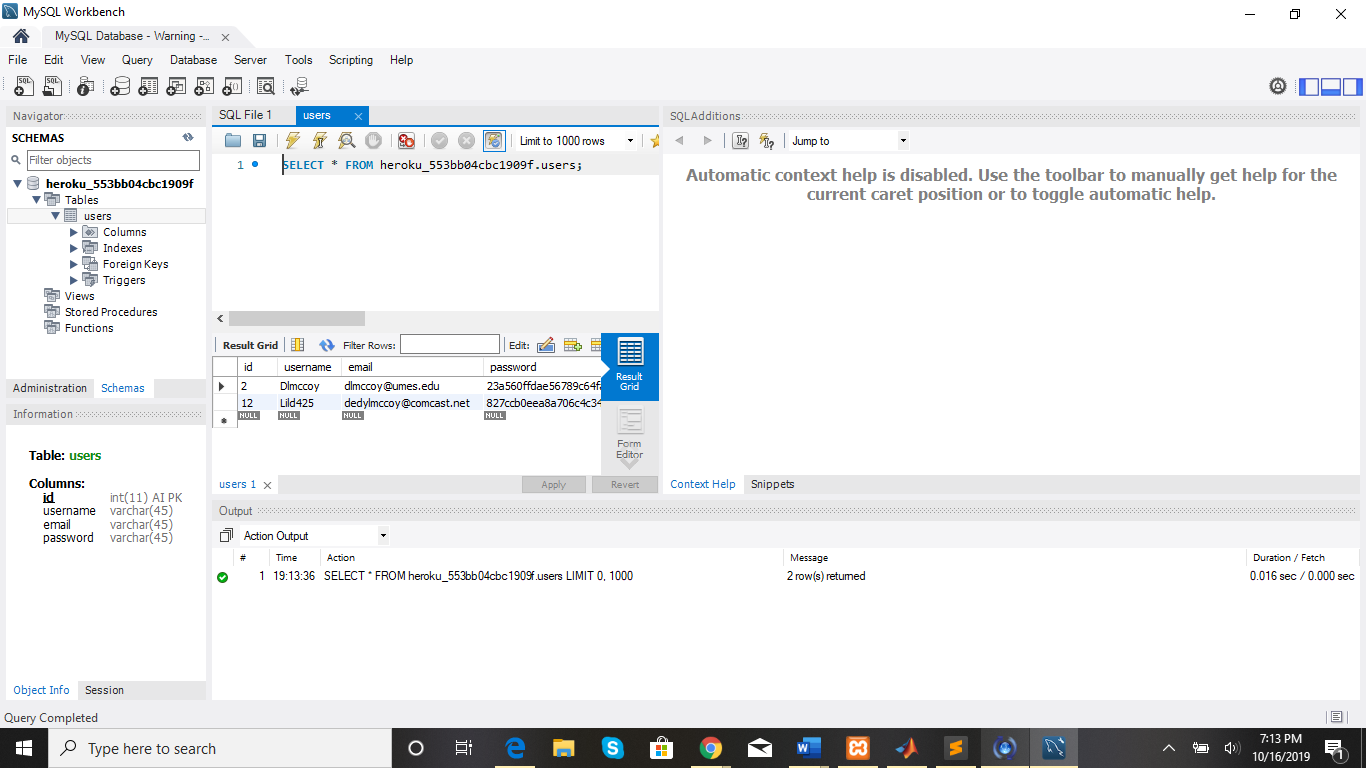


Figure 32: ClearDB MySQL Database

* + 1. ***Implementation of subtask 3.3***

The layout of the floor plans needed a grid programmed over it so that each area of the building can be accessed by corresponding nodes. To do so, some configurations were made to the CSS for the building layout page. “Repeating-linear-gradient” was used to create grid lines. These grid lines are set so that they overlay the image, remain the same size as the image and move with the image if necessary. Shown below are the layouts of each floor on the webpage with the gridlines overlaid on them.

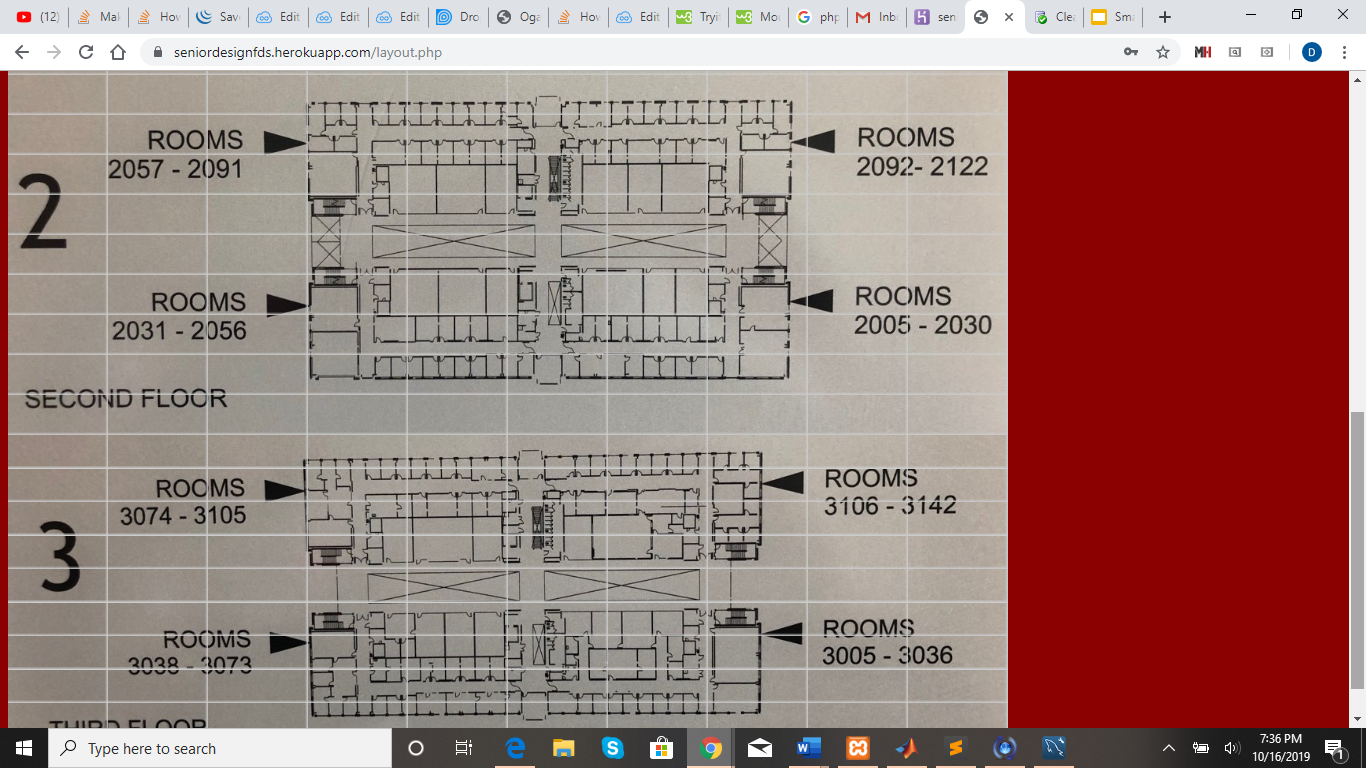


Figure 33: Building Layout with Grid Overlay

* + 1. ***Implementation of subtask 3.4***

The implementation of subtask 3.4 is to design a program that configures the raspberry pi and website. There is no direct way to link the two separate webpages into one contiguous unit. However, a website hosted by the apache server on the Raspberry Pi can be instantiated within another main webpage on the web. That is exactly what will be used to link the two websites together. To do this type the following command under the readings tab of the MySQL file *echo ‘<a href = \\”http://www.xxx.xxx.xxx.xxx/cgi-bin/console.cgi”>Click here</a>’*. This will directly link the webpage hosted on the Raspberry Pi that displays the node data to the main Fire Protection Website. The *xxx.xxx.xxx.xxx* represents the IP address of the Raspberry Pi being used for this project. Simply insert the IP of one’s own Raspberry Pi to ensure this step works properly. The Click here will be the link that will redirect the user to the website hosted on the Raspberry Pi.

## Implementation of Task 4 Configure Raspberry Pi and Website

* + 1. ***Implementation of subtask 4.1***

The implementation of subtask 4.1 is the design a program that will transmit and display WSN data on webpage. This particular implementation is an extension of subtask 4.2.8. The main aspect that will be modified on this particular part of the project is the console.html file. To access this file first SSH into the raspberry pi, open a session in putty, and type the command *cd /usr/lib/cgi-bin* and hit enter. Once in the cgi-bin directory open the console.html file by using sudo nano console.html. This will open the html file that will be modified for this portion of the project. Once in the file window modify the code so that the back ground of the webpage is red. Then write the code to place the word Node above the first number of the output from the pi. After this is done repeat the same process starting first with Humidity, Temperature, Smoke Particulate, and finally Carbon Monoxide. Then save the file by typing control x, then enter y in the terminal window, press enter and save the file. The code for this portion of the project can be found in the index of this file under modified console.html.

* + 1. ***Implementation of subtask 4.2***

The implementation of subtask 4.2 is the design of a program that will display warning messages from pi on webpage. To implement this portion of the project first SSH into the Raspberry Pi and start a session in putty. Once done with this open the transmitter.py file in the */home/pi* directory by using the command *sudo nano transmitter.py*. Once the file is open in the command window modify the code by putting print statements in each of the *“if”* and *“else if”* blocks of the transmitter file. Each block will have its own unique warning message corresponding to the type of input from the receiver. The first three *“if”* and *“else if”* blocks correspond to node one. Likewise, the next three blocks correspond to the output from node two. Finally, the last three blocks will correspond to node three and so on. The first block of each will correspond to smoke particulate, the second will correspond to carbon monoxide, and the last will correspond to temperature. To create each unique warning message type in *print(“Warning! There is a possible (Type of Issue) in room (Room Number)”)*. Do this for each of the warning messages for all node ID’s. The code for this section can be found in the index under the modified *transmitter.py*.

* + 1. ***Implementation of subtask 4.3***

The implementation of subtask 4.3 is the configuration of node ID data from the raspberry Pi and the website. The implementation of this task is one of the more simple tasks for this project. Each node is programed with its own unique node ID and that information is passed directly to the receiver that the Raspberry Pi reads the data from. The only thing that needs to be modified for this section of the project is again the transmitter.py file on the Raspberry Pi. To do this SSH into the Raspberry Pi and open a session in Putty. Then open the transmitter.py file in the */home/pi* folder. To do this simply type in sudo nano transmitter.py in the terminal window of the putty session. Once the file is open add the node designation to each one of the *“if”* and *“else if”* blocks of the python code. Type *if(A[0] == 1 and A[2] > 400)*, this is an example of the code that should be created in each block. The designation *A[0]* corresponds to the section of the array that the unique node ID is stored in. A special note for this section all output from the raspberry pi will be string values. Thus, ensure that the node ID is bracketed by parentheses in the code or it will not work properly.

* + 1. ***Implementation of subtask 4.4***

The implementation of subtask 4.4 is to program an alarm system shut off for administrative use. This part of the project is actually an inadvertent built in feature. To do ensure that this piece works correctly SSH into the Raspberry Pi and open a session in Putty. Once this has been completed run the *transmitter.py* file located in the */home/pi* folder. To run the file type in python *transmitter.py* and hit the enter key. Then let the system run for approximately five minutes to stabilize all sensors on the network. Then simply click control C on the keyboard of the computer or on the keyboard the pi is plugged into and this will terminate the file. Since the *transmitter.py* file is the main driver file that operates all of the system alarm info this will inadvertently give the administrator the ability to terminate the system if there is a false alarm.

## Implementation of Task 5 Design Automated Emergency Phone Call

* + 1. ***Implementation of subtask 5.1***

The implementation of subtask 5.1 is the set up and installation of Twilio on the raspberry pi. The first step in this process is SSH into the Raspberry-pi and then open up a new session in Putty. Twilio is installed using the pip command so ensure that the Raspberry-pi being used has the pip library installed. To do this type the command *sudo apt-get install pip* into the command line of the Putty session. If pip is already installed this process will state zero new items installed 0 objects changed. If it isn’t pip will be installed through its dependency tree and prompt will state installation complete on the command line of the Putty terminal. Now that pip has been installed it is time to install Twilio. To do this type *sudo pip install Twilio* in the command line of the Putty terminal and press the enter key, this will prompt the installation of Twilio on the Raspberry-pi. Twilio requires the users to make an account so that user can be assigned a unique ID and a phone number for their automated service. To complete the remaining configuration of Twilio simply visit their website, register for a new account, and enter ones information. Be sure to take a picture of the unique ID that is associated with the new account and the phone number that is assigned to it. An example of this process is detailed in the figure below.



Figure 34: Installation and Configuration of Twilio on the Raspberry Pi

* + 1. ***Implementation of subtask 5.2***

The implementation of subtask 5.2 is the configuration of Twilio with the WSN network. Now like espeak that was used in an earlier subtask in this project Twilio is a Linux based system, not a python based system. Therefore when using Twilio in python the Linux bases system has to be imported into the code the exact same way that espeak did. To do this use the command from *Twilio.rest import Client*. Then the rest is quite simple just type the number of the user or users you wish to send the notification alert too in the *“to=”* part of the program code, the from is the personal number assigned from Twilio, finally the body is where one will put the type of threat that triggered the alarm. An example of this code can be seen in figure 35 below.

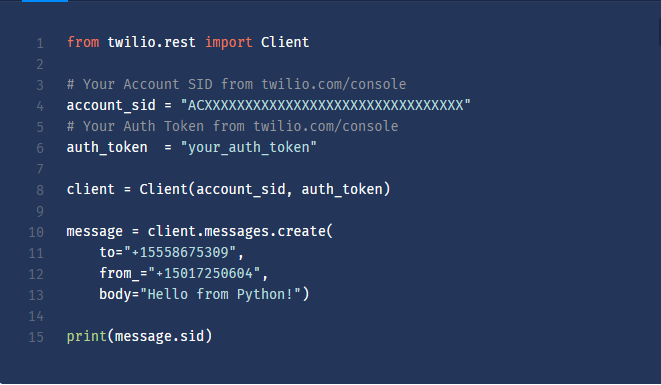


Figure 35: Example Twilio Code using Python call.

* + 1. ***Implementation of subtask 5.3***

Subtask 5.3 is the implementation of an autonomous phone call that corresponds to each unique node ID. The code for this task can be seen in figure thirty six below. To implement this code first SSH onto the Raspberry-Pi and open a new session in Putty. Twilio requires the user to import pip3 so do this using the command import and then type pip3 afterwards. To ensure that this particular segment of the code will run one will have to put in their account ID and the authentication token to ensure Twilio can be instantiated within the python code. The code is written using if statements, if the statement occurring is true then the code above will run, send a message, and call all building occupants and 911 dispatch. Obviously for sanities sake this project can’t actually call 911 with a false emergency as that would be illegal. In figure thirty six below, one can see the call ID printed. Both Twilio and python have their benefits and pitfalls. Deciding which platform would better fit the technology used in this project while providing the user with a unique experience is difficult. Twilio requires one to upgrade one’s account to enjoy all the platforms amenities. This makes implementing this particular platform a bit more difficult.

A screenshot of a cell phone

Description automatically generated

Figure 36: Full implementation of Twilio Code

## Implementation of Task 6 Design SMS Message Data Base

* + 1. ***Implementation of subtask 6.1***

Subtask 6.1 is the implementation and design of a local database for building occupant phone number storage. To initiate this task search for a server called X-Ming on google and download it onto the computer that will be connected to the raspberry pi. This will allow the raspberry pi to establish a terminal connection to the computer to display the local phone book for administrative use. To first ensure that X-Ming will be able to establish a secure connection with the Raspberry-pi open a session in Putty and check the SSH options on the left hand side. Click the plus mark on this option and scroll down to port 22 and click on it. Once in the port 22 options box, click X-Ming enabled and type the host name as local host 0:0. This step will ensure a secure connection between the Raspberry-pi and PC. Once this step is completed SSH into the raspberry pi and open a session in Putty. Create a python file in the */home/pi* folder using the command sudo nano phonebook.py. This file will be where the code for this section of the project will be saved under the *home/pi* bin of the raspberry pi. Once the code has been completed saved the file by pressing control X then yes to save the modified buffer. Finally in the command terminal of the SSH session type *python phonebook.py* and the output should look exactly like the figure displayed below.

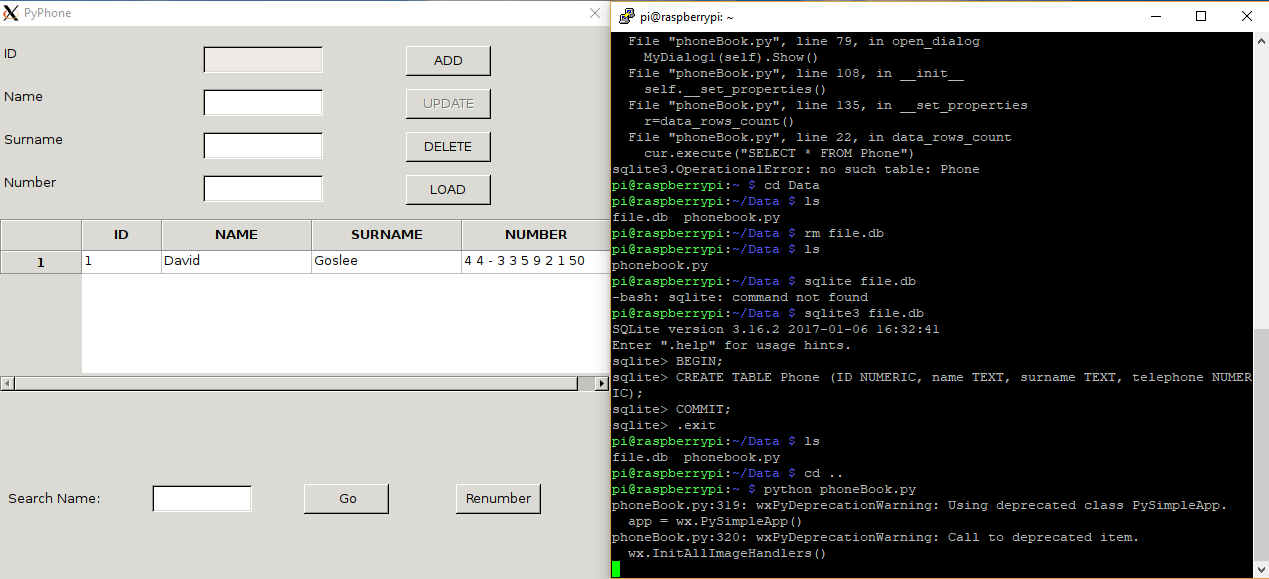


Figure 37: Local Phone book on Raspberry Pi

1. **Implementation of Task 7 Design 3-D printed housings for the transmitter and receiver**

***4.7.1. Implementation of subtask 7.1-7.3***

Subtask 7.1 is the design of the 3-D printed housing for the transmitter and receiver. To accomplish this task, you’ll need the solid works software. Using solid works, you can, sketch, Extrude, and fillet your design to be 3-D printed. Using Solid Works smart dimensions, you can precisely identify the size of your design. This design was created to 165mm x 80mm to ensure there enough room to hold the PCB and its content. The thickness set to 0.10in and a height of 1.5in at the base for the bottom and .5in for the top. This particular design has 2 holes on the top lid, to allow some air flow and to be able to view the LEDs and 4 corner screw holes to fasten down the PCB. Repeat the steps with the receiver using different measurement, to attain a snug design holding our device stable. The model shown in figure 39 shows the base of the receiver this particular model has not point of entry, completely encapsulating the system, for later models other designs were made, the final design fit the product we had in mind.



Figure 38: 3-D transmitter Print out

A close up of a screen

Description automatically generated

*Figure 39: 3- receiver Print out*

1. **Conclusion (Discussion and Future Plans)**

**Acknowledgment**

# Appendix

## Component Specs

### Specs of Arduino UNO

* 1. Microcontroller: ATmega328
  2. Operating Voltage: 5V
  3. Input Voltage (recommended): 7-12V
  4. Input Voltage (limits): 6-20V
  5. Digital I/O Pins: 14 (of which 6 provide PWM output)
  6. Analog Input Pins: 6
  7. DC Current per I/O Pin: 40 mA
  8. DC Current for 3.3V Pin: 50 mA
  9. Flash Memory: 32 KB of which 0.5 KB used by bootloader
  10. SRAM: 2 KB (ATmega328)
  11. EEPROM: 1 KB (ATmega328)
  12. Clock Speed: 16 MHz

### Specs of Arduino NANO

* 1. Microcontroller ATmega328
  2. Operating Voltage (logic level): 5 V
  3. Input Voltage (Recommended): 7-12 V
  4. Input Voltage (limits): 6-20 V
  5. Digital I/O Pins : 14 (of which 6 provide PWM Output)
  6. Analog Input Pins: 8
  7. DC Current per I/O Pin: 40 mA
  8. Flash Memory 32 KB (ATmega328) of which 2 KB used by bootloader
  9. SRAM: 2 KB (ATmega328)
  10. EEPROM: 1 KB (ATmega328)
  11. Clock Speed: 16 MHz
  12. Measurements: 0.73" x 1.70"

### Specs of Raspberry Pi 3

* 1. SoC: Broadcom BCM2837.
  2. CPU: 4× ARM Cortex-A53, 1.2GHz.
  3. GPU: Broadcom VideoCore IV.
  4. RAM: 1GB LPDDR2 (900 MHz)
  5. Networking: 10/100 Ethernet, 2.4GHz 802.11n wireless.
  6. Bluetooth: Bluetooth 4.1 Classic, Bluetooth Low Energy.
  7. Storage: microSD.
  8. GPIO: 40-pin header, populated.

1. ***Specs of MQ-2 Smoke Sensor***
   1. Operating Voltage is +5V
   2. Can be used to Measure or detect LPG, Alcohol, Propane, Hydrogen, CO and even methane
   3. Analog output voltage: 0V to 5V
   4. Digital Output Voltage: 0V or 5V (TTL Logic)
   5. Preheat duration 20 seconds
   6. Can be used as a Digital or analog sensor
   7. The Sensitivity of Digital pin can be varied using the potentiometer

### Specs of MQ-7 Carbon Monoxide Sensor

* 1. Sensitive for carbon monoxide
  2. Output voltage boosts along with the concentration of the measured gases increases
  3. Fast response and recovery
  4. Adjustable sensitivity
  5. Signal output indicator
  6. Power: 2.5V ~ 5.0V
  7. Dimension: 40.0mm \* 21.0mm
  8. Mounting holes size: 2.0mm
  9. VCC ↔ 2.5V ~ 5.0V
  10. GND ↔ power supply ground
  11. AOUT ↔ MCU.IO (analog output)
  12. DOUT ↔ MCU.IO (digital output)

### Specs of DHT22 Temperature and Humidity Sensor

* 1. Low cost
  2. 3 to 5V power and I/O
  3. 2.5mA max current use during conversion (while requesting data)
  4. Good for 0-100% humidity readings with 2-5% accuracy
  5. Good for -40 to 80°C temperature readings ±0.5°C accuracy
  6. No more than 0.5 Hz sampling rate (once every 2 seconds)
  7. Body size 27mm x 59mm x 13.5mm (1.05" x 2.32" x 0.53")
  8. 4 pins, 0.1" spacing
  9. Weight (just the DHT22): 2.4g

### Specs of 433 MHz RF Receiver

* 1. Wireless (RF) Simplex Transmitter and Receiver
  2. Receiver Operating Voltage: 3V to 12V
  3. Receiver Operating current: 5.5mA
  4. Operating frequency: 433 MHz
  5. Transmission Distance: 3 meters (without antenna) to 100 meters (maximum)
  6. Modulating Technique: ASK (Amplitude shift keying)
  7. Data Transmission speed: 10Kbps
  8. Circuit type: Saw resonator
  9. Low cost and small package

### Specs of 433 MHz RF transmitter

* 1. Wireless (RF) Simplex Transmitter and Receiver
  2. Transmitter Operating Voltage: +5V only
  3. Transmitter Operating current: 9mA to 40mA
  4. Operating frequency: 433 MHz
  5. Transmission Distance: 3 meters (without antenna) to 100 meters (maximum)
  6. Modulating Technique: ASK (Amplitude shift keying)
  7. Data Transmission speed: 10Kbps
  8. Circuit type: Saw resonator
  9. Low cost and small package

### Specs of 4-Channel Logic Level Converter

* 1. Logic level shifter Bi-Directional with 4 channels between high logic and low logic
  2. Small module with 1.3cm X 1.5cm
  3. With 2 rows of 6 pin

### Specs of Piezo Buzzer

* 1. Wide operating voltage: 3~250 V
  2. Lower current consumption: less than 30 mA higher rated frequency
  3. Larger footprint
  4. Higher sound pressure level

## Source Code.

### Source Code of DHT22 Temperature and Humidity Sensor

// DHT Temperature & Humidity Sensor

// Unified Sensor Library Example

// Written by Tony DiCola for Adafruit Industries

// Released under an MIT license.

// REQUIRES the following Arduino libraries:

// - DHT Sensor Library: https://github.com/adafruit/DHT-sensor-library

// - Adafruit Unified Sensor Lib: https://github.com/adafruit/Adafruit\_Sensor

#include <Adafruit\_Sensor.h>

#include <DHT.h>

#include <DHT\_U.h>

#define DHTPIN 2 // Digital pin connected to the DHT sensor

// Feather HUZZAH ESP8266 note: use pins 3, 4, 5, 12, 13 or 14 --

// Pin 15 can work but DHT must be disconnected during program upload.

// Uncomment the type of sensor in use:

//#define DHTTYPE DHT11 // DHT 11

#define DHTTYPE DHT22 // DHT 22 (AM2302)

//#define DHTTYPE DHT21 // DHT 21 (AM2301)

// See guide for details on sensor wiring and usage:

// https://learn.adafruit.com/dht/overview

DHT\_Unified dht(DHTPIN, DHTTYPE);

uint32\_t delayMS;

void setup() {

Serial.begin(9600);

// Initialize device.

dht.begin();

Serial.println(F("DHTxx Unified Sensor Example"));

// Print temperature sensor details.

sensor\_t sensor;

dht.temperature().getSensor(&sensor);

Serial.println(F("------------------------------------"));

Serial.println(F("Temperature Sensor"));

Serial.print (F("Sensor Type: ")); Serial.println(sensor.name);

Serial.print (F("Driver Ver: ")); Serial.println(sensor.version);

Serial.print (F("Unique ID: ")); Serial.println(sensor.sensor\_id);

Serial.print (F("Max Value: ")); Serial.print(sensor.max\_value); Serial.println(F("°C"));

Serial.print (F("Min Value: ")); Serial.print(sensor.min\_value); Serial.println(F("°C"));

Serial.print (F("Resolution: ")); Serial.print(sensor.resolution); Serial.println(F("°C"));

Serial.println(F("------------------------------------"));

// Print humidity sensor details.

dht.humidity().getSensor(&sensor);

Serial.println(F("Humidity Sensor"));

Serial.print (F("Sensor Type: ")); Serial.println(sensor.name);

Serial.print (F("Driver Ver: ")); Serial.println(sensor.version);

Serial.print (F("Unique ID: ")); Serial.println(sensor.sensor\_id);

Serial.print (F("Max Value: ")); Serial.print(sensor.max\_value); Serial.println(F("%"));

Serial.print (F("Min Value: ")); Serial.print(sensor.min\_value); Serial.println(F("%"));

Serial.print (F("Resolution: ")); Serial.print(sensor.resolution); Serial.println(F("%"));

Serial.println(F("------------------------------------"));

// Set delay between sensor readings based on sensor details.

delayMS = sensor.min\_delay / 1000;

}

void loop() {

// Delay between measurements.

delay(delayMS);

// Get temperature event and print its value.

sensors\_event\_t event;

dht.temperature().getEvent(&event);

if (isnan(event.temperature)) {

Serial.println(F("Error reading temperature!"));

}

else {

Serial.print(F("Temperature: "));

Serial.print(event.temperature);

Serial.println(F("°C"));

}

// Get humidity event and print its value.

dht.humidity().getEvent(&event);

if (isnan(event.relative\_humidity)) {

Serial.println(F("Error reading humidity!"));

}

else {

Serial.print(F("Humidity: "));

Serial.print(event.relative\_humidity);

Serial.println(F("%"));

}

}

* + - 1. ***Source Code of MQ-2 Smoke Sensor***

int redLed = 12; // red LED pin

int greenLed = 11; // green LED pin

int buzzer = 10; // Piezo Buzzer pin

int smokeA0 = A5; // MQ-2 analog output pin

// Your threshold value

int sensorThres = 18;

void setup() {

pinMode(redLed, OUTPUT);

pinMode(greenLed, OUTPUT);

pinMode(buzzer, OUTPUT);

pinMode(smokeA0, INPUT);

Serial.begin(9600);

}

void loop() {

int analogSensor = analogRead(smokeA0);

Serial.print("Pin A0: ");

Serial.println(analogSensor);

// Checks if it has reached the threshold value

if (analogSensor > sensorThres)

{

digitalWrite(redLed, HIGH); // Will light red LED if threshold value is reached.

digitalWrite(greenLed, LOW); // Will not light green LED if threshold value is reached.

tone(buzzer, 1000, 200); // Will activate buzzer if threshold value is reached.

}

else

{

digitalWrite(redLed, LOW); // Will not light red LED.

digitalWrite(greenLed, HIGH); // Will light green LED.

noTone(buzzer); // Buzzer will not be activated.

}

delay(100);

}

* + - 1. ***Source Code of MQ-7 Smoke Sensor***

#define analogMQ7 A0 // Signal

#define ledPin D7 // Device internal LED

int MQ7sensorValue = 0; // value read from the sensor

void setup()

{

Serial.begin(9600);

pinMode(analogMQ7, INPUT);

pinMode(ledPin, OUTPUT);

}

void loop() {

analogWrite(analogMQ7, HIGH); // HIGH = 255

// heat for 1 min

delay(60000);

// now reducing the heating power: turn the heater to approx 1,4V

analogWrite(analogMQ7, 71.4); // 255x1400/5000

// heat for 90 sec

delay(90000);

analogWrite(analogMQ7, HIGH);

delay(50); // Get an analog reading takes approx 100uSec

MQ7sensorValue = analogRead(analogMQ7);

Serial.print("MQ-7 PPM: ");

Serial.println(MQ7sensorValue);

if (MQ7sensorValue <= 200)

{

Serial.println("Air-Quality: CO perfect");

}

else if ((MQ7sensorValue > 200) || (MQ7sensorValue <= 800)) // || = or

{

Serial.println("Air-Quality: CO normal");

}

else if ((MQ7sensorValue > 800) || (MQ7sensorValue <= 1800))

{

Serial.println("Air-Quality: CO high");

}

else if (MQ7sensorValue > 1800)

{

digitalWrite(ledPin, HIGH); // optical information in case of emergency

Serial.println("Air-Quality: ALARM CO very high");

delay(3000);

digitalWrite(ledPin, LOW);

}

else

{

Serial.println("MQ-7 - cant read any value - check the sensor!");

}

}

* + - 1. ***Source Code of 433 MHz Transmitter.***

//TRANSMITTER

// RadioHead - Version: Latest

#include <RH\_ASK.h>

// SPI - Version: Latest

// Not actualy used but needed to compile

#include <SPI.h>

RH\_ASK driver;

void setup()

{

Serial.begin(9600);

if (!driver.init())

Serial.println("init failed");

}

void loop()

{

const char \*msg = "Hello World!";

driver.send((uint8\_t \*)msg, strlen(msg));

driver.waitPacketSent();

delay(1000);

}

* + - 1. ***Source Code of 433 MHz Receiver.***

//RECEIVER

// RadioHead - Version: Latest

#include <RH\_ASK.h>

// SPI - Version: Latest

// Not actualy used but needed to compile

#include <SPI.h>

RH\_ASK driver;

void setup()

{

Serial.begin(9600);

if (!driver.init())

Serial.println("init failed");

}

void loop()

{

uint8\_t buf[12];

uint8\_t buflen = sizeof(buf);

if (driver.recv(buf, &buflen))

{

int i;

Serial.print("Message: ");

Serial.println((char\*)buf);

}

}

* + - 1. ***Source Code For Motion Detector.***

int speakerOut = 9;//Piezo buzzer's positive terminal is connected to digital pin 9

byte names[] = {'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C'};

int tones[] = {1915, 1700, 1519, 1432, 1275, 1136, 1014, 956};

byte melody[] = "2d2a1f2c2d2a2d2c2f2d2a2c2d2a1f2c2d2a2a2g2p8p8p8p";

// count length: 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0

// 10 20 30

int count = 0;

int count2 = 0;

int count3 = 0;

int MAX\_COUNT = 24;

int statePin = LOW;

void siren();

volatile byte intruder;

void setup()

{

Serial.begin(115200);

attachInterrupt(0, intruder\_detect, RISING);//Initialize the intterrupt pin for the motion sensor (Arduino digital pin 2)

intruder = 0;

}

void loop()

{

}

void intruder\_detect()//This function is called whenever an intruder is detected by the arduino

{

intruder++;

Serial.println("Intruder detected");

for(int i=0; i<3; i++)//Play the alarm three times

siren();

}

void siren()//This function will make the alarm sound using the piezo buzzer

{

for (count = 0; count < MAX\_COUNT; count++) {

for (count3 = 0; count3 <= (melody[count\*2] - 48) \* 30; count3++) {

for (count2=0;count2<8;count2++) {

if (names[count2] == melody[count\*2 + 1]) {

analogWrite(speakerOut,1023);

delayMicroseconds(tones[count2]);

analogWrite(speakerOut, 0);

delayMicroseconds(tones[count2]);

}

if (melody[count\*2 + 1] == 'p') {

// make a pause of a certain size

analogWrite(speakerOut, 0);

delayMicroseconds(100);

}

}

}

}

}

1. ***Project Code.***
2. ***Code of 433 MHz Transmitter and DHT22***

#include <VirtualWire.h>

#include "DHT.h"

#define DHTPIN 4

#define DHTTYPE DHT22

const int led\_pin = 13;

const int transmit\_pin = 12;

struct package

{

float temperature ;

float humidity ;

};

typedef struct package Package;

Package data;

DHT dht(DHTPIN, DHTTYPE);

void setup()

{

// Initialise the IO and ISR

vw\_set\_tx\_pin(transmit\_pin);

vw\_set\_ptt\_inverted(true); // Required for DR3100

vw\_setup(500); // Bits per sec

pinMode(led\_pin, OUTPUT);

}

void loop()

{

digitalWrite(led\_pin, HIGH); // Flash a light to show transmitting

readSensor();

vw\_send((uint8\_t \*)&data, sizeof(data));

vw\_wait\_tx(); // Wait until the whole message is gone

digitalWrite(led\_pin, LOW);

delay(2000);

}

void readSensor()

{

dht.begin();

delay(1000);

data.humidity = dht.readHumidity();

data.temperature = dht.readTemperature();

}

1. ***Code of 433 MHz Receiver and DHT22***

#include <VirtualWire.h>

#include <Adafruit\_ST7735.h>

#include <Adafruit\_GFX.h>

#define TFT\_CS 10

#define TFT\_RST 8

#define TFT\_DC 9

Adafruit\_ST7735 tft = Adafruit\_ST7735(TFT\_CS, TFT\_DC, TFT\_RST);

// Option 2: use any pins but a little slower!

#define TFT\_SCLK 13 // set these to be whatever pins you like!

#define TFT\_MOSI 11 // set these to be whatever pins you like!

const int receive\_pin = 12;

char temperatureChar[10];

char humidityChar[10];

struct package

{

float temperature = 0.0;

float humidity = 0.0;

};

typedef struct package Package;

Package data;

void setup()

{

tft.initR(INITR\_BLACKTAB);

tft.fillScreen(ST7735\_BLACK);

printUI();

delay(1000);

// Initialise the IO and ISR

vw\_set\_rx\_pin(receive\_pin);

vw\_setup(500); // Bits per sec

vw\_rx\_start(); // Start the receiver PLL running

}

void loop()

{

uint8\_t buf[sizeof(data)];

uint8\_t buflen = sizeof(data);

if (vw\_have\_message()) // Is there a packet for us?

{

vw\_get\_message(buf, &buflen);

memcpy(&data,&buf,buflen);

Serial.print("\nPackage:");

Serial.print(data.temperature);

//String temperatureString = String(data.temperature,1);

//temperatureString.toCharArray(temperatureChar,10);

//tft.fillRect(10,20,80,30,ST7735\_BLACK);

//printText(temperatureChar, ST7735\_WHITE,10,20,3);

//String humidityString = String(data.humidity,1);

//humidityString.toCharArray(humidityChar,10);

//tft.fillRect(10,95,80,100,ST7735\_BLACK);

//printText(humidityChar, ST7735\_WHITE,10,95,3);

Serial.print("\n");

Serial.println(data.humidity);

}

}

void printText(char \*text, uint16\_t color, int x, int y,int textSize)

{

tft.setCursor(x, y);

tft.setTextColor(color);

tft.setTextSize(textSize);

tft.setTextWrap(true);

tft.print(text);

}

void printUI()

{

printText("TEMPERATURE", ST7735\_GREEN,30,5,1); // Temperature Static Text

printText("o", ST7735\_WHITE,90,13,2);

printText("C", ST7735\_WHITE,105,20,3);

printText("HUMIDITY", ST7735\_BLUE,30,80,1); // Temperature Static Text

printText("%", ST7735\_WHITE,90,95,3);

}

1. ***Code of MQ-7 and MQ-2 sensors.***

#define analogMQ7 A0 // Signal

#define ledPin D7 // Device internal LED

int MQ7sensorValue = 0; // value read from the sensor

void setup()

{

Serial.begin(9600);

pinMode(analogMQ7, INPUT);

pinMode(ledPin, OUTPUT);

}

void loop() {

analogWrite(analogMQ7, HIGH); // HIGH = 255

// heat for 1 min

delay(60000);

// now reducing the heating power: turn the heater to approx 1,4V

analogWrite(analogMQ7, 71.4); // 255x1400/5000

// heat for 90 sec

delay(90000);

analogWrite(analogMQ7, HIGH);

delay(50); // Get an analog reading takes approx 100uSec

MQ7sensorValue = analogRead(analogMQ7);

Serial.print("MQ-7 PPM: ");

Serial.println(MQ7sensorValue);

if (MQ7sensorValue <= 200)

{

Serial.println("Air-Quality: CO perfect");

}

else if ((MQ7sensorValue > 200) || (MQ7sensorValue <= 800)) // || = or

{

Serial.println("Air-Quality: CO normal");

}

else if ((MQ7sensorValue > 800) || (MQ7sensorValue <= 1800))

{

Serial.println("Air-Quality: CO high");

}

else if (MQ7sensorValue > 1800)

{

digitalWrite(ledPin, HIGH); // optical information in case of emergency

Serial.println("Air-Quality: ALARM CO very high");

delay(3000);

digitalWrite(ledPin, LOW);

}

else

{

Serial.println("MQ-7 - cant read any value - check the sensor!");

}

}

1. ***Code of 433 MHz Transmitter and All Sensors.***

// Include RadioHead Amplitude Shift Keying Library

#include <RH\_ASK.h>

// Include dependant SPI Library

#include <SPI.h>

//#include <VirtualWire.h>

// Include DHT Libraries from Adafruit

// Dependant upon Adafruit\_Sensors Library

#include "DHT.h";

// Define Constants

#define DHTPIN 7 // DHT-22 Output Pin connection

#define DHTTYPE DHT22 // DHT Type is DHT 22 (AM2302)

// Define Variables

float hum; // Stores humidity value in percent

float temp; // Stores temperature value in Celcius

float smoke; // Stores smoke value PPM

// Define output strings

String str\_humid;

String str\_temp;

String str\_out;

String str\_smoke;

int redLed = 9;

int greenLed = 11;

int buzzer = 10;

float smokeA0 = A5;

// Your threshold value

int sensorThres = 200;

// Create Amplitude Shift Keying Object

RH\_ASK rf\_driver;

// Initialize DHT sensor for normal 16mhz Arduino

DHT dht(DHTPIN, DHTTYPE);

void setup() {

// Initialize ASK Object

rf\_driver.init();

// Start DHT Sensor

dht.begin();

pinMode(redLed, OUTPUT);

pinMode(greenLed, OUTPUT);

pinMode(buzzer, OUTPUT);

pinMode(smokeA0, INPUT);

}

void loop()

{

delay(2000); // Delay so DHT-22 sensor can stabalize

smoke = analogRead(smokeA0);

hum = dht.readHumidity(); // Get Humidity value

temp= dht.readTemperature(); // Get Temperature value

// Convert Humidity to string

str\_humid = String(hum);

// Convert Temperature to string

str\_temp = String(temp);

// Convert Smoke PPM to String

str\_smoke = String(smoke) ;

// Combine Humidity and Temperature

str\_out = str\_humid + "," + str\_temp + "," + str\_smoke;

// Compose output character

static char \*msg = str\_out.c\_str();

rf\_driver.send((uint8\_t \*)msg, strlen(msg));

rf\_driver.waitPacketSent();

if (smoke > sensorThres)

{

digitalWrite(redLed, HIGH);

digitalWrite(greenLed, LOW);

tone(buzzer, 1000, 200);

}

else

{

digitalWrite(redLed, LOW);

digitalWrite(greenLed, HIGH);

noTone(buzzer);

}

}

1. ***Code of 433 MHz Receiver and All Sensors.***

#include <RH\_ASK.h>

// Include dependant SPI Library

#include <SPI.h>

String str\_humid;

String str\_temp;

String str\_analog;

String str\_out;

// Create Amplitude Shift Keying Object

RH\_ASK rf\_driver;

void setup()

{

// Initialize ASK Object

rf\_driver.init();

// Setup Serial Monitor

Serial.begin(9600);

}

void loop()

{

// Set buffer to size of expected message

uint8\_t buf[11];

uint8\_t buflen = sizeof(buf);

// Check if received packet is correct size

if (rf\_driver.recv(buf, &buflen))

{

// Message received with valid checksum

// Get values from string

// Convert received data into string

str\_out = String((char\*)buf);

// Split string into two values

for (int i = 0; i < str\_out.length(); i++) {

if (str\_out.substring(i, i+1) == ",") {

str\_humid = str\_out.substring(0, i);

str\_temp = str\_out.substring(i+1);

break;

}

}

// Print values to Serial Monitor

Serial.print("Humidity: ");

Serial.print(str\_humid);

Serial.print(" - Temperature: ");

Serial.println(str\_temp);

Serial.print(" - Smoke(PPM: ");

Serial.print(str\_analog);

}

}

}

**REFERENCES**

[1] “Arduino 433MHz RF Tutorial with DHT22.” *educ8s.Tv - Watch Learn Build*, 21 Feb. 2018, educ8s.tv/arduino-433mhz-rf-tutorial-dht22/.

[2] “How to Set Up the DHT11 Humidity Sensor on an Arduino.” *Circuit Basics*, 22 June 2018, [www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-an-arduino/](http://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-an-arduino/).

[3] Instructables, Ardre Sentosa. “Raspberry Pi - Arduino Serial Communication.” *Instructables*, Instructables, 3 Oct. 2017, www.instructables.com/id/Raspberry-Pi-Arduino-Serial-Communication/.

[4] Miller, Ben. “Building a Wireless Sensor Network in Your Home.” *Computer Skills Envato Tuts*, 21 Mar. 2014, computers.tutsplus.com/tutorials/building-a-wireless-sensor-network-in-your-home--cms-19745.

[5] “MQ-7 Carbon Monoxide Sensor Circuit Built With Arduino.” *MQ-7 Carbon Monoxide Sensor Circuit Built with an Arduino*, www.learningaboutelectronics.com/Articles/MQ-7-carbon-monoxide-sensor-circuit-with-arduino.php.

[6] Sanjeev, Arvind. “How to Build an Arduino-Powered Motion-Sensor Alarm  | Arduino.” *Maker Pro*, Maker Pro, 8 May 2019, maker.pro/arduino/tutorial/how-to-build-an-arduino-powered-motion-sensor-alarm.

[7] “Smoke Detection Using MQ-2 Gas Sensor.” *Arduino Project Hub*, create.arduino.cc/projecthub/Aritro/smoke-detection-using-mq-2-gas-sensor-79c54a.