EasyChair Preprint

№ 3348

Secure Data Collection Using Autonomous Unmanned Aerial Vehicles

John Bowman, Jordan Brooks, Chandler Lopez and Anaseli Marcos-Martinez

May 7, 2020

James Madison University
Integrated Science & Technology (ISAT)
ISAT 493 Final Deliverable
2019 - 2020 Senior Capstone

Secure Data Collection Using Autonomous Unmanned Aerial Vehicles

---

Author(s):
John Bowman
Jordan Brooks
Chandler Lopez
Anaseli
Marcos-Martinez

Submitted to:
Dr. Ahmad Salman

May 5, 2020

Contents

List of Figures

1 Abstract

In recent years, Unmanned Aerial Vehicles (UAVs) also commonly referred as drones have become a practical solution for data collection in both academic and commercial fields. The use of drones can allow users to collect data without disrupting the environment and keep safe from dangerous areas such as: military operations and surveillance in locations where maneuverability is difficult. The different data collected via UAVs include field work GPS coordinates, environmental quality measurements, photographs/video and much more. The data collected by these UAVs may contain sensitive information, leading to an increased value on the security of that data. Without proper security services, information may be used in unforeseen, or even malicious ways, such as planting false information or leaking particularly sensitive data. In this paper we introduce a secure data collection method from wireless sensor nodes using an autonomous UAV. We built a sensor node with multiple sensors designed to collect data for rainfall, temperature, humidity, ultra violet index, and soil moisture. Along with another sensor node with sensors used to collect data from water, we created a Wireless Sensor Network (WSN) where the sensor data was collected using an UAV. The sensor nodes collected environmental data through several sensors which are securely stored in the sensor node itself. The collected data was then transferred securely to a central node mounted on a DJI F450 Flamewheel quad-copter, which was loaded with a predetermined flight path via Mission Planner to perform the autonomous flight. Aside from previous environmental focus, technical improvements were made such as adding a solar battery to the sensor node and implementing energy efficiency in the nodes' systems while the drone collected data autonomously during a predetermined flight path. This project served as a proof-of-concept that communication with a wireless sensor node has the capability to be deployed for data collection in remote areas efficiently and effectively while maintaining a low power and energy consumption.

2 Introduction

2.1 Environmental Data Gathering

Drones are becoming much more mainstream, while most attention had gone to military usage other industries such as commercial have gained a widespread of popularity with uses ranging from real estate, photography, local/state governments, industrial inspections, agriculture, and insurance companies [1]. Aside from the lack of safety, privacy, and insurance the drone market isn't slowing down. Drones bring many opportunities including wireless communication systems that are cost-effective for wireless connectivity devices such as Wireless Sensor Nodes (WSNs), these wireless communication systems don't require infrastructure coverage [2]. Potter et al. [3] demonstrated UAVs can be used in conjunction with WSNs to enhance data collection methods. WSNs are also gaining usage in many industries and consumer applications such as monitoring, controlling, and storing data to a wireless network. Our focus in this project was to gather multiple data from different sensors in different nodes using an autonomous drone. When collecting data manually from multiple nodes, in a remote area it may be difficult to physically get to the nodes, a disruption in the ecosystem may occur, the area may expose danger to the user, and hiring multiple users is cost and time ineffective in order to gather all the data needed. In this paper, we present a data collection system which uses multiple WSNs to collect environmental data from various sensors autonomously operated UAV. The system is composed of two WSNs one which collects water data and another collects soil data as well as atmospheric data from inside the node itself.

Potter et al. [3] constructed a WSN to monitor the quality of water of Boones Run, a small stream located on the eastern side of the Massanutten Mountain in Rockingham Country, Virginia. A sensor node was submerged in the stream and collected data which was then transferred to a central node attached to an UAV once it reached a certain distance for connectivity. In this project, soil moisture, rainfall, and soil temperature/humidity sensors were used to build our sensor node. The sensor node was programmed to detect the physical and biological properties of soil, soil moisture is vital and means different things to different disciplines [4]. Soil moisture is water stored in the soil and is important to determine the type of biome present and the sustainability of land in agriculture. Due to climate change, the moisture in soil and rainfall are closely being monitored, measured and used to predict droughts in low precipitation areas all around the world. Rainfall is also important in soil quality because it affects the moisture in soil but it also determines how much water different biomes receive. The temperature/humidity in soil is a regulator of plant growth and is used to determine the types of plants to grow during the different seasons in the year [5]. Many environmental government organizations and private companies use soil quality measurements to evaluate the sustainability of land management in agroecosystems.

3 Background Information

3.1 Stakeholders

The FAA (Federal Aviation Administration) establishes rules and regulations that apply differently depending on what the drones are being used for (stakeholders). Each stakeholder has a different set of rules that must be followed, for example, a recreational flier must register their drone with the FAA, make sure they review the rules and be sure they know where they can fly their drone, same goes for the Certificated Remote Pilots. Public Safety Agencies, such as Law Enforcement hold a bigger position in the UAVs (Unmanned Aerial Vehicles) community.

Although the case of our project is designed to collect environmental data from nearby nodes, the capabilities of this drone are very transferable to other areas of interest. One major aspect of our project is the security of data once it has been collected. Secure data transportation is important to any company or individual responsible for the safety of sensitive information. This can be especially troublesome, however, once the physical object containing the sensitive information is not in a controlled environment such as an office or warehouse. Once the drone leaves the immediate area of the users, there are typically no security measures for its contents unless prior arrangements have been made to encrypt the data. If data is left unsecured, it is vulnerable to theft and alterations. For our project, vulnerable data could lead to unknown changes in our environmental data collection. Changes in environmental data could lead to untrue conclusions being drawn about the current state and trends of an area and lead to unfavorable solutions that would cause very harmful impacts to that area. Making sure that the data is secure is a key element of drone data collection.

Finally, the ability to modify these drones relatively affordable for any specific data collection makes this project so diverse in terms of who could use it. Government agencies could implement similar technologies to allow unmanned drones to enter areas that might be dangerous for an individual to enter. For example, a drone could be modified with an infrared camera and told to film a room to dangerous to enter without prior knowledge of its layout and contents. Of course, drones could be used to collect environmental data in areas where humans themselves would pose a threat if they were to enter. Drones could enter areas that are too dangerous for humans, such as wildfires or areas with chemical spills. These type of drones would be incredibly low impact as they would not need to physically interact with anything in a given space to collect data.

3.2 Ethical Considerations

As previously mentioned, privacy is a major concern among stakeholders. Infringing upon people's privacy was a major concern when smartphones became the norm and in doing so, having a reasonable expectation of privacy within a

public place such as a street or other public areas become, somewhat, obsolete. In areas deemed as public locations, there is little to no legal resources that can be taken in the way of ensuring your anonymity to unwanted filming or photography, coupled with the common use of social media to share content, there is little to stop your personal likeness for being shared across the world.

This problem of unwanted "surveillance" is only expanded upon by the use of drones. Now areas, where privacy would normally be expected, are changing, and the need/want limitations on filming are growing. This can be felt especially within the less economically well off population where legal action is simply unattainable given the fees associated with it. Additional ethical dilemmas can be seen in the use of drones within warfare. During the Obama administration, drone warfare became a common term in the fight against Al Qaeda. Describing drone warfare within "Pakistan, Yemen and elsewhere as part of 'a just war – a war waged proportionally, in last resort, and in self-defense" [6]. Although the risk to American soldiers were greatly diminished through the use of drones, civilian casualties were seen as a necessary risk, this, however, lead to a "rising tide of criticism". [7]

Finally, the use of drone warfare has become a form of what some call "risk-less warfare," suggesting a lack of risk, however, this "leads to more states into taking risks to kill more enemies with drone strikes in other lands" and intern "create a world in which more and more states engage in targeted killings because they are easy and because so many of their rivals do it. Such a world would be closer to the rule of the jungle than to the rule-based international order that the US sought to create and sustain over the last 50 years". [8]

3.3 Public Policy

With the new and exponentially growing consumer and professional drone market lawmakers are faced with a never before experienced technology and must regulate it in a fair but not limiting way. Most legal involvement in drone technologies is focused primarily on privacy concerns. Many fear drones for their privacy being infringed upon by drones through the use of surveillance not only from individuals, and business, but also through unfair use by law enforcement. In 2013 the "FAA approved of the use of unmanned aircraft on a case-by-case basis" [9]. This new regulation was primarily focused on the use for governmental agencies being "largely limited to military airspace and the U.S. borders, along with use in special circumstances, such as reconnaissance over forest fires.

In 2014 federal regulation moved to focus on the commercial sector where a handful of companies were given permission to use drones for business purposes. [9] In 2015, due to the current regulation vagueness and lack of everyday users legal options to fly drones the FAA enacted a new policy in which "all owners of small unmanned aircraft, including small unmanned aircraft operated as a model aircraft in accordance with the statutory requirements for model aircraft operations ... may take advantage of the new registration process" [9]. This registration process applies to an unlimited amount of drones per user, however, failure to comply with regulation can lead to a fine of $27,500 and additional

fines up to $250,000 and the possibility of incarceration.

On the Virginia state level, starting in 2015, warrants by government entities are required unless the drone is used in the case of an emergency. This policy, however, does not apply to: "(i) utilization of such systems to support the Commonwealth for purposes other than law enforcement; (ii) certain search and rescue operations; (iii) certain Virginia National Guard and the United States Armed Forces functions; (iv) research and development conducted by institutions of higher education or other research organizations; or (v) the use of unmanned aircraft systems for private, commercial, or recreational use."

3.4 Cultural Dynamics

Drones are capable of, and have been increasingly used for, practical efforts outside the military arena. The private sector has been gradually adopting drone technology for a multitude of applications. For example, the BP gas company uses drones to survey the area they have built their pipelines to ensure safety and efficiency [10]. In their first tests of using drones for surveillance of their operations, BP used a quadcopter, the same type of drone we will be using for our capstone. In 2013, Amazon has announced a program to deliver packages using drones, but this was met with some backlash. People were worried about the idea of drones constantly flying around.

Drones have also been a catalyst for job creation. Engineers, technicians, and pilots are needed to build and operate them. Subsequently, this has resulted in a shift in types of jobs available and the median salary for these new jobs. The issue arises when outdated professionals begin to lose their jobs due to this new technology. For example, delivery truck drivers and cargo planes pilots are expected to begin losing jobs as large companies like Amazon attempt to use drones for delivery. Agencies that monitor and protect wild animals against poachers have started using drones for surveillance, resulting in less poached animals and an easier prosecution case against the poachers [11]. Another example of drones increasing safety is using them for exploring disaster drones. Hurricanes, tornadoes, and places with dangerous radiation and heat can be explored with a drone instead of sending a human into the area, saving time and money.

A more whimsical example of drones being adopted took place in Dubai. They launched an unmanned single carrier drone above the city for the highest bidder to take a ride in. This was the first case of an autonomously controlled vehicle taking a passenger airborne in the Middle East [12]. Dubai, of course, has been known for its luxurious taste and insatiable desire for sleeker and faster. It is no surprise the emirate that holds this mindset has taken it upon themselves to be a key driver in the implementation of drones among civil transportation.

4 Methodology

This section will primarily focus on the our development of the soil quality node depicted in Table 1 and the steps we took to achieve semi-autonomous flight. In our research, we used a custom UAV to add extra hardware in order to perform autonomous flights, the node constructed followed methods from [3] with different types of sensors aimed at collecting soil quality parameters. Moreover, technical improvements were made on energy efficiency as well as data storage. This is where details about the project are explained

4.1 System Design

Our WSN is intended to act indepentetly of human intervention except for: the initial placement of nodes the launch/landing of the drone itself, data transfer from the drone to a tertiary source. Each node contains their own solar powered battery source to allow for an initial charging before implementation a solor source to charge while they are in the field. Each node is designed to collect and write to two separate csv files; one that is transferred to the drone and one that is kept on the node as a backup. These CSV files are time stamped for each individual set of data readings and a new file is created everyday and named with the appropriate date. The files are transferred through a simple socket connection on the drones on board Raspberry Pis own wireless network. drones on board Raspberry Pi is constantly searching for the nodes to connect and will transfer the data automaticly once they are within range. The drone...

4.2 Hardware & Equipment

Here we will discuss the importance of each piece of hardware and their function within the system.

4.2.1 Node Hardware: Purpose and Reasoning

4.2.1.1 Adafruit STEMMA Soil Sensor Traditional low cost soil sensors are "resistive style", meaning two metal prongs are exposed and conductivity is measured between the two. These traditional sensors are prone to oxidation as a result of their exposed prongs creating a need for constant calibration, For this reason we chose to use the Adafruit STEMMA Soil Sensor - I2C Capacitive Moisture Sensor [13]. This soil sensor utilizes capacitive measurements, meaning there is zero exposed metal that would be subject to oxidation. The sensor is able to measure moisture and temperature of the given soil. The soil sensor was physically wired to a Raspberry Pi through the I2C interface.

4.2.1.2 SHT-30 Temperature/Humidity Sensor The Temperature and Humidity sensor were being used to measure and collect the outside air temperature and humidity was the SHT-30 Mesh-protected Weather-proof Temperature/Humidity Sensor [14]. This sensor is weatherproof and comes attached to

Table 1: List of sensors used for the soil quality sensor node

Soil Quality Sensor Node	
Component	**Description**
SI1145-UV Sensor	Detects the amount of UV light
Adafruit STEMMA Soil Sensor	Measures the amount of water in the soil, determines whether the soil is very dry or very wet
FC-37 Rain Sensors	Measure the amount of rainfall in a location
SHT-30 Temperature/Humidity Sensor	Dual sensor, it measures the temperature and humidity
Sunfounder DS18b20 Temperature Sensor	Measures temperature
Solar Charger 26800mAh Hiluckey 18W Power Bank	A solar power bank that stores and discharge power as needed
Pelican 1060 MICRO CASE RAVEN, Black/Clear	Weather resistant case used to store the node when placed in the environment

a 1M cable from the manufacturer meaning we did not have to weatherproof any part of this sensor or wiring as we had to do with other sensors physically located outside the node. The soil sensor was physically wired to the raspberry pi again through the I2C.

4.2.1.3 Node Inner Sunfounder DS18b20 Temperature Sensor The temperature sensor was used inside of the node itself, the DS18b20 sensor [15]. We chose to measure the inside temperature of the node due to overheating concerns. With all the power and computational components being stored within the node, overheating became a concern especially in warmer environments.

4.2.1.4 FC-37 Rain Sensors and the Arduino Uno The sensors we used to measure rainfall are the FC-37 Rain Sensors. They are weather-proof and use conductivity to measure the amount of water that is landing on the sensor. We used two rain sensors to measure if the rainfall was equal around the entire node. We used an Arduino Uno to connect the sensors because the rain sensors use analog sensor values and The Raspberry Pi does not have a way to read analog inputs. Setting up the sensor followed the same method in [16].

4.2.1.5 SI1145-UV Sensors and the Arduino Uno The sensor we used to measure UV was the SI1145-UV Sensor. The sensor was placed inside of the node and its function is to check if the node is getting enough UV rays to charge the solar powered battery. The setup follows the same method as in [17].

4.2.1.6 Power Source The entire Node, consisting of: a Raspberry Pi 0, Arduino Uno, and the previously described sensors is powered by a Solar Charger 26800mAh Hiluckey 18W Power Bank. This specific power bank was chosen for its size, large storage capacity, solar charging capabilities, and high amperage output. The battery has a capacity of 26800 mAh with a 5.5V solar panel input to charge it over time. This charge over time was desired for the hope of the device could be left in the field uninterrupted for as long as possible. The battery can also output up to 3Amps/5V per port which is more than enough to power the entire node as we show in section 6.

4.2.1.7 Environmental Storage The entire node is encapsulated in a 1060 MICRO RAVEN Pelican Case. This case is rated at IP67 water resistance, while also being crush proof and dust proof. With a clear top, the node is still able to have a UV power source while still being completely protected. For the sensors that were needed to be mounted outside of the case in order to measure data, a small hole was drilled through the top where the sensor wires were then fed through. This hole was then completely sealed to protect the inside of the case against rain and other weather conditions. If the exposed wires did not come weatherproof from their manufacturer, they were heat wrapped and then sealed off at the ends.

The finished node with all its components exposed can be seen in Figure 1.

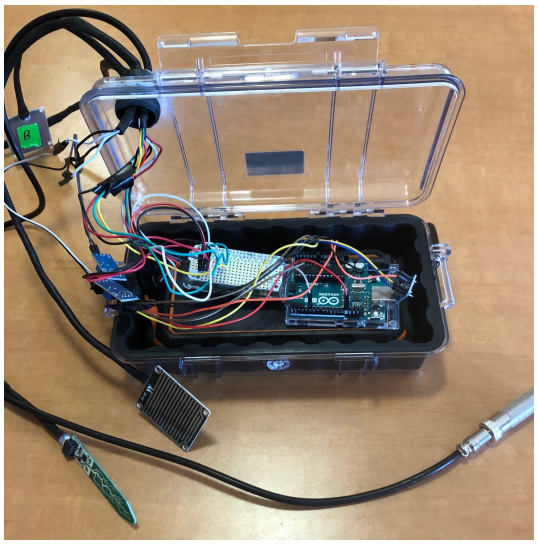

Figure 1: Final Node with sensors

4.3 Software

4.3.1 Raspberry Pi Client and Node Initial Software: Purpose and Reasoning

4.3.1.1 Installing Raspbian OS All the Raspberry Pis used in the project are running Raspbian OS. Raspbian is a version of Linux designed to work on Raspberry Pi's and comes preloaded with software and community support that helped with developing the wireless Sensor Network.

We modified the WiFi configurations on the node to password protect it and encrypt the communication channel between it and the central node using WiFi Protected Access with Extensive Authentication Protocol (WPA-EAP) to provide confidentiality and integrity to the system. The full configurations are shown in Figure 2

```
ctrl_interface=DIR=/var/run/wpa_supplicante GROUP=netdev
update_config=1
country=US

network={
        ssid="(BoonesRun2)"
        key_mgmt=NONE
}

network={
        ssid="JMU-Official-Wireless"
        proto=RSN
        key_mgmt=WPA-EAP
        pairwise=CCMP
        auth_alg=OPEN
        identity="bowma3jm"
        password="Ventus14$$"
        phase1="pealable=0"
        phase2="auth=MSCHAPV2"
        priority=1
}
```

Figure 2: Edited wpa supplicant.conf file

4.3.1.2 Utilization of Python All codes used in the node are written in Python3. "Adafruit Blinka" provides the layer that translates the python RPi.GPIO library [18][19]. This means that any code we have for CircuitPython will be instantly and easily able to run on Linux.

4.3.1.3 Parsing the Sensor Data The data collected from different sensors was written to a Comma Separated Value (CSV) file. CSV data is easily time stamped, stored, and parsable. This is especially important for collecting and analyzing data over time.

The CSV file has 8 columns created and labeled as "time", "Node_Temperature", "Air_Temperature","Air_Humidity", "Soil_Temperature", "Soil_moisture", "Rain", and "UV" for their respective sensor data. An example of how the sensor data is written to a csv file can be seen in Figure 3

	A	B	C	D	E	F	G	H
1	Time	Node_Temperature	Air_Temperature	Air_Humidity	Soil_Temperature	Soil_Moisture	Rain	UV
2	13:41:27	22.5 C	19.8 C	50.30%	24.29040611	322	dry	0.03

Figure 3: CSV file with encoded data

4.3.1.4 Code Running on Boot We wanted to have the attached sensors to collect and store environmental data as soon as they are put in place. Additionally, we wanted to have the nodes begin to search for the UAV's wireless network as soon as possible and connect as soon as it entered within WiFi range. This process was summarized and adapted from [20].

4.3.2 Raspberry Pi Node Sensors Setup

4.3.2.1 Enabling The STEMMA Soil Sensor, DS18b20 Temperature Sensor, and SHT-30 Temperature/Humidity Sensor We first tested each of the the sensors independently for recognition on the Raspberry Pi. Once the Raspberry Pi was able to successfully and continuously detect the sensors we were able to test each sensor for functionality. All devices were then tested to make sure that their measured value was matching with the expected value. For example, since the STEMMA Soil Sensor tested for different levels of capacitance between points, we tested this sensor by placing it in water and noticing a jump in moisture.

4.3.2.2 Setting up the Arduino Uno As mentioned above, we used an Arduino Uno to connect the rain sensors and the UV sensor to the node. We connected two FC-37 Rain Sensors and a SI1145-UV sensor to the Arduino Uno using the I2C protocol and tested them for functionality. We then connected the Arduino Uno to the Raspberry Pi 0 through USB.

4.3.2.3 Socketing Between the Node and Drone A simple socket connection was used to transfer data between the Node (Pi 0) and the Drone (Pi 3). As seen in Figure 4, first the host (Pi 3) ip and port are established for the socket. The code only runs if the static ip addresses are correct, which adds another layer of security to the system.

If the IP addresses match correctly, a socket or tunnel is then established between the two devices. Our client is then able to send the CVS file containing the collected sensor data over the secured WiFi connection. This code is run indefinitely once it is started and currently only stops once a connection has been established and the necessary files are sent.

4.3.3 Raspberry Pi Drone Set Up

4.3.3.1 Socketing Between Drone and Node The Raspberry Pi 3 acts as a wireless Access Point (AP). Once a device, in our case the Raspberry Pi 0, is connected to the AP; the Raspberry Pi 3 would assign it an IP and begin the socketing process. A simple socket connection was used to transfer data between

```
print("searching for drone")
import socket
import netifaces
import time

done = True
S = socket.socket
host = '192.168.2.2'
port = 12345
data = open('home/pi/csvsensordata.csv', 'r+')
bdata = data.read().encode()

def is_interface_up(interface):
  addr = netifaces.ifaddresses(interface)
  return netifaces.AF_INET in addr

while done:
  if is_interface_up('wlan0'):
    IPAddr = netifaces.ifaddresses('wlan0')[netifaces.AF_INET][0]['addr']
    print(IPAddr)
    if IPAddr == '192.168.2.12':
      s.connect((host, port))
      s.send(bdata)
      print("sent csv to drone")
      done = False
    else:
      print('Wrong IP')
      time.sleep(30)
  else:
    print('No Connection, sleeping 30 sec')
    time.sleep(30)
print('exit')
open('/home/pi/csvsensordata.csv', w+).close()
```

Figure 4: Client code on the Raspberry Pi 0

the node (Raspberry Pi 0) and the UAV (Raspberry Pi 3). The socketing code was then modified to allow more than one socket connection and file transfer, this code can be seen in Figure 5.

5 System Components

In this section we discuss of different components in the system were physically and digitally implemented. We also go into the specifics of how the devices were used. Again, we will primarily focusing on the soil quality node as well as the drone.

5.1 Hardware

5.1.1 Soil Quality Node

All sensors were first tested individually for functionality through the use of jumper cables and breadboards. Once functionality was confirmed on each sensor, we made sure that they were able to collect data through the same command at the same time. At this point we had the results write to two CSV files. This process was then made to run on startup of the system and run indefinitely. Specifics of the methodology can be seen in Section 4. The Sensors were then soldered to a printable circuit board to allow for a more compact and secure wiring.

```
import socket
import time
s=socket.socket
host = '192.168.2.2'
print(host)
port = 12345
tst = open('rec_data.csv', 'w')
tst2 = open('rec_data.txt', 'w')
s.bind((host, port))
s.listen(5)
while True:
  c, addr=s.accept
  print('Received connection from', addr)
  data = c.recv(1024)
  tst.write(data.decode())
  print('Done')
  if not data: break
  c.send('Thanks for connecting'.encode())
  if data: exit()
  c.close()
  tst.close()
time.sleep(30)
while True:
    c,addr=s.accept
    print('Received connection from', addr)
    data = c.recv(1024)
    tst2.write(data.decode())
    print('Done')
    if not data: break
    c.send('Thanks for connection'.encode())
    if data: exit()
    c.close()
    tst.close()
```

Figure 5: Host code on the Raspberry Pi 3

5.1.1.1 Adafruit STEMMA Soil Sensor The soil moisture sensor was physically wired to raspberry pi by connecting the following pins:

- Pi 3V3 to sensor VIN
- Pi GND to sensor GND
- Pi SCL to sensor SCL
- Pi SDA to sensor SDA

5.1.1.2 SHT-30 Temperature/Humidity Sensor The outer temperature and humidity sensor was physically wired to raspberry pi by connecting the following pins:

- Pi GND to sensor GND
- Pi 3V3 to sensor VIN
- Pi SDA to sensor SDA
- Pi SCL to sensor SCL

5.1.1.3 DS18b20 Temperature Sensor The inner temperature sensor was physically wired to raspberry pi by connecting the following pins:

- Pi GND to sensor GND

- Pi GPIO7 to sensor SIG
- Pi 5V to sensor VIC

5.1.1.4 Connecting the Arduino Uno to the Raspberry Pi We used an Arduino Uno to connect the sensors because the rain sensors and UV sensor use analog sensor values and The Raspberry Pi does not have a way to read analog inputs. We connected the Arduino Uno to the Raspberry Pi 0 through USB.

5.1.1.5 Connecting the FC-37 Rain Sensors to the Arduino Uno The sensors we used to measure rainfall are the FC-37 Rain Sensors. They are weather-proof and use conductivity to measure the amount of water that is landing on the sensor. We used two rain sensors to measure if the rainfall was equal around the entire node. The rain sensors were physically wired to arduino uno by connecting the following pins: Rain Sensor A:

- Arduino 5V to sensor VCC
- Arduino GND to sensor GND
- Arduino D0 to sensor D4
- Arduino A0 to sensor A0

Rain Sensor B:

- Arduino 5V to sensor VCC
- Arduino GND to sensor GND
- Arduino D1 to sensor D4
- Arduino A1 to sensor A0

5.1.1.6 Connecting the SI1145-UV Sensors to the Arduino Uno The sensor we used to measure UV was the SI1145-UV Sensor. The sensor was placed inside of the node and its function is to check if the node is getting enough UV rays to charge the solar powered battery. The UV sensor were physically wired to arduino uno by connecting the following pins:

- Arduino 3.3V to sensor 3Vo
- Arduino GND to sensor GND
- Arduino A5 to sensor SCL
- Arduino A4 to sensor SDA

5.2 Software

This section covers the specifics on how we set up the raspberry pi's components to correctly interact with the pi itself. Specifics of the methodology can be seen in Section 4 The complete code running on the Soil Quality Node can be seen in Section 9.

5.2.1 Soil Quality Node

5.2.1.1 Installing Raspbian OS NOOBS was used as a way of installing raspbian OS onto the raspberry pi. "NOOBS is an easy operating system installer" which contains the Raspian OS which we will be running on all our Raspberry Pi's. Raspian, as defined by its official website, "is the Foundation's" (Raspberry Pi's) "official supported operating system" and comes pre-installed with "with plenty of software for education, programming and general use. It has Python, Scratch, Sonic Pi, Java and more." Raspbian is a version of the Linux distribution. Once Raspian was downloaded as a zip file onto a separate computer, the file was then transferred to a micro sd card which will serve as all of the Raspberry Pi's system storage. It is important to note that the micro sd card must be formatted beforehand as a fat32 device. For sd cards larger than 32gb additional steps must be taken. For this project we only used sd cards of 32 gb or less so these extra steps did not need to be taken. On the newly formatted sd card, the file was then unzipped using Winrar. Other applications may be used to unzip this file, however, Winrar is a free to use software and very easy to use. Once the file has been successfully unzipped, we safely eject the sd card by right clicking the device within the file explorer and selecting "safely eject." The micro sd card was then inserted into the Raspberry Pi Zero.

The steps were then followed on the installation screen GUI of the raspberry pi. Using our networks login credentials, we then logged into our wifi network in order to access the internet. It is important to note that if you are using an enterprise network (such as a school or business) that requires a network id, username, and password, additional steps will need to be taken.

5.2.1.2 Connecting the Raspberry Pi to Wifi Fully installing Raspbian OS requires an internet connection. Using an enterprise network (such as a school or business) that requires a network id, username, and password, requires manually entering the wifi credentials. We manually edited the wpa_supplicant.conf file through the command "sudo nano /etc/wpa_supplicant/wpa_supplicant.conf". We then added the the code at the bottom of the file.

5.2.1.3 Enabling CircuitPython Circuit Python is a variant of Python that can fit on a microcontroller. All code running on the wireless sensor network is written in python3. Python3 is the newest version of the programming language python. Python is a powerful and agile programming language with a lot of online support in terms of coding issue projects.

5.2.1.4 I2C Protocol As seen in the previous section, many of the sensors chosen are interacting with the raspberry pi through the I2C protocol. I2C is a commonly used standard which allows computer chips to talk to one another. I2C utilizes two lines: a clock line and single data line which is used for both sending and receiving data. The Clock line, called SCL, pulses high and low to drive the sending and receiving of bits. The Data line, called SDA, contains the value of a sent or received bit during clock line transitions. Multiple I2C devices can be connected to the same clock and data lines. However the number of I2C devices is limited to 127 unique devices connected to one bus. We enabled I2C on the raspberry pi by navigating a simple graphical interface.

5.2.1.5 SPI Protocol SPI or Serial Peripheral Interface is a communication interface specification used for short distance communication for two devices to send and receive data. Unlike I2C, SPI uses an explicit data input and data output wire instead of sharing a single data wire. Like I2C, there is a clock wire, but with SPI it has the freedom to use speeds of a few kilohertz to hundreds of megahertz. SPI uses 3 to 4 wires for sending and receiving data. A clock line toggles up and down to drive bits being sent and received. A MOSI wire, or 'master output, slave input' is the data output from your board and sent to a connected device. A MISO wire, or "master input, slave output", is for sending data from the device to the board receiving it. A CS wire, or chip select, toggled to tell the chip that it should listen and respond to requests on the SPI bus. Each device typically requires its own unique CS line. We enabled SPI on the raspberry pi by navigating a simple graphical interface.

5.2.1.6 SSH Protocol Much of the interaction with the raspberry pi itself was done through the use of SSH, or "Secure Shell Protocol." SSH allows for secure remote login from one computer to another. This process of SSHing is used within our project to remotely use the raspberry pi without the need for any external peripherals physically connected to the Pi such as a monitor and keyboard. We enabled SSH on the raspberry pi by navigating a simple graphical interface.

5.2.1.7 Code Running on Boot We wanted to have the attached sensors collect and store environmental data as soon as the node was powered. Additionally, we wanted to have the nodes begin to search for the drones wireless network as soon as possible and connect as soon as it entered within WiFi range. In order to get the Raspberry Pi to run code on boot, we first added opening a new terminal to the list of startup tasks. We then had all new terminals run our sensor collection code and socket code once opened.

5.2.2 Drone

5.2.2.1 Mission Planner (Purpose) Ground Station Control: Mission Planner is a full-featured ground station application for the ArduPilot open

source autopilot project. It is a ground station control compatible with planes, rovers and copters. Mission Planner serves to:

- Load the firmware into the autopilot board (i.e. Pixhawk series) that controls your vehicle.
- Setup, configure, and tune your vehicle for optimum performance.
- Plan, save and load autonomous missions into you autopilot with simple point-and-click way-point entry on Google or other maps.
- Download and analyze mission logs created by your autopilot.
- Interface with a PC flight simulator to create a full hardware-in-the-loop UAV simulator.
- Through telemetry hardware, monitor your vehicle's status while in operation.

Mission Planner served as a reliable and capable ground station control program throughout the duration of the project. Any issues we encountered were solved by online forums and ArduPilot documentation.

5.2.2.2 Mission Planner (Calibration) Note: The propellers were taken off for all calibration processes. After the UAV was fully assembled, it was calibrated. The UAV calibration was done via the Mission Planner software. Mission Planner software can be installed onto a Windows PC and was downloaded from the following cite: [21]. Once the Mission Planner software was installed, it was launched and a Micro-USB to USB cable was used to connect the Pixhawk to the Win- dows PC. The Micro-USB end of the cord was plugged into the side of the Pixhawk as seen in Figure 6.

Figure 6: Pixhawk

Upon launching Mission Planner, in the upper right dialogue box 57600 was selected. This number represented the baud rate or speed of communication between the Pixhawk and computer. Then the connect button next to the dialogue boxes was selected to connect the Pixhawk to the computer. See Figure 7 for confirmation the Pixhawk is connected to Mission Planner

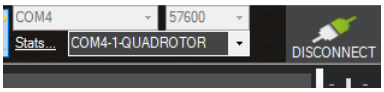

Figure 7: Baud Rate of Connection

On the top pane, the "initial setup" option was selected. Under this tab, there are mandatory hardware configurations shown on the side panel as seen in Figure 8. The "Frame Type" selected was the "X, Y6A" class as it aligned with the configuration of the DJI450 Flamewheel.

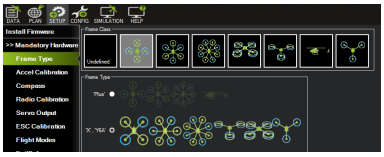

Figure 8: Frame Type Selection for Drone

Next, the "Accel Calibration" tab was selected to calibrate the accelerometer. The steps to calibrate the accelerometer were followed from the following link: [22].

Next, the compass was calibrated. To calibrate the compass, the "Compass" tab on the left hand side of the window was selected. The step-by-step instructions that were followed to calibrate the compass can be found on the following link: [23]. Refer to Figure 9 for evidence that the LED light on the Pixhawk changed to green to indicate that it had a GPS lock.

Figure 9: GPS Lock Confirmation on Drone

To confirm via Mission Planner that the GPS is locked and tracking correctly, the GUI will display the drone and the direction it is facing. See Figure 10; the orange line shows the direction the drone is facing.

Next, the "Radio Calibration" tab was chosen to calibrate the remote control. The step-by-step instructions that were followed to calibrate the remote

Figure 10: Mission Planner GUI showing direction drone is facing

control can be found on the following link: [24].

Then the electronic speed controllers (ECS) that control the spinning of the motors at the speed requested by the autopilot were calibrated. The propellers were not attached during this process. The step-by-step instructions that were followed to complete the ECS calibration can be found at [25]. The "All at Once Calibration" was used as the means of ESC calibration throughout the project.

To maintain control of the UAV throughout flight, the FailSafe mechanism was engaged within Mission Planner [26]. To access the FailSafe mechanism within Mission Planner, the "initial setup" option was selected. The FailSafe tab was selected from the left-hand side of the screen, see Figure 11.

Figure 11: Failsafe Tab

Next, the "Return to Land" (RTL) option was selected for "Battery" and

"Radio" (reference Figure 12). If the battery level threatens to fall below 13 mAh, the UAV will automatically return to the launch location based upon GPS coordinates. Similarly, if the UAV flies beyond the range of communication with the radio controller, the UAV will automatically return to the launch location based upon GPS coordinates. To test the functionality of the FailSafe, consult the following website: [22].

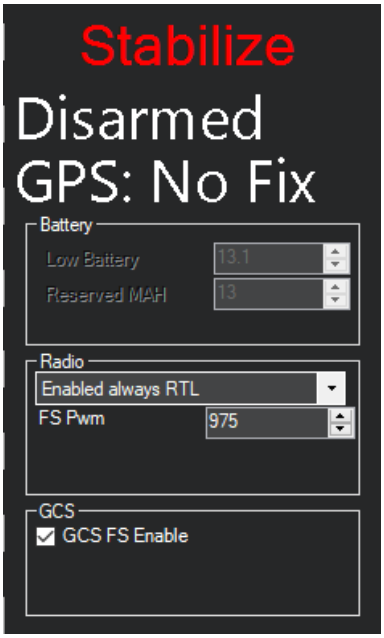

Figure 12: Return to Launch Functionality

A failsafe redundancy chosen to ensure the drone does not stray too far off course is the geofence functionality on the planning GUI of Mission Planner. The Plan tab was selected on the top menu bar then the button circled in red in Figure 13 was clicked and the "Draw a Polygon" option selected.

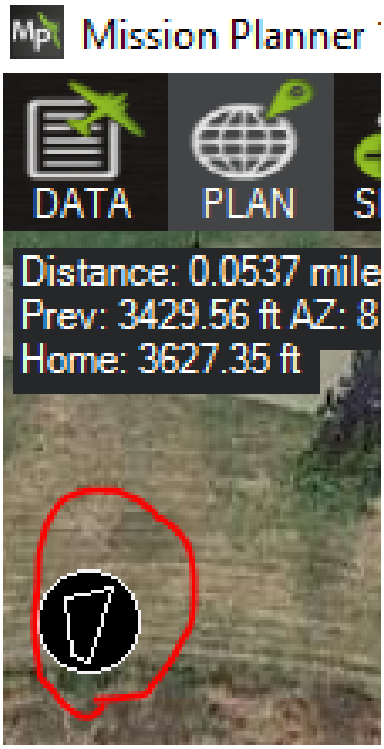

Figure 13: Geofence button

The polygon is then drawn around the intended mission waypoints. If the drone ventures beyond these limits, the Return to Launch function is enabled and the drone returns to the Home location. See Figure 14.

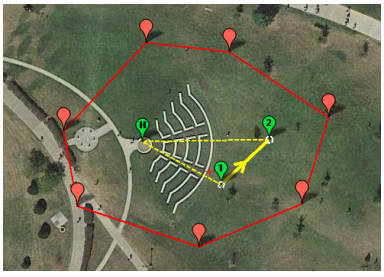

Figure 14: Geofence Polygon Boundary

Next, the Telemetry system is configured. A 915 MHz 3DR system was used. Ensure the drone radio is securely attached to the frame. Plug in the complimentary radio into the machine running Mission Planner via USB port. Go to the "Optional Hardware" option in the SetUp tab and click Sik Radio.

Ensure the two Net ID's match up as well as the Max Window, See Figure 15 for reference.

Figure 15: Telemetry System Set Up

Click Load Settings then click Copy required to remote. To ensure the Telemetry system is operating properly, both radios must be blinking green. See figure 16 for reference.

Figure 16: Blinking Green Light on Telemetry system proves it is operating correctly

Finally, test each individual motor. Ensure propellers are OFF for this calibration. Go to "Motor Test" in the Optional Hardware section under the SetUp tab. Click "Test all in Sequence". Each motor will then spin for 2 seconds each at a power output of 5 percent. See Figure 17 for reference.

Figure 17: Motor Test Set Up Page

5.2.2.3 Mission Planner (Starting the UAV) Before starting the UAV, the battery had to be sufficiently charged. The battery type used was a Venom 3s LiPo battery, meaning that it had three cells. It was charged using a LiPo Battery Balance Charger set to 3.0A and 11.1V(3s). See Figure 18 for charging configuration.

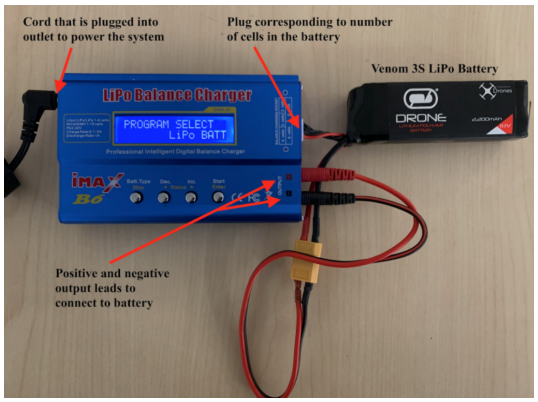

Figure 18: Configuration for charging Lithium Batteries

To begin charging the battery, follow these steps: Program Select -¿ LiPo BATT, then Start was pressed twice. Once the left value of amperage was blinking, and 3.0A was selected, Start was pressed. The right value of voltage began blinking. The option 11.1V(3s) was selected. To begin charging the battery, the Start button was held down until the monitor display changed, and then Start was selected again. When the battery was fully charged, a “FULL” message was displayed on the monitor. Expect charging to take 20 minutes from empty to full.

To start the UAV, the battery was plugged in to power on the UAV. The male XT60 end of the battery was plugged into the female end of the XT60 plug on the APM power module. The battery was secured between the upper and lower plates using a zip tie.

Once the battery is connected, the buzzer will begin to beep. Then the default failsafe must be turned off. A blinking red light will flash on the failsafe, to turn it off hold the button until a single long beep is emitted and the flashing red turns solid. See Figure 19.

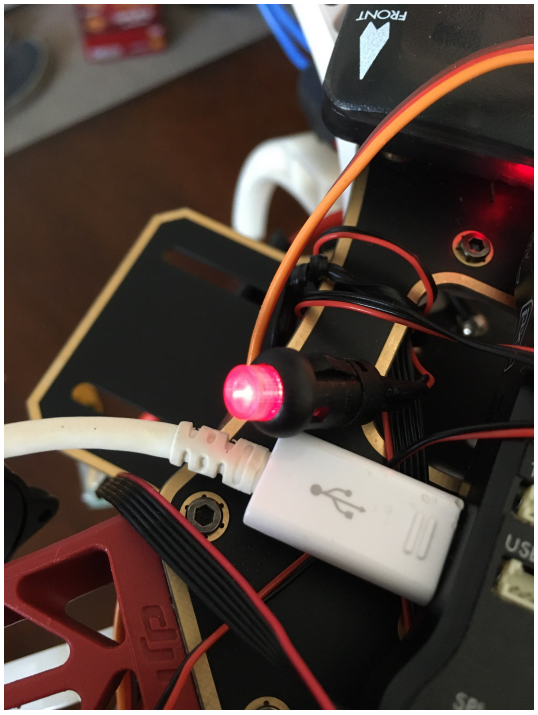

Figure 19: Solid Red failsafe button meaning the drone is ready to read info and be armed

Once the failsafe is solid red, turn on the remote control. The UAV was ready to arm after the LED light on the Pixhawk started blinking blue. (If GPS is locked on, then light will be green). Both blue and green lights mean the drone is ready to be armed. In order to ensure that the UAV would connect to the receiver, the battery had to be sufficiently charged and the UAV had to be positioned upon a level surface. Refer to Figure 20.

Figure 20: Drone armed and ready to fly

The UAV was then armed by using the left controller on the remote control and holding it down and to the right as seen in Figure 21. The motors then started turning and the UAV was ready to fly.

Figure 21: Arming drone via Remote Control

The drone was then manually test flown multiple times. See Figure 22.

Figure 22: Test Flight of Drone

Once the drone is armed, the throttle is increased until the drone takes flight. The ESC calibration spin speeds were calibrated as such: The spin speed when armed is 10 percent of total power. Spin minimum (when any throttle above "Spin when Armed" is applied) is 13 percent total power. Max spin speed is 95 percent total power to ensure the battery maintains its integrity. See Figure 23 for setup via Mission Planner. The "All at Once" Calibration was used [27].

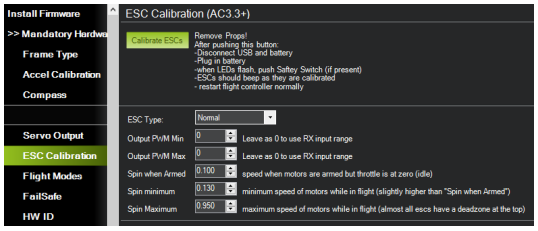

Figure 23: ESC Calibration

Once liftoff has occurred, control the throttle and roll with the left RC stick. Control the pitch and yaw with the right RC stick. When landing, it is recommended to approach the ground very slowly to reduce risk of crashing. Once landed, to disarm the drone move the left RC stick to the extreme bottom left position. Then, unplug the battery. See Figure 24.

Figure 24: Disarming the Drone via Remote Control

Remove propellers.

6 Results

The Nodes we are using are collecting data autonomously and are being placed in the field with battery packs to keep them powered. The battery for the Soil Quality Node can hold 26800 mAh and the battery for the Water Quality Node can hold 12000 mAh and they are both solar powered. We wanted to make sure that the batteries have enough energy to power the nodes while having enough time to be recharged through the solar interface without running out of power. To test that, we performed power, amperage, and voltage measurements on the nodes while all sensors on both nodes are collecting data and writing to the csv file.

While operational, the soil quality node was drawing 5.08 Volts, 0.16 Amps, and 0.81 Watts for the power, current and voltage measurements respectively. While the water quality node was drawing 5.16 Volts, 0.20 Amps, and 1.03 Watts for the power, current and voltage measurements respectively. This means that assuming the batteries received no additional charge from the sun, based on the readings received from the nodes, they would be able to power the entire Soil Quality Node for 167.5 hours and the Water Quality Node 60 hours. The

results are summarized in Table 2 and a comparison between the two nodes can be seen in Figure 25.

Table 2: Comparing power, voltage and energy consumption in both nodes

Unit	Soil Quality Node	Water Quality Node
Volt (V)	5.08	5.16
Current (A)	0.16	0.20
Power (W)	0.81	1.03
No. of Sensors	6	3
Battery Capacity (mAh)	26800	12000
Batter Life (h)	167.5	60

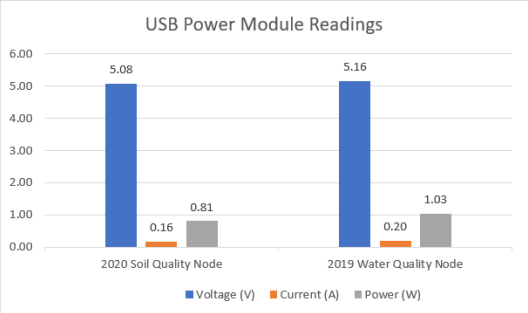

Figure 25: Comparison of power consumption of the sensor nodes

Both nodes where also tested for functionality to ensure a successful simultaneous connection between each of them and the central node mounted on the UAV. The data is only shared once the Raspberry Pi 0 in each of the nodes has a verified given IP address and agreed upon port number. The Raspberry Pi 3 is then able to take hold of the given data file, decode that file and store it for safe delivery once the UAV mission flight is completed.

7 Conclusion & Future Work

In this paper we presented a secure and efficient method to collect data in a WSN. We used two sensor nodes to collect environmental data using water and soil sensors. The sensor data was collected from the nodes through a secure wireless connection established with a central node mounted on a UAV. The data collection nodes are constantly searching for our central node to connect to and recording data at the same time. The secure connection is established once the UAV is within WiFi range of connectivity. The central node is constantly checking for ground nodes to receive data from, this allows for the simple addition of new nodes if we choose to add them. The UAV is setup with a semi-autonomous

flight control to ensure the collection of data effortlessly and efficiently. Our results show that the nodes can be sustained for a long time with the current power source provided without the need of human intervention. This is done through an internal power source that can be initially charged then constantly recharged through solar energy.

While the work completed was able to incorporate aspects of a semi-autonomous flight, future work would be to include the execution of a fully autonomous flight. Semi-autonomous flight requires an operator to manually takeoff and land the copter. Fully autonomous flight would be able to omit the use of an RC and allow the operator to execute any orders via Mission Planner. We also plan on encrypting the stored data using a lightweight post-quantum cryptography algorithm. At the moment we are strongly considering using the GRAIN-128AED. The algorithm is still in development and is a second-round candidate of the current lightweight cryptography project led by the National Institute of Standards and Technology [28]. At the moment we have translated the encryption algorithm to python to seamlessly work with our existing data collection code. We are currently in the process of testing the code for accuracy as well as efficiency both in terms of time and eventually power.

8 Appendix

8.1 Appendix A - Soil Sensor Node Code

```
print("sensor data is being collected")
#!/usr/bin/env python
#-----------------------------------------------------------------
#   Note:
#       ds18b20's data pin must be connected to pin7.
#       replace the 28-000009a239ef as yours.
#-----------------------------------------------------------------
import os
from datetime import datetime
import time
import board
from board import SCL, SDA
import busio
import adafruit_sht31d
from adafruit_seesaw.seesaw import Seesaw
import csv
import serial
c=open('csvsensordata.csv', mode='w+')
with c as csvsensordata:
    date = datetime.today()#pi recognizes current date/time
    csvsensordata_writer = csv.writer(csvsensordata, delimiter=',')
    csvsensordata_writer.writerow(["time",
                                           "Node_Temperature",
                                           "Air_Temperature",
                                           "Air_Humidity",
                                           "Soil_Temperature",
                                           "Soil_moisture",
                                           "Rain",
                                           "UV"])


    f = open(date.strftime('%d.%m.%Y') + 'sensordata.txt', 'a+')#creates a perminant text file with todays date
    data = open('/home/pi/data_temp.txt', 'w+')#creates a temp text file
    f.write("Current Day, Month, Year: ")#starts the perminant text file with current date
    f.write(date.strftime('%d.%m.%Y'))#writes current date to perm text file
    f.write("\nCurrent Hour, Minute, Second: ")
    f.write(date.strftime('%H:%M:%S'))  #writes current time to per text file
    data.write("Current Day, Month, Year: ")#writes date to temp text file
    data.write(date.strftime('%d.%m.%Y'))
```

Figure 26: This shows the soil quality monitoring code part 1/4

```
ds18b20 = ''
i2c = busio.I2C(board.SCL, board.SDA)
outsidetempsensor = adafruit_sht31d.SHT31D(i2c)
i2c_bus = busio.I2C(SCL, SDA)
ss = Seesaw(i2c_bus, addr=0x36)
ser=serial.Serial("/dev/ttyACM0",9600)
def setup():
    global ds18b20
    for i in os.listdir('/sys/bus/w1/devices'):
        if i != 'w1_bus_master1':
            ds18b20 = i


def read():
#   global ds18b20
    location = '/sys/bus/w1/devices/' + ds18b20 + '/w1_slave'
    tfile = open(location)
    text = tfile.read()
    tfile.close()
    secondline = text.split("\n")[1]
    temperaturedata = secondline.split(" ")[9]
    temperature = float(temperaturedata[2:])
    temperature = temperature / 1000
    return temperature

def loop():
    loopcount = 0
    while True:
        if read() != None:

            #print("\nTime: ", date.now().strftime('%H:%M:%S'))

            #node temp sensor
            #print("Node temperature : %0.3f C" % read())
            f.write("\nNode_Temperature : %0.3f C" % read())
            f.write(date.now().strftime(' @%H:%M:%S'))
            data.write("\nNode_Temperature : %0.3f C" % read())
            data.write(date.now().strftime(' @%H:%M:%S'))
```

Figure 27: This shows the soil quality monitoring code part 2/4

```
            #outside temp sensor
            #print("Air_Temperature: %0.1f C" % outsidetempsensor.temperature)
            #print("Air_Humidity: %0.1f %%" % outsidetempsensor.relative_humidity)
            f.write("\nAir_Temperature: %0.1f C" % outsidetempsensor.temperature)
            f.write("\nAir_Humidity: %0.1f %%" % outsidetempsensor.relative_humidity)
            data.write("\nAir_Temperature: %0.1f C" % outsidetempsensor.temperature)
            data.write("\nAir_Humidity: %0.1f %%" % outsidetempsensor.relative_humidity)
            loopcount += 1


            # every 10 passes turn on the heater for 1 second
            if loopcount == 10:
                loopcount = 0
                outsidetempsensor.heater = True
                #print("Sensor Heater status =", outsidetempsensor.heater)
                time.sleep(1)
                outsidetempsensor.heater = False
                #print("Sensor Heater status =", outsidetempsensor.heater)

            #arduino sensors
            read_serial = ser.readline()
            print(read_serial.decode("utf-8"))
            f.write("\nArduino Readings: " + read_serial.decode("utf-8"))
            data.write("\nArduino Readings: " + read_serial.decode("utf-8"))
```

Figure 28: This shows the soil quality monitoring code part 3/4

```
            #soil moisture sensor
            # read moisture level through capacitive touch pad
            touch = ss.moisture_read()
               # read temperature from the temperature sensor
            temp = ss.get_temp()
            #print("Soil_Temperature: " + str(temp))
            f.write("\nSoil_Temperature: " + str(temp))
            data.write("\nSoil_Temperature: " + str(temp))
            #print("Soil_moisture: " + str(touch))
            f.write("\nSoil_moisture: " + str(touch))
            data.write("\nSoil_moisture: " + str(touch))
            csvsensordata_writer.writerow([date.now().strftime('%H:%M:%S'),
                                           "%0.3f C" % read(),
                                           "%0.1f C" % outsidetempsensor.temperature,
                                           "%0.1f %%" % outsidetempsensor.relative_humidity,
                                           str(temp),
                                           str(touch),
                                           read_serial.decode("utf-8")])


            f.flush()
            data.flush()
            c.flush()

def destroy():
    pass

if __name__ == '__main__':
    try:
        setup()
        loop()
    except KeyboardInterrupt:
```

Figure 29: This shows the soil quality monitoring code part 4/4

9 Acknowledgments

We would like to thank Brooke Potter, Gina Valentino, and Laura Yates for their efforts in the initial phase of the project. We also like to thank Safaa Al Badry, lab manager in the Department of Integrated Science and Technology for his technical assistance, and the faculty from the James Madison University X-Labs, especially Kevin Giovanetti for his technical assistance and equipment. We would also like to thank our advisor Dr. Salman organizing this project for us to work on as well as always pushing us in the right direction when roadblocks accrued. Lastly, we would like to give a special thanks to the College of Integrated Science and Engineering for funding and providing resources to the research project.

10 References

[1] P. B. Air. (). 33 eye-opening drone stats - key trends for 2019, [Online]. Available: https://www.phillybyair.com/blog/drone-stats/.

[2] Y. Zeng, R. Zhang, and T. J. Lim, "Wireless communications with unmanned aerial vehicles: Opportunities and challenges," *IEEE Communications Magazine*, vol. 54, no. 5, pp. 36–42, 2016.

[3] B. Potter, G. Valentino, L. Yates, T. Benzing, and A. Salman, "Environmental monitoring using a drone-enabled wireless sensor network," *2019 Systems and Information Engineering Design Symposium (SIEDS)*, pp. 1–6, 2019.

[4] (). Soil moisture, [Online]. Available: https://www.drought.gov/drought/data-maps-tools/soil-moisture.

[5] T. C. Kaspar and W. L. Bland, "Soil temperature and root growth," vol. 154, no. 4, pp. 290–299, 1992.

[6] (). Remarks by the president at the national defense university, [Online]. Available: https://obamawhitehouse.archives.gov/the-press-office/2013/05/23/remarks-president-national-defense-university.

[7] (). The efficacy and ethics of u.s. counterterrorism strategy.

[8] (). The legal and ethical implications of drone warfare, [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/13642987.2014.991210?scroll=top&needAccess=true.

[9] (). Faa takes initial steps to introduce private drones u.s. skies.

[10] J. Benham. (). Using drones for construction, engineering, and surveying, [Online]. Available: https://jamesbenham.com/drones-for-construction-architecture-engineering-surveying.

[11] (). 5 jobs that could be taken over by drones today — make: ..., [Online]. Available: https://makezine.com/2015/03/19/5-jobs-taken-drones-today/.

[12] (). Drone flying cars will soar over dubai this summer, [Online]. Available: https://www.popularmechanics.com/flight/drones/news/a25196/ehang-dubai/.

[13] L. Ada. (). Adafruit stemma soil sensor - i2c capacitive moisture sensor, [Online]. Available: https://learn.adafruit.com/adafruit-stemma-soil-sensor-i2c-capacitive-moisture-sensor/python-circuitpython-test.

[14] T. DiCola. (). Adafruit sht31-d temperature & humidity sensor breakout, [Online]. Available: https://learn.adafruit.com/adafruit-sht31-d-temperature-and-humidity-sensor-breakout/python-circuitpython.

[15] Mazentop. (). Lesson 26 ds18b20 temperature sensor, [Online]. Available: https://www.sunfounder.com/learn/sensor-kit-v2-0-for-raspberry-pi-b-plus/lesson-26-ds18b20-temperature-sensor-sensor-kit-v2-0-for-b-plus.html.

[16] MisterBotBreak. (). How to use a rain sensor, [Online]. Available: arduino.cc/projecthub/MisterBotBreak/how-to-use-a-rain-sensor-bcecd9.

[17] L. da. (). Adafruit si1145 breakout board - uv index / ir / visible sensor, [Online]. Available: learn.adafruit.com/adafruit-si1145-breakout-board-uv-ir-visible-sensor/wiring-and-test.

[18] (). Adafruit-blinka, [Online]. Available: Accessed%20April%201,%202020.%20https://pypi.org/project/Adafruit-Blinka.

[19] (). Rpi.gpio, [Online]. Available: https://pypi.org/project/RPi.GPIO.

[20] (). Five ways to run a program on your raspberry pi at startup, [Online]. Available: https://www.dexterindustries.com/howto/run-a-program-on-your-raspberry-pi-at-startup/.

[21] (). Installing mission planner (windows), [Online]. Available: http://ardupilot.org/planner/docs/mission-%20planner-installation.html.

[22] (). Accelerometer calibration in mission planner, [online], [Online]. Available: http://ardupilot.org/copter/docs/common-accelerometer-calibration.html.

[23] (). Compass calibration, [online]], [Online]. Available: http://ardupilot.org/copter/docs/common-compass-calibration-in-mission-planner.html.

[24] (). Radio control calibration in mission planner, [online], [Online]. Available: http://ardupilot.org/copter/docs/common-radio-control-calibration.html.

[25] (). Electronic speed controller (esc) calibration, [online], [Online]. Available: http://ardupilot.org/copter/docs/esc-calibration.html.

[26] (). Radio failsafe, [online], [Online]. Available: http://ardupilot.org/copter/docs/radio-failsafe.html.

[27] ().
electronic speed controller (esc) calibration¶." electronic speed controller (esc) calibration - copter documentation], [Online]. Available: ardupilot.org/copter/docs/esc-calibration.html.

[28] T. Meltem. (). On the nist lightweight cryptography standardization, [Online]. Available: https://csrc.nist.gov/CSRC/media/Presentations/on-the-nist-lwc-standardization/images-media/Talk-Elliptic-Curve-Crypto-Meltem_Dec2019.pdf.

11 Author Information

John Bowman, Undergraduate Student, Integrated Science and Technology, James Madison University

Jordan Brooks, Undergraduate Student, Integrated Science and Technology, James Madison University

Chandler Lopez, Undergraduate Student, Integrated Science and Technology, James Madison University

Anaseli Marcos-Martinez, Undergraduate Student, Integrated Science and Technology, James Madison University