University of Alabama in Huntsville
LOUIS

Honors Capstone Projects and Theses | Honors College

4-24-2022

Smart Farm

Nolan Patrick Anderson

Follow this and additional works at: https://louis.uah.edu/honors-capstones

 Part of the Other Electrical and Computer Engineering Commons

Recommended Citation

Anderson, Nolan Patrick, "Smart Farm" (2022). *Honors Capstone Projects and Theses*. 684.
https://louis.uah.edu/honors-capstones/684

This Thesis is brought to you for free and open access by the Honors College at LOUIS. It has been accepted for inclusion in Honors Capstone Projects and Theses by an authorized administrator of LOUIS.

Smart Farm

by

Nolan Patrick Anderson

An Honors Capstone

submitted in partial fulfillment of the requirements

for the Honors Diploma

to

The Honors College

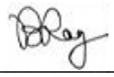
of

The University of Alabama in Huntsville

April 24, 2022

Honors Capstone Director: Dr. Biswajit Ray

Assistant Professor in the Department of Electrical and Computer Engineering

	04/18/2022
Student (signature)	Date
	04/18/2022
Director (signature)	Date
	4/21/22
Department Chair (signature)	Date
William Wilkerson Digitally signed by William Wilkerson Date: 2022.04.24 14:18:07 -05'00'	
Honors College Dean (signature)	Date

HONORS COLLEGE
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Honors College
Frank Franz Hall
+1 (256) 824-6450 (voice)
+1 (256) 824-7339 (fax)
honors@uah.edu

Honors Thesis Copyright Permission

This form must be signed by the student and submitted as a bound part of the thesis.

In presenting this thesis in partial fulfillment of the requirements for Honors Diploma or Certificate from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department, Director of the Program, or the Dean of the Honors College. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

Nolan Anderson

Student Name (printed)

Student Signature

04/18/2022

Date

Table of Contents

- Abstract……1
- Introduction…..…….1
- Chapter 1: Smart Farm Project Summary.…….…..…….4
    - Section One: Network Overview.……5
    - Section Two: Cost and Hardware Components.…….…….6
    - Section Three: Software Implementation.…..…….7
    - Section Four: Field Test.…….…….…….…….……9
    - Section Five: GPS Location.……..10
- Chapter 2: Excluded Topics and Self-Evaluation……..11
- Conclusion.……....12
- Bibliography.…….13
- Appendix.……...14

Abstract

Throughout history, farming has been at the forefront of technological advancement out of necessity to overcome the immense challenges of food production. As population increases and rural work decreases, there is an ever-increasing need for new technologies to automate and assist farming techniques. The Internet of Things is an expanding market of network capable devices designed to connect different aspects of our lives together over the internet. While currently deployed devices can face many challenges, they generate what is called a "Smart Farm" when their solutions help to alleviate the challenges of food production. They provide data, control, management, traceability, and monitoring of crucial aspects of a farm. An outline of the Smart Farm project and current implementation provides an example of a proper Internet of Things system. Future implementations of the product will be deployed to provide farmers with a cost-effective solution to monitoring the current state of soil conditions on a farm.

Introduction

The Internet of Things (IoT) is an ever-increasing popular term, and its functions regularly affect the lives of everyone. The devices and systems that fall under this category are often hidden in plain site which highlights its permanence in society. IoT helps to transform "physical objects into an information ecosystem shared between wearable, portable, and even implantable devices" (Mouha). Fitness apps and health monitoring have become popular functions of wearable tech in recent years. Many rely on this functionality to share data with their doctors and has a direct impact on their health. This has the added benefit of less doctor visits and more up to date information. Any device can have an IoT classification if it contains "any type of sensor embedded with the ability to collect data and transmit it across the network without manual intervention" (Mouha). The definition of IoT devices is intentionally broad and shows how much potential this sector has.

Nearly every person in modern society owns electronic devices, but not all of them are connected to the internet. Companies are beginning to realize and capitalize on this fact, which is why so many new devices have internet capability. For example, traditional cash registers are generally not connected to the internet, but companies such as Square have created products, services, and software that publish sales metrics to the cloud. With proper login credentials, a company can securely and easily access sales information from nearly any device with an internet connection. One of the largest cell-phone manufactures, Huawei, predicts "100 billion IoT connections by 2025 […] estimating the potential economic impact of the Internet of Things from $3.9 to $11 trillion annually in 2025" (Mouha). The potential growth of this sector shows that more IoT devices are coming, and fast. Society is going to have no choice but to rely on them.

There are many potential issues with IoT devices, and many discussions highlight security risks with connecting all our devices to the internet. A company may sell you a device that shares data with your doctor but may steal your personal information for their benefit in the same process. Continually, credit card skimmers have risen in popularity and steal your payment information. While implementations vary, skimmers slip directly on top of card scanners and can quickly send your information to an attacker without your knowledge. These security issues are likely to increase as network capable devices become smaller. Alternatively, as we grow to depend on IoT devices, reliability and durability become major concerns. Many systems' functionality directly relies on the data that IoT devices can provide. A farmer may use network capable cameras to ensure the safety and security of his property and livestock. In an event where the device becomes inoperable, the farmers investment is in jeopardy. If society is to rely on these devices, they need to always function. As a result, they need to be physically robust and secure from malicious intent, such as a hack. Many other issues may become apparent as society continues to adopt and integrate

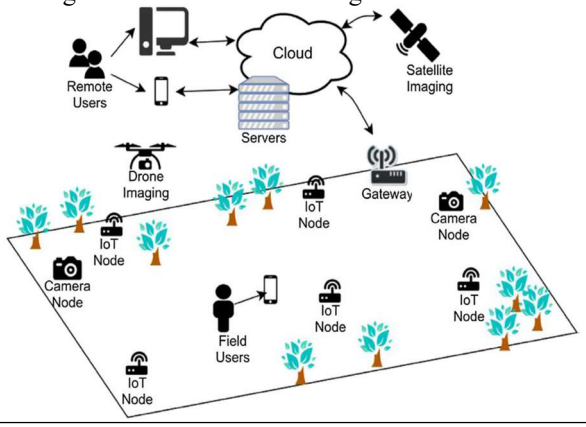
these devices into our daily lives. Therefore, the IoT sector needs to be on the forefront of technology, security, and reliability when designing these devices.

One major challenge society faces today is that "world population growth is increasing the demand for food production. Furthermore, the reduction of the workforce in rural areas and the increase in production costs are challenges for food production" (Navarro et al.). As a result, there is an ever-increasing need for new technologies that increase production and reduce costs and labor. However, farmers around the world have already begun implementing these systems into their businesses. From drones to satellite imagine, there are many existing problems farmers face that IoT solutions can solve. These devices are critical for "Smart Farm" techniques. Figure 1 is a list of the most common applications and solutions IoT technologies provide on a farm (Navarro et al.). Drones can spray pesticides, seed, and fertilizer and take images in hard-to-reach areas of a property. IoT nodes provide accurate and easy access to data that show the general status of a farm. For example, sensors can detect the moisture levels of soil. An irrigation system can then be configured to turn on when moisture levels reach a certain threshold. At this point, a large portion of a farm is completely automated which makes the life of a farmer much easier. Figure 2 depicts an IoT Instrumented Agricultural Field environment. This view does not represent all the current smart farm techniques used today

Figure 1 Common Smart Farm Applications (Navarro et al.)

- Chemical control (e.g., pesticides and fertilizers).
- Crop monitoring.
- Disease prevention.
- Irrigation control.
- Soil management.
- Supply chain traceability
- Vehicles and machinery control.

Figure 2 IoT Instrumented Agricultural Field



but provides insight into the role the Smart Farm project plays. Systems like these provide farmers with the data they need to maintain a healthy farm and business. As you begin to add more devices and sensors around a farm, you begin to gather a wealth of information. This information influences crop health, harvest and planting season, farm automation, and much more. However, installing and maintaining a large range of devices can generate additional cost and maintenance. As a result, IoT solutions need to prioritize robust hardware and software designs along with rigorous testing. A faulty sensor node or software that collects incorrect data could influence incorrect and potentially costly decisions. Continued maintenance and replacement of hardware leads to information gaps and system downtime that drastically increase costs. If installation of a system is obtrusive and costly, the solution may not be a worthwhile investment despite the benefits it may provide. Overall, if farmers have access to cost-effective solution that accurately represent their farm, they can conduct a more efficient and profitable business. The influence and importance of IoT in the agricultural industry will only grow as time progresses.

Chapter 1: Smart Farm Project Summary

Smart Farm is an affordable distributed wireless sensor network system for monitoring soil conditions across a large area. It consists of several intelligent sensor nodes that sense, process, and communicate sensor readings to a local server that is connected to the internet. The sensor nodes are built using Heltec ESP-32 IoT platforms that connect to an array of sensors for sensing soil moisture, nutrients, and environmental conditions. The sensor nodes utilize LoRa radios to connect to a gateway. This gateway manages the network of sensor nodes, samples data from the sensor nodes, and transmits the data to a RaspberryPi which provides services to remote clients. The main aspects of the project are development of a custom impedance analyzer sensor for monitoring soil conditions, developing firmware for sensor nodes to interface various sensors and

communicate with the gateway, developing software for the local server, and integration and testing of the system.

Section 1: Network Overview

The Smart Farm network is comprised of three types of devices: sensor nodes, a gateway, and a server. The sensor nodes are ESP-32 microcontrollers that have a wired connection to various sensors that communicate via I2C protocol. These nodes are configured to measure temperature, humidity, air pressure, altitude, and capacitance. The sensor stack that obtains these measurements is the STEMMA soil sensor and the BME280. The gateway is also an ESP-32 microcontroller. Its purpose is to manage the mesh network of sensor nodes. When it collects data, it forwards the data via a physical serial connection to the server. When a sensor node comes online, it persists in requesting an ID until the gateway assigns one to it. The sensor node includes a randomly generated number in its request. When the gateway broadcasts the ID, it includes the token to ensure that the correct sensor node receives the ID. When the gateway comes online, it broadcasts a reset signal to all the sensor nodes. This tells them to reset their network connection. The gateway further manages the network by coordinating when data is sent, when all nodes enter deep-sleep mode, and how long they sleep. It sends a packet to each node on the network requesting data from it. After it has requested data from each node, it broadcasts a packet that instructs them to fall asleep and provides the duration for their sleep. The gateway stays awake while the connected nodes sleep so that it may admit nodes that are not connected onto the network. Each time a node requests to join the network, the gateway prunes stale connections and admits it to the network. Appendix A5 provides a visual aid of the gateway and node network in a block diagram format. The server powers the gateway via the serial connection between the two devices. It runs a script on startup that parses sensor data that the gateway collects into a comma separated variable file.

The server hosts this data and has a script that produces graphs of the data. The farmer can run these scripts at any time he desires; however, a more user-friendly way to visualize the data exists. The server hosts a website that can be accessed on any device with an internet connection which displays graphs and gives the user filter options. A Raspberry Pi is used as the server due to its power demands, portable design, and has adequate storage space. Figure 3 shows a high-level system view of the network.

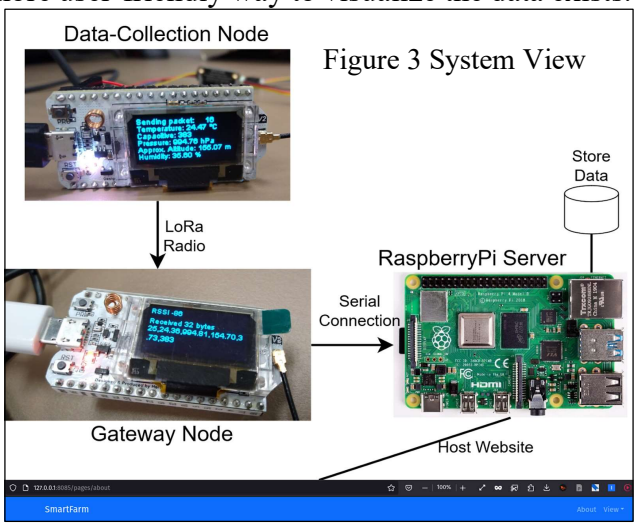


Figure 3 System View

Section 2: Cost and Hardware Components

The Smart Farm project has many hardware requirements; however, this is the current life cycle and is subject to change in the future. The Raspberry Pi, used for storing data from and powering the gateway node, can be purchased for less than $100. The Adafruit Stemma sensor, around $10, collects temperature and moisture from the soil. The Heltec ESP-32 and LORA Radio, sold together, are available for around $20 but price can depend on availability. A BME device, $15 on Adafruit's website, is an interface between the ESP-32 and Stemma Sensor and reads barometric pressure, temperature, and humidity. Figure 4 shows a current implementation of a sensor node but lacks the connection of a power source. The sensor nodes are currently deployed with a MakerFocus 3000mAh to power the ESP-32 devices. A pack of four batteries can be

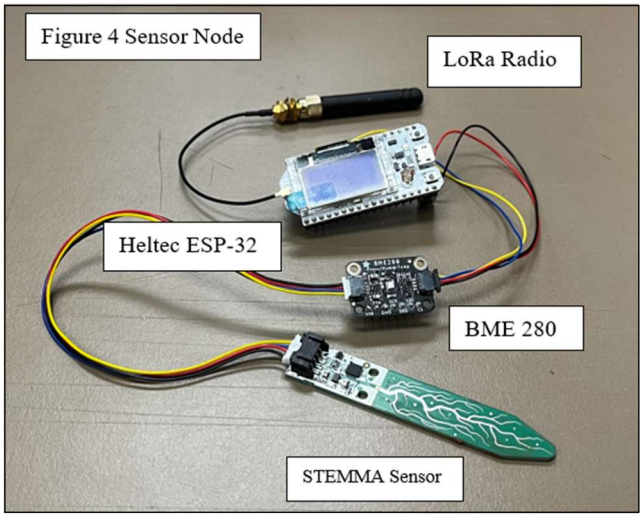


Figure 4 Sensor Node

purchased online for $30 and solar panels were considered to keep them charged. This may be a future requirement, but the cost is currently unknown. Lastly, a box of jumper wires and I2C connectors will be needed to connect the devices. The current hardware cost for a system of four nodes, as seen in our field test, is around $300 and much cheaper than currently available solutions. Hardware requirements will fluctuate with project progress and because the project is in its infancy, estimating final costs is extremely difficult.

Section 3: Software Implementation

Because the sensor nodes are ESP-32's, software is implemented using Arduino Sketch Files (.ino). These files "tell your board what to do by sending a set of instructions to the microcontroller on the board" (File.org). Continually, the open-source Arduino Software (IDE) makes it easy to write and upload code to the board. The first task is to flash a unique hardware ID to the non-volatile memory of each sensor node. In other words, the ESP-32 can lose power or receive software updates without losing its hardware ID. This allows for the ability to distinguish which device corresponds to the data received. The hardware ID is flashed using LoRa_Hardware_ID.ino in appendix A1. Next, the block diagram in Figure 5 outlines the software running on the gateway that requests and receives data from the sensor nodes. The code, Lora_Receive.ino in appendix A3, continually requests and reads packets from each sensor node based on hardware ID. The gateway uses a serial connection to the Raspberry Pi for power and has the added benefit of the execution of a python script. This script, readSerial.py in appendix A4, interfaces with the serial port to store data received into an excel file that is

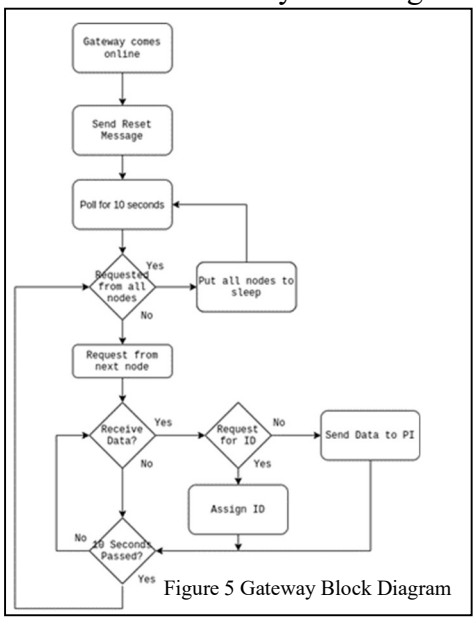


Figure 5 Gateway Block Diagram

received from the gateway. This data is then stored over the Raspberry Pi's network. Lastly, the sensor nodes in the field need a way to collect and send their data. Initially, LoRa_Send.ino (see appendix A2) runs its setup and pulls the hardware ID from the non-volatile memory. The software then interfaces with the BME to gather data from the Stemma Sensor. Lastly, the data is bundled into a packet and sent back to the gateway. In summary, the software outlined above creates its own network between several ESP-32 boards and a Raspberry Pi to store soil data from several different locations. The .ino files and the python script are outlined in the appendix but more comprehensive and up to date versions can be found on the Smart Farm GitHub in the bibliography.

Section 4: Field Test

A field test was conducted on the UAH campus. One node, placed in a tree, lost connection with one of the sensors. However, the software has checks for physical connection issues. As a result, the node displayed an error message detailing what sensor had been disconnected and did not send data to the gateway. Despite that, data was collected from the remaining three sensor nodes. Two

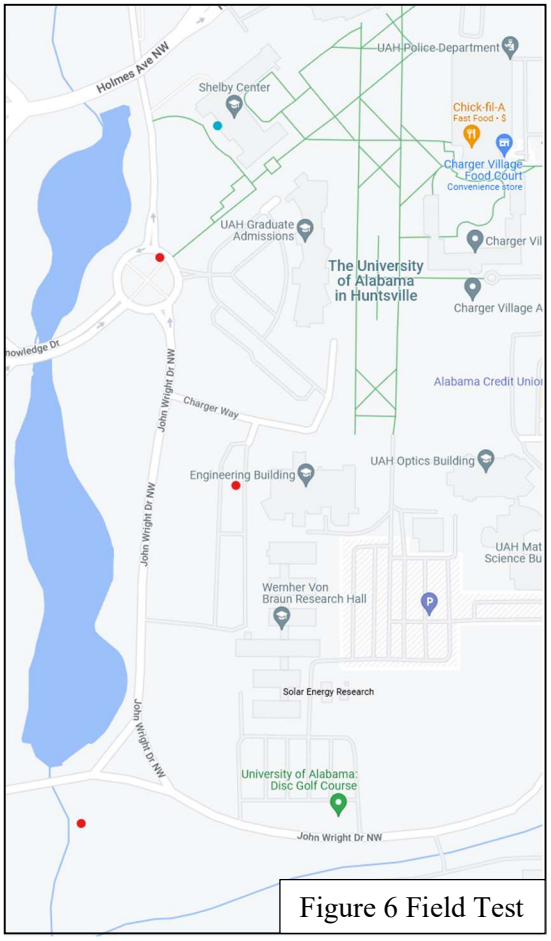


Figure 6 Field Test

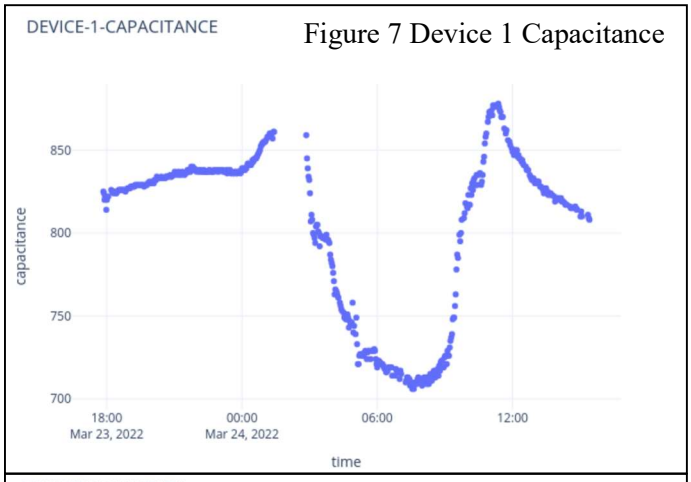


Figure 7 Device 1 Capacitance

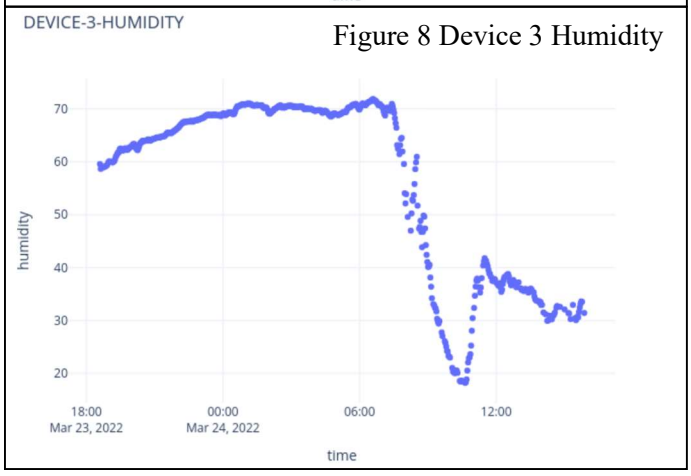


Figure 8 Device 3 Humidity

were placed in the roundabout, one near the Engineering Building, and one south of John Wright Drive NW. The red dots on figure 6 show the location of each sensor node. The gateway, represented with a blue dot, was placed on the Shelby Center balcony. The longest distance between the gateway and any node was around 0.446 miles, or 718.54 meters. A previous senior design team tested the same sensors over a two-mile distance, but the maximum distance is

unknown. Given proper line of sight between the nodes, however, should satisfy the needs of a farm. Figures 7 and 8 show capacitance and humidity data collected from two of the nodes and stored on the Raspberry Pi. Continually, more of this data can be found on the Smart Farm GitHub page. The collection of this data is incredibly important in outlining the overall success of the network implementation.

Section 5: GPS

Ideally, a farmer would like to know the location of their sensor nodes in the field. Between GPS location and hardware ID on the device, it is easy to understand where data is coming from. To accomplish this, we ordered a NEO 6M GPS module that is compatible with the Heltec ESP-32 device. The module has Tx (transmit), Rx (receive), ground, and 3.3-volt pins. After the code runs initialization and setup, it then checks for an available serial connection and can begin polling for satellites. If a satellite connection is made, and GPS data is available, it prints this over the serial connection back to the ESP-32. The code also checks for date and time in a similar fashion. Lastly, code interfaces with the Heltec library to print the data it has collected to the screen. The current implementation can be found in appendix A6, but a more up to date version will reside on the Smart Farm GitHub page. Future implementations would be integrated into the sensor node code. The GPS location would be collected and sent alongside the packet containing the sensor data.

Unfortunately, the GPS module was troublesome to work with. The main issue is that establishing connection with a satellite can take up to 15 minutes. This made the code difficult to test and debug as it was unclear if the device was polling for a satellite. Another issue is the device draws a lot of power even in sleep mode. As a result, the sensor node would always draw excess power after the initial location had been set. In other words, you likely do not want the location of the node every time you receive sensor data. Additionally, the location would need to be updated if the sensor node was moved and would require a reconnection with a satellite. Because of semester time constraints and the shortcomings of the device, this

requirement was dropped. Likely, another GPS solution should be pursued as similar devices will run into the same problems. One potential solution is requiring the farmer to input the location of their sensor nodes. A benefit of this is the device becomes much cheaper and power efficient but adds more burden on the farmer and is likely less accurate.

Chapter 2: Excluded Topics and Self Evaluation

The ideas covered in the sections and chapters above do not reflect the total work completed on the Smart Farm project. The project director, Dr. Ray, is interested in an in-house design to replace the STEMMA soil sensor. This design will be a lower cost solution for measuring soil parameters. Unfortunately, I was not involved with the sensor design as this was separate from the networking portion of the project. As a result, this document does not include information regarding the sensor design or an analysis of the STEMMA sensor readings. A web app to compile and visualize sensor data has been designed from the ground up. The source code can be found on the GitHub link in the bibliography. This web app will be the most important for the end user as the data will be compiled into readable graphs, such as figures 7 and 8. Eventually, farmers will be able to access this information from any device that can connect to the internet. As mentioned in the introduction, data collected by the nodes needs to be stored on a network for a system to truly be in the realm of IoT. This website is key in accomplishing this metric. Lastly, some work has been completed on generating a casing to install all the hardware in a more permanent solution but will not be covered here.

Several points come to mind when considering self-evaluation. The first point is obviously the failure of implementing a functional GPS system. Personally, more work could have been done on my end to ensure that this goal was met. However, I did my best given the time constraints of the semester and the unforeseen complications with the device. When looking back on the sensor

design, no work was contributed on my end as I was focusing more on networking and GPS. Another area I could have contributed more time is helping to test different implementations along the way. While I did contribute to completing the field test, effort was not completed in any other areas of testing. The senior design course requires the completion of many documents, however, and I was a crucial asset in completing these to their fullest extent by the due date. Overall, while I did contribute much of my time to accomplishing the goals of this project, I certainly could have done more by being more present.

Conclusion

Technological advancements play a necessary and critical role for farmers, and in many cases, farmer's needs ignite and support these new inventions. In recent years, there has been an explosion of new IoT capable devices on the market available to farmers. However, many of these solutions can become quite expensive due to the nature of how they are designed. While existing soil sensing products perform similar operations to Smart Farm exist, their price turns many away. Providing farmers with a cost-effective solution for monitoring crop conditions quickly, accurately, and easily is crucial to understanding the importance of the Smart Farm objective. The data collected from the field test, figures 7 and 8, shows that this objective is possible. If done correctly, this system can be deployed virtually anywhere with a proper way to power the Raspberry Pi server. This will prove especially important for farmers who lack the financial ability to purchase more expensive solutions. Many additional requirements to needs to be accomplished for this system, but they could not be completed due to the time constraints of senior design.

Bibliography

File.org. “INO File Type.” *File.Org*, 26 Feb. 2022, file.org/extension/ino.

Mouha, Radouan. “Internet Of Things (IoT).” *Scientific Research An Academic Publisher*, Scientific Research, 1 May 2021, www.scirp.org/journal/paperinformation.aspx?paperid=108574.

Mills, Daniel, et al. (2022) Smart Farm Repository [Source code]. https://github.com/dandeto/Smart-Farm

Navarro, Emerson, et al. “A Systematic Review of IoT Solutions for Smart Farming.” *National Library Of Medicine*, National Library Of Medicine, 29 July 2020, www.ncbi.nlm.nih.gov/pmc/articles/PMC7436012.

Appendix

A1 - LoRa_Hardware_Id.ino

```
#include <Preferences.h>
#define ID 2

void setup() {
        Serial.begin(115200);

        // Initialize interface to NVS
        Preferences p;
        // Open the node namespace
        if(!p.begin("node", false)) {
                Serial.println("This board is broken");
                return;
        }

        unsigned long id = p.getULong("id", 0);

        if(id != 0) Serial.println("ID was: " + String(id));

        p.putULong("id", ID);
        Serial.println("ID is now: " + String(p.getULong("id", 0)));
        p.end();
}
void loop() {}
```

A2 - LoRa_Send.ino

```
/*
  OLED pins to ESP-32 GPIOs:
  OLED_SDA -- GPIO4
  OLED_SCL -- GPIO15
  OLED_RST -- GPIO16
*/

#include <Preferences.h>
#include "heltec.h"
#include "images.h"
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include "Adafruit_seesaw.h"

#define BAND              915E6       // you can set band here directly e.g. 868E6,915E6
#define SEALEVELPRESSURE_HPA (1013.25)
//#define uS_TO_S_FACTOR      1000000ULL // Conversion factor for micro seconds to seconds

unsigned long hardware_id = 0;
RTC_DATA_ATTR int id      = 0; // not erased in sleep mode
long token              = 0;
volatile long rx_token    = 0;
volatile bool rx          = false;
String type;
volatile int rx_id;
Adafruit_BME280 bme;
Adafruit_seesaw ss;
float temperature, pressure, altitude, humidity;
uint16_t capactive;

// This ISR will run even during a delay function
void getPacket(int size) {
  if(size) rx = true;
  type = "";
  type += (char)LoRa.read(); // type is first 2 bytes
  type += (char)LoRa.read();
  rx_id = LoRa.parseInt();
  if(type == "ID") rx_token = LoRa.parseInt();
  //Serial.println("Packet: " + type + " " + String(rx_id) + " " + String(token)); // debug
}

// get and assign sensor data
void setSensorData() {
  temperature = bme.readTemperature();
```

```
  pressure    = bme.readPressure() / 100.0F;
  altitude    = bme.readAltitude(SEALEVELPRESSURE_HPA);
  humidity    = bme.readHumidity();
  capactive   = ss.touchRead(0);
}

// display sensor data to OLED screen
void displaySensorData() {
  Heltec.display->clear();
  Heltec.display->setTextAlignment(TEXT_ALIGN_LEFT);
  Heltec.display->setFont(ArialMT_Plain_10);
  Heltec.display->drawString(0, 0,  id ? "ID: " + String(id) : "Waiting for ID...");
  Heltec.display->drawString(0, 10, "Temperature: " + String(temperature) + " °C");
  Heltec.display->drawString(0, 20, "Capacitive: " + String(capactive));
  Heltec.display->drawString(0, 30, "Pressure: " + String(pressure) + " hPa");
  Heltec.display->drawString(0, 40, "Approx. Altitude: " + String(altitude) + " m");
  Heltec.display->drawString(0, 50, "Humidity: " + String(humidity) + " %");
  Heltec.display->display();
}

void setup() {
  Heltec.begin(true /*DisplayEnable Enable*/, true /*Heltec.Heltec.Heltec.LoRa Disable*/, true /*Serial Enable*/, true /*PABOOST Enable*/, BAND /*long
BAND*/);
  Heltec.display->init();
  Heltec.display->flipScreenVertically();
  Heltec.display->setFont(ArialMT_Plain_10);
  Heltec.display->clear(); // need this?
  Heltec.display->drawXbm(0,5,logo_width,logo_height,logo_bits);
  Heltec.display->display();
  delay(1500);
  Heltec.display->clear();

  Serial.println("ID is" + String(id));

  // Get Hardware ID
  Preferences p;
  while (!p.begin("node", true)) {
    Heltec.display->drawString(0, 0, "Cannot Access NVM\n - This Board May Be Broken");
    Heltec.display->display();
    delay(100);
  }

  hardware_id = p.getULong("id", 0);
  if(hardware_id == 0) {
    Heltec.display->drawString(0, 0, "Board not provisioned.");
    Heltec.display->display();
    while(1);
  }

  p.end();

  // Initialize BME280
  while (!bme.begin(0X76) && !bme.begin(0x77)) {
    Heltec.display->drawString(0, 0, "ERROR! BME not found");
    Heltec.display->display();
    delay(100);
  }

  Heltec.display->clear();

  // Initialize Seesaw
  while (!ss.begin(0x36)) { // i2c address for capacitance sensor is 0x36
    Heltec.display->drawString(0, 0, "ERROR! Seesaw not found");
    Heltec.display->display();
    delay(100);
  }

  /*
   * LoRa.setTxPower(txPower,RFOUT_pin);
   * txPower -- 0 ~ 20
   * RFOUT_pin could be RF_PACONFIG_PASELECT_PABOOST or RF_PACONFIG_PASELECT_RFO
   *   - RF_PACONFIG_PASELECT_PABOOST -- LoRa single output via PABOOST, maximum output 20dBm
   *   - RF_PACONFIG_PASELECT_RFO     -- LoRa single output via RFO_HF / RFO_LF, maximum output 14dBm
```

```
  */
  LoRa.setTxPower(14,RF_PACONFIG_PASELECT_PABOOST);

  LoRa.onReceive(getPacket); // ISR for when receive done
  //LoRa.onTxDone(onTxDone); // ISR for when transmission done
  LoRa.receive(); // put radio in receive mode
}

void loop() {
  while(!id) {
    delay(random(100,5000));
    Heltec.display->clear();
    Heltec.display->drawString(0, 0,  id ? "ID: " + String(id) : "Waiting for ID...");
    Heltec.display->display();
    // ask for id
    do {
      LoRa.beginPacket();
      LoRa.print("ID ");
      token = random(1,247483647);
      LoRa.print(token);
      LoRa.endPacket();
    } while(!LoRa.endPacket());
    LoRa.receive();
    Serial.println("Request ID");
    // poll for 2 seconds
    int count = 0;
    while(type != "ID" && ++count < 3000) {
      delay(10);
    }
    if(type == "ID" && token == rx_token) {
      Serial.println("Get ID");
      id = rx_id;
    }
  }

  // respond to request for data
  if(rx) {
    Serial.println("Packet: " + type + String(rx_id));
    if(type == "RQ" && rx_id == id) { // server request packet
      setSensorData();
      do {
        LoRa.beginPacket();
        LoRa.print("RS"); // packet type - response
        LoRa.print(id);
        LoRa.print(",");
        LoRa.print(hardware_id);
        LoRa.print(",");
        LoRa.print(temperature);
        LoRa.print(",");
        LoRa.print(pressure);
        LoRa.print(",");
        LoRa.print(altitude);
        LoRa.print(",");
        LoRa.print(humidity);
        LoRa.print(",");
        LoRa.print(capactive);
      } while(!LoRa.endPacket()); // returns 1 on success 0 on failure
      LoRa.receive(); // yes, actually need to put it back into receive mode after sending
      displaySensorData();

      // flash LED to indicate response
      digitalWrite(LED, HIGH);
      delay(100);
      digitalWrite(LED, LOW);

    } else if(type == "SL") { // sleep
      // network id should be RTC_DATA_ATTR so it persists...
      esp_sleep_enable_timer_wakeup(rx_id * 1000000ULL);
      Serial.println("Go to Sleep for " + String(rx_id));
      esp_deep_sleep_start();
      // goes back thru void setup() when timer expires
    } else if(type == "RS" && rx_id == 0) { // server reset
      Serial.println("RESET");
```

```
    id = 0; // reset
    do { // request id
     LoRa.beginPacket();
     LoRa.print("ID");
    } while(!LoRa.endPacket());
    LoRa.receive();
   }

   rx = false;
  }

  delay(2000); // wait for 2 seconds
}
```

A3 - LoRa_Receive.ino

```
#include "heltec.h"
#include "HardwareSerial.h"
#include "images.h"
#include "Node.h"

#define BAND           915E6 // set frequency band here,e.g. 868E6,915E6
#define sleep_delay     10   // these are all in seconds
#define node_sleep_delay 5
#define request_delay   8

NodeManager nodeManager;

void displayPacket(String rssi, int packetSize, String packet){
 Heltec.display->clear();
 Heltec.display->setTextAlignment(TEXT_ALIGN_LEFT);
 Heltec.display->setFont(ArialMT_Plain_10);
 Heltec.display->drawString(0, 15, "Received "+ String(packetSize, DEC) + " bytes");
 Heltec.display->drawStringMaxWidth(0 , 26 , 128, packet);
 Heltec.display->drawString(0, 0, rssi);
 Heltec.display->display();
 Serial.println(packet);
}

void readPacket(int packetSize) {
 String type = "";
 type += (char)LoRa.read(); // packet type is first 2 bytes
 type += (char)LoRa.read();

 if(type == "ID") { // request for ID
  // grab token
  long token = LoRa.parseInt();
  if(token) {
   //Serial.println("Recieved token " + String(token)); // debug
   int id = nodeManager.add();
   delay(1000);
   do {
    LoRa.beginPacket();
    LoRa.print("ID ");
    LoRa.print(id);
    LoRa.print(" ");
    LoRa.print(token);
   } while(!LoRa.endPacket());
   LoRa.receive();
   //Serial.println("Sent Packet: ID " + String(id) + " " + String(token)); //debug
  }
 } else if(packetSize) {
  int id = LoRa.parseInt(); // get network id
  LoRa.read(); // throw away comma
  nodeManager.response(id);
  String packet;
  for (int i = 0; i < packetSize; i++) packet += (char) LoRa.read();
  String rssi = "RSSI " + String(LoRa.packetRssi(), DEC);
  displayPacket(rssi, packetSize, packet);
 }
 //nodeManager.print(); // debug
```

```
}

void poll(int _delay) {
 // poll for number of seconds specified by _delay
 int d = 0;
 while(++d < _delay*100) {
  int packetSize = LoRa.parsePacket();
  if (packetSize) readPacket(packetSize);
  delay(10);
 }
}

void setup() {
 Heltec.begin(true /*DisplayEnable Enable*/, true /*Heltec.Heltec.Heltec.LoRa Disable*/, false /*Serial Enable*/, true /*PABOOST Enable*/, BAND /*long
BAND*/);
 Serial.begin(115200);
 Heltec.display->init();
 Heltec.display->flipScreenVertically();
 Heltec.display->setFont(ArialMT_Plain_10);
 Heltec.display->clear();
 Heltec.display->drawXbm(0,5,logo_width,logo_height,logo_bits);
 Heltec.display->display();
 delay(1500);
 Heltec.display->clear();
 Heltec.display->drawString(0, 0, "Heltec.LoRa Initial success!");
 Heltec.display->drawString(0, 10, "Wait for incoming data...");
 Heltec.display->display();
 delay(1000);

 LoRa.beginPacket();
 LoRa.setTxPower(14,RF_PACONFIG_PASELECT_PABOOST);
 LoRa.print("RS "); // tell nodes to reset
 LoRa.print(0);
 LoRa.endPacket();
 LoRa.receive();
}

void loop() {
 poll(10); // poll for 10 seconds

 // Request packet from each sensor node
 int id = nodeManager.next();
 while(id) {
  do {
   Serial.println("Request From ID " + String(id)); // debug
   LoRa.beginPacket();
   LoRa.print("RQ " + String(id));
  } while(!LoRa.endPacket());
  nodeManager.request(id);
  poll(8);
  id = nodeManager.next();
 }

 // Put all nodes to sleep
 do {
  LoRa.beginPacket();
  LoRa.print("SL");
  LoRa.print(node_sleep_delay);
 } while(!LoRa.endPacket());
}
```

A4 - readSerial.py

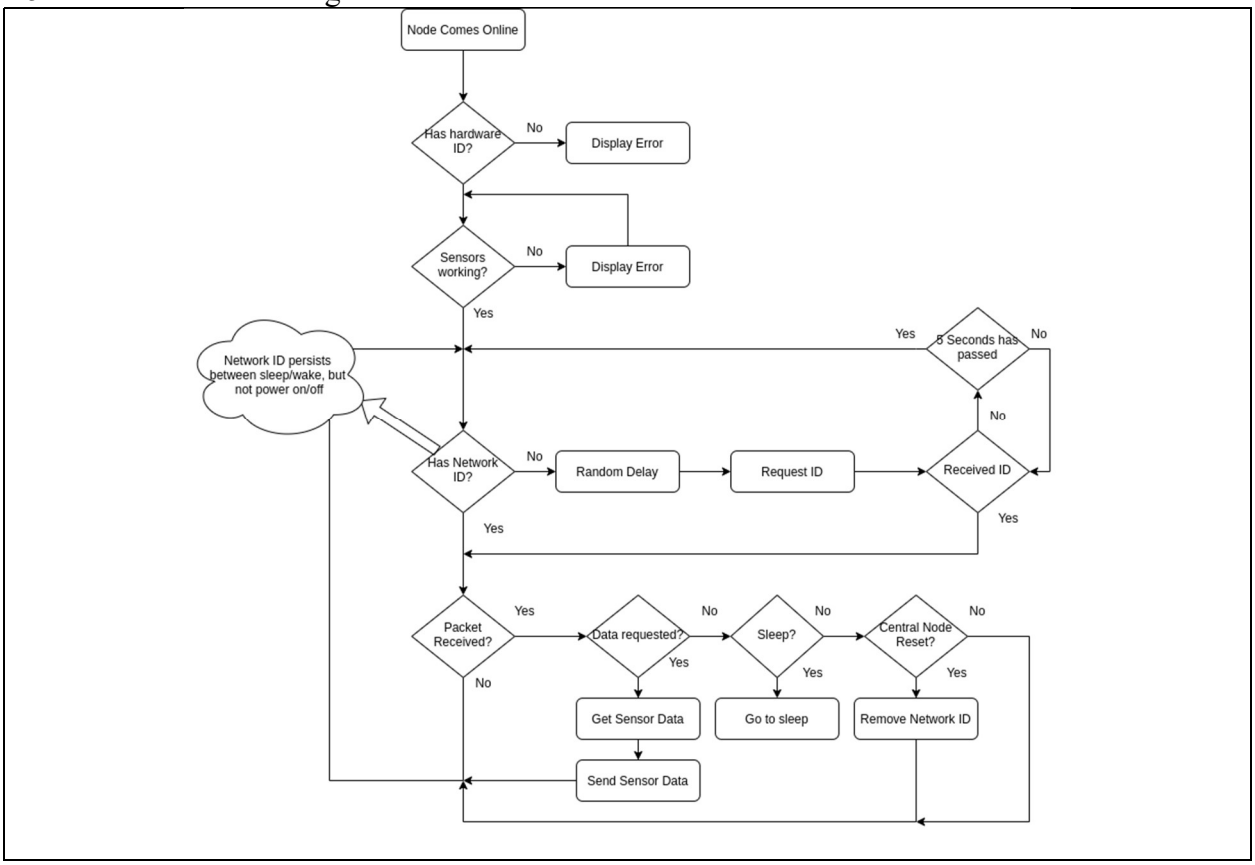
```
import serial
from datetime import datetime as dt

# Steve PC - Bus 01 - USB 005 and 007 --- /dev/bus/usb/001/005 ---- /dev/ttyUSB5"
# Change to proper USB port as needed
serialPort = serial.Serial(port="/dev/ttyUSB1", baudrate=115200, timeout=1, stopbits=serial.STOPBITS_ONE)

while(1):
```

```
    with open("recieved.csv","a") as file:
        serialInput = serialPort.readline().decode(errors='ignore')
        if serialInput:
            data = serialInput.strip()
            senderId, temp, pressure, altitude, humidity, capacitive = data.split(',')
            time = dt.now()
            file.write(f"{str(time)}, {senderId}, {temp}, {pressure}, {altitude}, {humidity}, {capacitive}\n")
```

A5 – Network Block Diagram



A6 – gps.ino

```
#include "TinyGPS++.h"
#include "HardwareSerial.h"
#include "heltec.h"

#define BAND 915E6  //you can set band here directly,e.g. 868E6,915E6

float latitude , longitude;
String  lat_str , lng_str, date_str , time_str;
int month, year;
TinyGPSPlus gps;
HardwareSerial SerialGPS(1);

#define RXD2 16
#define TXD2 17

void setup()
{
   Heltec.begin(true /*DisplayEnable Enable*/, true /*Heltec.Heltec.Heltec.LoRa Disable*/, true /*Serial Enable*/, true /*PABOOST Enable*/, BAND /*long
BAND*/);
   SerialGPS.begin(9600, SERIAL_8N1, RXD2, TXD2);
}
```

```
void loop()
{
  while (SerialGPS.available() > 0) {
    if (gps.encode(SerialGPS.read()))
    {
       if (gps.location.isValid())
       {
         latitude = gps.location.lat();
         lat_str = String(latitude , 6);
         longitude = gps.location.lng();
         lng_str = String(longitude , 6);
         Serial.print("Latitude = ");
         Serial.println(lat_str);
         Serial.print("Longitude = ");
         Serial.println(lng_str);
       }
       if(gps.date.isValid())
       {
         date_str = gps.date.day();
         month = gps.date.month();
         year = gps.date.year();
         Serial.print("Date");
         Serial.println(date_str);
         Serial.print("Month");
         Serial.println(month);
         Serial.print("Year");
         Serial.println(year);
       }
    delay(1000);
    Serial.println();
    }
  }
  Heltec.display->clear();
  Heltec.display->setTextAlignment(TEXT_ALIGN_LEFT);
  Heltec.display->setFont(ArialMT_Plain_10);
  Heltec.display->drawString(0, 0, "Latitude: " + lat_str);
  Heltec.display->drawString(0, 10, "Longitude: " + lng_str);
  Heltec.display->drawString(0, 20, "Date: " + date_str);
  Heltec.display->drawString(0, 30, "Month: " + String(month));
  Heltec.display->drawString(0, 40, "Year: " + String(year));
  Heltec.display->display();
}
```