Internet of Machines

DESIGN DOCUMENT

Team May 1715

Vermeer Corporation

Mr. Keith Bryant
Mr. Steven Lischer

Dr. Sang Kim

Dean VanEvery, James Kluesner,
Matt Gustin, Sam Ellis,
Cody Lample, Kojo Otchere-Badu

may1715@iastate.edu

http://may1715.sd.ece.iastate.edu

Revised: 23 April 2017

Contents

1 Introduction

1.1 Project statement

The “Internet of Machines” project aims to implement the “Internet of Things” philosophy in a modular, compact, affordable, and long-ranged design. This design must be able to collect diagnostic data from machine implements (targeting specifically Vermeer-produced farm machinery) and compile it into a centralized database for analysis. This exchange of data must be delivered in a timely and reliable manner.

1.2 Purpose

This project is intended to enhance the owner/operator’s ability to monitor the operation of their machinery’s systems. The target of the design is use in implements outside of the range of traditional WiFi and other wireless network architectures, primarily the agricultural implements of our sponsor. By creating a centralized collection of relevant data, the operator will have a better understanding of the conditions their machinery is working under, and subsequently be able to better judge the frequency and types of maintenance. This will increase the efficiency and longevity of devices which would implement this design.

In practice, this design would be able to deliver data types outside of operational diagnostic data, and thus our implementation is made to be expandable to support this consideration. Due to the highly developmental state of the “Internet of Things” philosophy and tools our design attributes much to, we have chosen to modularize our hardware choices as much as possible.

1.3 Goals

The goals we sought to meet in the final implementation of the project are as follows:

- Create a reproducible, documented, and affordable solution
- Use Low-power, Long-range (LoRa) hardware to transmit data over long distances
- Design a sustainable set of hardware components, requiring minimal maintenance
- Deliver proper documentation on the implementation of the design in practice

2 Design

2.1 System Standards & Specifications

The system can be separated into three key components, as follows:

- Transmitter/Node module
- Receiver/Gateway module
- Database

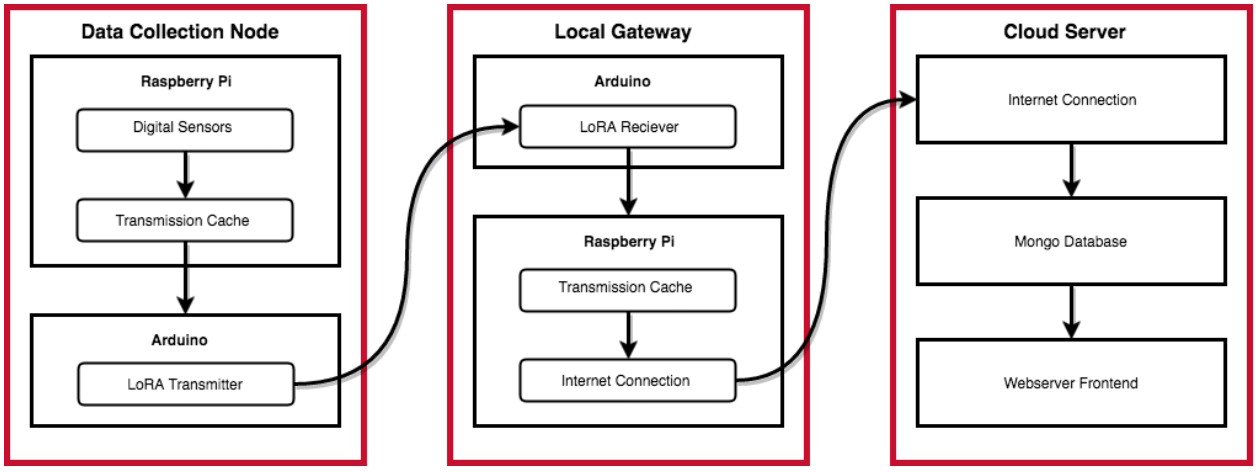
The **Transmitter/Node** module describes the hardware/software combination that is used in monitoring and subsequently transmitting data from the target machinery. It consists of two microcontrollers; One microcontroller is responsible for gathering data from an arbitrary number of sensors and packetizing the data set. The second is responsible for establishing the connection with its associated gateway module, and parsing the data packet via the LoRa hardware. These implement the arduino and Raspberry Pi 2

microcontrollers, respectively. The transmitting hardware for the node module is the Liberium LoRa SX1272 transmitter.

The **Receiver/Gateway** module describes the portion of the system responsible for compiling data from an arbitrary number of node modules, and subsequently posting a corresponding JSON request to the database. Like the node module, the hardware for the gateway also consists of two microcontrollers. One which is meant to receive packetized node data, and one meant to form the received data into a request usable by the database. These microcontrollers are also the arduino and Raspberry Pi 2 microcontrollers, respectively. Like the node, the receiving hardware for the gateway module is the Liberium LoRa SX1272 transmitter.

Finally, the **Database** describes the server on which the data received by an arbitrary number of gateway modules is stored. Our implementation uses the MongoDB database software in storing the node data, as it is a schema-less database and it is expected that a gateway may be pulling from nodes with vastly different sensor types.

A full block diagram of the design is as follows:



2.1.1 Non-functional Requirements

- The client system must have reasonable portability, such that it could be used to measure data from a variety of different devices, while not impeding regular operation of said devices.
- The client must be expandable, such that it can support an arbitrary number of sensors/retrieve an arbitrary amount of sensor data
- The client system must be able to send relevant data to the server in a timely and stable rate.
- The server and client must be configurable to associate the client with the server.

2.1.2 Functional Requirements

- The system on the client machine must be able to retrieve relevant data from a set of sensors, including sensors both internal and external relative to the raspberry pi.
  This includes, but is not limited to:
  - Uptime (how long the system has been running since the previous power on).
  - Lifetime (how long since the system was initialized).

    - Temperature, Pressure, and Altitude
    - Other unique, system-specific data based on the machine that is being monitored.
- The client system must be able to connect to the server system over a LoRa connection at a range of several miles (1-5 miles).
- The server system must be able to serve a multitude of clients simultaneously.
- The server must be able to distinguish clients, and compile data from each independent client in a format which can be compiled and exported into a Microsoft Excel spreadsheet format (EG .xls, .csv).

2.2 Hardware Specifications

The hardware for both the node and gateway modules consists of two main components:

- Transmitting microcontroller
- Data Manipulation microcontroller

This distinction was borne from a hardware compatibility consideration in which the transmitting hardware interfaced only with the GPIO headers of the Arduino microcontroller. Due to our design still requiring a microcontroller able to implement a Bash shell, which would not be possible in the Arduino hardware, we separated these two responsibilities in the final design.

The **Transmitting** microcontroller is responsible, as stated, with interfacing with the transmitting hardware and association with its corresponding gateway or node module(s). It consists of a single Arduino microcontroller interfaced with a Liberium LoRa SX1272 via the Liberium Arduino shield. Due to the requirement of the physical build of the module being stable, we chose to use the shield to interface with the microcontroller as opposed to using a breadboard and/or soldering the SX1272 module to the GPIO headers.

The **Data Manipulation** microcontroller is responsible for packetizing/de-packetizing data in the node and gateway modules respectively, and transmitting the data to its gateway/pushing data to the database in the node and gateway modules respectively. It consists of a single Raspberry Pi 2 microcontroller, and is connected to the transmitting microcontroller via serial connection. On the node side, it has the additional responsibility of managing sensors, and regularly polling connected sensors of updates. On the Gateway side, this module has the additional responsibility with contacting the database and forming the JSON necessary to add collected data to the central server over some WAN connection, including WiFi and ethernet.

2.3 Software Specifications

The individual components of the software making up the design are as follows:

- Node Software
- Gateway Software
- Database Software

The **Node software** is responsible for the steps of reading data from the associated sensors, packetizing the poll result, and transmitting the packet over LoRa to the associated gateway. All steps of this process are implemented using the C language, due to the large number of operations on the serial ports to accomplish this step of the design.

The **Gateway software** is responsible for reading packets in from the LoRa connection, parsing its data into a JSON object defined by the server API, and finally sending the JSON to the server. This is done using a combination of C code, and python scripting for reading in data and packetizing it, respectively. This choice was made largely due to constraints in working with the Arduino IDE combined with the ability to use bash scripting on the linux-based Raspberry Pi operating system.

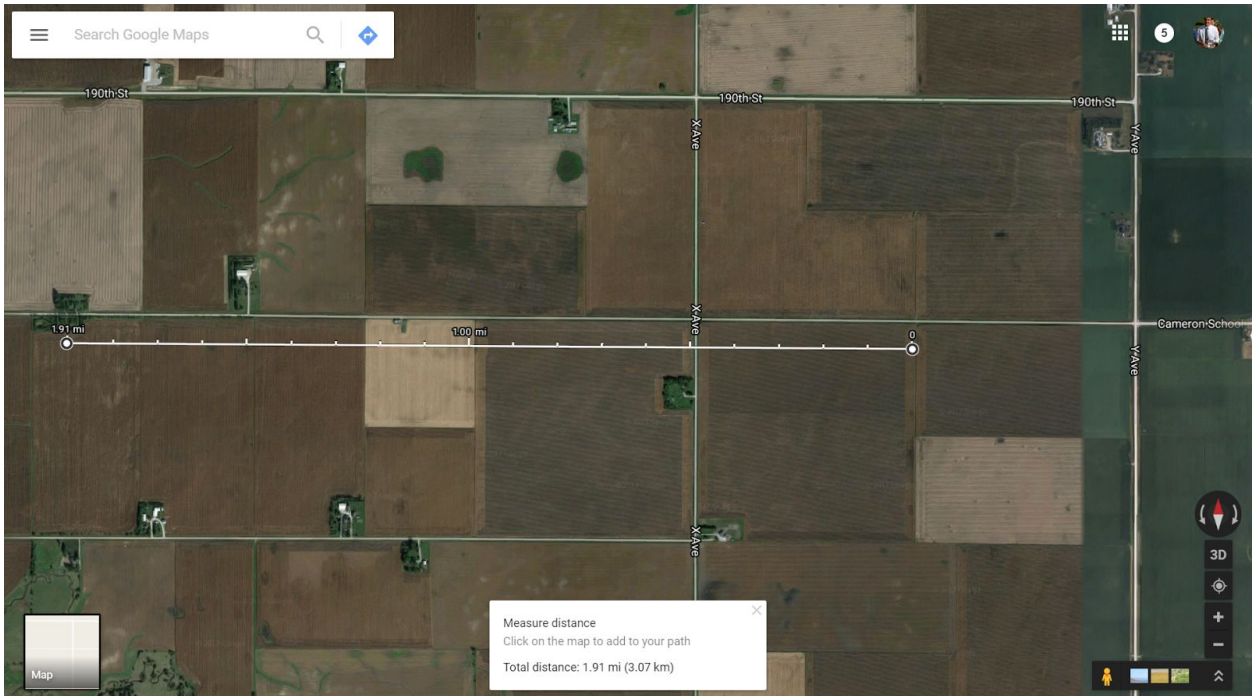
Finally, the **Database software** is the suite of components used to read in, host, and display data from the associated gateway components. Data is parsed into the database with http requests, the database itself being an instance of the MongoDB database suite.

3 Testing

For this project, the usefulness of the product is ultimately dependant on how reliably and how far data can be transmitted. We decided to test the transmission reliability and range under two circumstances, line of sight (LOS) and non-line of sight (NLOS). Since we expect the primary usage of this device to be in more open fields, we focused more on LOS testing, therefore we have more established methods and results for LOS tests.

3.1 Testing Methods

Line-of-Sight (LOS): We took our device to Cameron School Road in north east Ames, IA, to test. While here, we had the receiver in one car and the transmitter in the other. We parked the receiver and drove the transmitter incrementally farther away. We tested on three different modes 1 (BW 125, CR 4/5, SF 12), 5(BW 250, CR 4/5, SF 10) and 10(BW 500, CR 4/5, SF 7), generally estimated to give us the highest, mode 1, and lowest, mode 10, transmission distance, and one in between, mode 5. We tested at 0.1, 0.2, 0.3, 0.4, 0.7, 1.2, and 1.75 miles. We also tested at 1 mile for mode one, but on the road that we were on there was a dip and we lost line of sight, so we did not test there for the other modes. The map below shows where we tested on Cameron School Road near X ave. We held the antennae out the windows to preserve LOS.



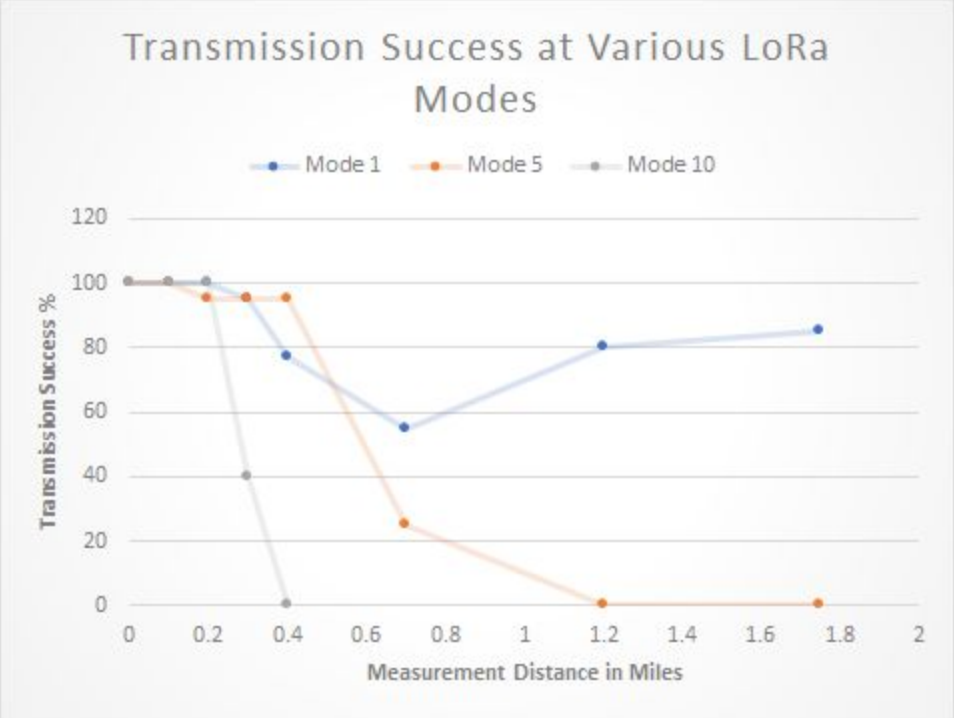
Non-Line-of-Sight (NLOS): We kept the receiver in Coover room 1301 and carried the transmitter through Coover and towards design (through the hallway to the east). We simply carried this until we lost reliable signal.

3.2 Testing Results

LOS: Our quantitative raw data is below, and a graph with all data presented dropping the NLOS data at one mile. As a note we found that on some occasions the signal would become more reliable after a few seconds had passed after extending the antenna out the window. This was particularly the case at further distances. For mode one, we can't be sure at this time how far the range could be, since our road went over a hill breaking line of sight. After the signal had 'settled' for a few seconds, we were getting every transmission as a successful reception.

Mode (M)	**0**	**0.1**	**0.2**	**0.3**	**0.4**	**0.7**	**1**	**1.2**	**1.75**
Mode 1	100	100	100	95	77	55	13.33	80	85
Mode 5	100	100	95	95	95	25	0	0	0
Mode 10	100	100	100	40	0	0	0	0	0

This table shows the distance measured across the top, and the mode down the left. The measurements taken are percentage of successful transmissions over our testing period.



This graph represents the same data above.

NLOS: We lost signal on mode 1 (long range) near the front door of Design.

4 Conclusion

The outcome of our project is a stepping stone to future work into integrating Vermeer machines into a wide network which implements internet of things principles. At the end of our two semesters we have delivered a proof of concept system for using LoRa technologies in the transmission and reception of sensor data. Taking the idea from concept to fruition has been a rewarding and frustrating experience. Overcoming shipping delays and hardware inconsistencies, our goals changed as time progressed but the overarching goal of providing the proof of concept was met. Moving forward, the project can expand opening up testing into multiple transmitters/receivers working together instead of just our one. Mounting the hardware into a more robust package is another future goal.

5 Appendices

5.1 Operation Manual

The following section includes a manual for both operating and constructing the system. Since this project is primarily conceptual, verbosity in instructions allows future developers to build upon our work.

5.1.1 Node Module Setup

Basic Sensor Connections

Perform basic connection and setup of sensors included in the SunFounder Sensor Kit V2.0 for Raspberry Pi.

DS18B20 Temperature Sensor

Required Components:

1 Raspberry Pi
1 Breadboard
1 GPIO Extension Board
Jumper wires
DS18B20 Sensor module

Hardware Setup:

Step 1: Connect VCC pin on DS18B20 module to 5VDC rail
Step 2: Connect GND pin to ground rail
Step 3: Connect SIG pin on DS18B20 to GPIO7 I/O pin on Raspberry Pi

Software Setup:

Step 1: Upgrade kernel of raspberry pi
Step 2: Download wiringPi API
Step 2: Enable GPIO interface
Step 3: Reboot raspberry pi
Step 4: Mount device drivers
Step 5: Modify the address of device in C code to match the memory address of the mounted device.
Step 6: Compile Code

BMP180 Barometer

Required Components:

1 Raspberry Pi

1 Breadboard
1 GPIO Extension Board
Jumper wires
BMP180 Barometer module

Hardware Setup:

Step 1: Connect VCC pin on BMP180 module to 5VDC rail
Step 2: Connect GND pin to ground rail
Step 3: Connect SDA pin on BMP180 to SDA I/O pin on Raspberry Pi
Step 4: Connect SCL pin on BMP180 to SCL I/O pin on Raspberry Pi

Software Setup:

Step 1: Download wiringPi API
Step 2: Enable GPIO interface
Step 3: Reboot raspberry pi
Step 4: Mount device drivers
Step 5: Modify the address of device in C code to match the memory address of the mounted device.
Step 6: Compile Code

Automated Sensing Program

This software is configured to run on boot at each node using a cron tab. This program maintains the data sample rate, samples the sensors for data, formats and locally caches collected data, and transmits packets to the LoRA module.

Sampling Rate

Since desired sampling rates can vary from seconds to days, the machinery, and therefore sensing hardware, may be turned on and off several times within one sampling period. To maintain a maximum sample rate regardless of power cycles, the program saves a text file of the last transmission time in unix time stamp form. This file is saved in the same directory of the program as "time.txt". Then the program waits in an infinite while loop until the difference between the current time and last transmission time is greater or equal to the desired sample rate. System time is not affected by power cycling due to an auxiliary timekeeping battery. The sample rate can be altered by modifying the corresponding value in the configuration file, "config.txt". Note that the automated sensing program must be restarted for these changes to take effect.

Sampling Sensors

The basis for the functions written to sample the temperature and pressure sensors is in the documentation provided by SunFounder. Altitude is computed using the output of the pressure sensor and linear equation. The altitude computation is used to simulate an additional sensor connected to the node. In further revisions, computation of altitude can and should be moved to the server side.

Formatting and Locally Caching Data

Because the the data rate of the LoRA standard is relatively low, especially over long distances, a single transmission was limited to 36 characters ,or 36 Bytes of data. The transmission is formatted with five comma separated values of varying lengths: node ID (3), transmission time stamp (10), temperature data in Celsius (5), pressure data in Pascals (6), and altitude data in meters (8). After

each sensor is sampled, the collected data is rounded off to fit the character requirements. Then the 36 character transmission is saved as a text file, “packet.txt” in the same directory as the program.

Transmitting

Since we are implementing an Arduino to interface with the LoRa transmission shield, the data contained in “packet.txt” needs to then become a char array in the Arduino IDE. However, since the Arduino IDE cannot access data files from the Raspberry Pi, we send the data into the IDE using a Python script. This script reads the characters in “packet.txt” and sends it via serial communication into the IDE for transmission. The python script runs automatically every time division set by the user, using the crontab settings in the Raspberry Pi command line. For our testing purposes, we ran the script to read in the new data every one minute and then transmit.

Compiling

Since the BMP180 sensor requires several C functions written by SunFounder to operate, math.h functions are used, and WiringPi memory mapped I/O is utilized, several arguments must be appended to the GCC compiler call when compiling the automated sensing program. The command should resemble the following:

```
gcc sample.c bmp180.c -o sample -lwiringPi -lm
```

Where bmp180.c contains the functions written by SunFounder to operate the barometer, lwiringPi includes the WiringPi API, and lm includes the math library.

5.1.2 Gateway Module Setup

For first time use for both the receivers and transmitters:

Install Arduino

sudo apt-get install arduino

Installing LoRa

Download the Libraries

http://www.cooking-hacks.com/media/cooking/images/documentation/tutorial_SX1272/SX1272_library_arduino_v1.4.zip

Libraries are often distributed as a ZIP file or folder. The name of the folder is the name of the library. Inside the folder will be the .cpp files, .h files and a examples folder.

To install the library, first quit the Arduino application. Then uncompress the ZIP file containing the library. For installing libraries, uncompress zip file. It should contain a folder called SX1272. Drag this folder into your libraries folder. Under Windows, it will likely be called "My Documents\Arduino\libraries". For Mac users, it will likely be called "Documents/Arduino/libraries". On Linux, it will be the "libraries" folder in your sketchbook.

The library won't work if you put the .cpp and .h files directly into the libraries folder or if they're tested in an extra folder. Restart the Arduino application. Make sure the new library appears in the Sketch->Import Library menu item of the software.

Transmitter:
[If first time] download the iom-node repository and move the Transmission folder to home/pi/
Turn on the Arduino Transmitter

1) Open the Arduino IDE
2) File>Open, navigate to and open: home/pi/Transmission/Transmission.pde
3) Ensure you are using the correct board and serial port
    a) Tools>Board (best with uno)
    b) Tools>Serial Port (usually the top option)
4) Upload the code to the Arduino
    a) Click the right facing arrow below the Edit menu
    b) Or use command ctrl-u

Run the sensor data collector code

1) Open the Terminal
2) Enter the command: cd ~/Documents/Production/ && ./run

Receiver:
[If first time] download the iom-node repository and move the RxWack folder to home/pi/
Turn on the Arduino Transmitter

1) Open the Arduino IDE
2) File>Open, navigate to and open: home/pi/RxWack/RxWack.ino
3) Ensure you are using the correct board and serial port
    a) Tools>Board (best with uno)
    b) Tools>Serial Port (usually the top option)
4) Upload the code to the Arduino
    a) Click the right facing arrow below the Edit menu
    b) Or use command ctrl-u

Set up transmission to database

1) Download the iom-gateway package to the user's home folder
2) Modify "Push_To_Database.py" to use a unique gateway ID
3) Add the following lines to the user's crontab:
    a) */30 * * * * python ~/data2txt.py
    b) */30 * * * * python ~/iom-gateway/run.sh ~/Output.txt

5.1.3 Database Setup

Preconditions:

- Access to hosting on an arbitrary domain accessible via WAN, which will be referred to as "database.this"
- Download of the "iom-gateway" package
- Download of the "internet-of-machines" package

1. **Installation of MongoDB on machine hosted under database.this**
   Full documentation on the installation of the database software on specific operating systems may be found on the official MongoDB website here.
   It is highly recommended that after installation, an administrative user is added to the "admin" collection and credential-less login be turned off.

2. **Creation of the initial collection**
   Schema may be found in the schema.json file located in the iom-gateway package.
   The new collection must have the following fields defined:
   - Name
   - Name for each gateway within the “gateways” array
   - The unique ID for each gateway
   - Name for each node within each gateway within the “nodes” array
   - Sensor names under the “sensors” array of each node
3. **Web Client installation**
   The web client is consists of static HTML, CSS, JS assets and is hosted by surge.sh. Updates can be made to the code in the ‘internet-of-machines’ package and then deployed by running `npm run deploy` and specifying the public folder as the target folder. The client then requests and updates data through the API via Ajax requests to the example domain.
   In addition, the data sets captured by the client will be viewable via http in your web browser of choice.

5.2 Alternate Designs

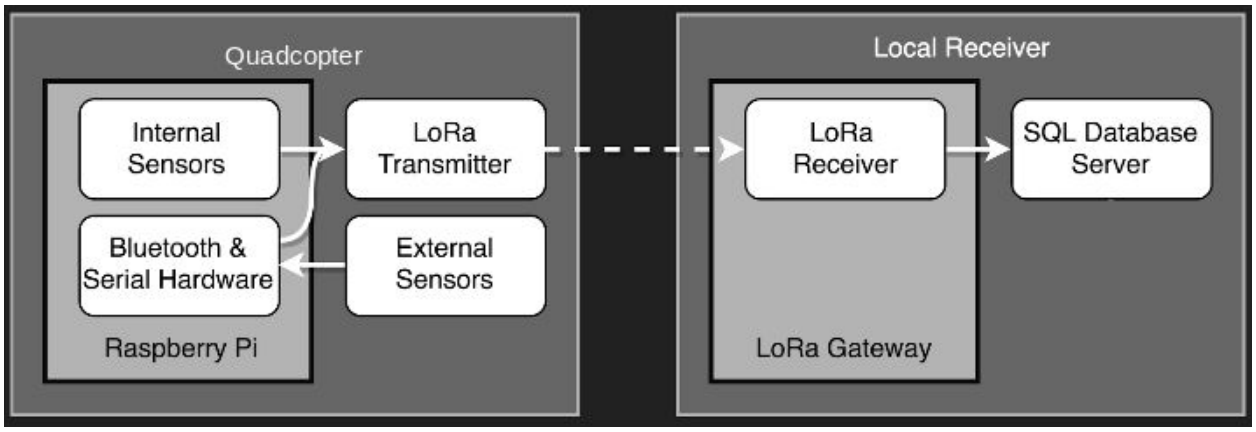
Over the course of the design and building phases of the project, we looked into several alternate designs we believed would meet the requirements of the project. However, for various reasons, each failed in certain practical aspects and eventually led us to our final design. The following is a comprehensive list of some of these unused designs:

1. **“Cellular Communication” variant**
   In the process of looking at different transmission options, we considered several different pieces of hardware to use in the communication between nodes and gateways. One that was the most compelling to consider was the 4G cellular communication module produced by Liberium.
   Using the same microcontroller hardware, the only difference between this design and our final one would the the replacement of the LoRa shield on the Arduino Uno microcontroller with the 4G one. However, several concerns with the cost of the hardware turned us away from using it in our final design. One big detriment in its use compared to the open-source band of the LoRa hardware is the requirement to license use through a cellular carrier, a cost which would be of considerable size if the project were to be implemented in large scale.

2. **“Raspberry Pi Only” variant**
   The final proposed design. In this variant, instead of the separation of responsibilities of data manipulation and data transfer between the Raspberry Pi and Arduino Uno microcontrollers, respectively, this design would have the Raspberry Pi be responsible for both. However, compatibility issues during the building phase made this not possible with the hardware at hand. Unfortunately, the LoRa shield’s pin set was one compatible only with the Arduino GPIO headers. This prompted a move to separate the data transfer from the rest of the required operations, and the creation of the two-microcontroller design used in the final build.
   In future development, it would be possible to use this design, and could also be an improvement over the final build. The primary consideration that would need to be taken for this would be the use of a LoRa shield compatible with the GPIO header of the Raspberry Pi module, instead of the Arduino Uno. This would require a modification of the python scripts reading from the Raspberry

Pi serial port connected to the transmitting Arduino to instead read from the GPIO header, as opposed to one of the serial ports.

The following is a block diagram of this original design:



5.3 Miscellaneous Considerations

The following are considerations and ideas that may be important in production of this design:

- In our design, we used an independent 5V USB power source for the Raspberry Pi for the "node" module, this may not be practical in use with machinery that already has its own, independent power source. As such, we recommend integration of the node module into its host machine's power source if possible.
- As LoRa is a rapidly advancing technology, it is likely that improved hardware will be developed frequently over the coming years. This may entail equally frequent research into these changes, but may also correspond with increased responsiveness and range should it be added in future design variants.
- As noted in Alternate Design 2, the primary reason for the use of the Arduino microcontroller is due to time constraints of the project. It is still preferable and highly recommended that future research into this design use a LoRa shield compatible with the Raspberry Pi hardware in order to mitigate this additional component. Not only would this cut down on the size of the module, allowing it to be integrated more discretely, but would also drastically lower power consumption.
- It is recommended that if using a Unix-based system in production that initialization of scripts on both the transmitting and receiving modules be added to the crontab to run on boot, in order to avoid the need to start all the scripts manually on each reboot.