

Evaluation of the Industry 4.0 Application for Better Farming

Written by James Howard

Master in Engineering

15921801

Auckland University of Technology

Auckland

2021

Acknowledgements

A big thank you to Sarat Singamneni for giving me the opportunity to take on this project and helping me to receive a scholarship to support me throughout the duration of it. He also shared his knowledge and offered guidance, keeping me in the right direction through the project. Thanks to the New Zealand Product Accelerator of the University of Auckland for funding this project and granting me the scholarship making this project possible.

A special thanks to Justin Matulich who helped me to resolve various technical problems regarding devices used which I ran into regarding my experimental work.

Thank you to Nick van der Geest and Misha Versteeg for sharing their knowledge to get set up with Raspberry Pi and Arduino devices used extensively through the project.

Abstract

Industry 4.0 is an emerging concept which involves optimizing production processes and increasing efficiency by taking advantage of wireless sensor networks in collaboration with cloud-based servers for monitoring systems. It focuses on the implementing Cyber Physical Systems to current production equipment and integrate them with ICT-based networks to provide resource and cost advantages compared to those same industries without these technologies. Being a relatively new way of optimizing production, there is a lot of space for improvement and development when implementing these concepts. Quite often, there are existing solutions which do apply I4.0, however they usually come with quite steep investment costs, and are difficult to implement into workplaces, dissuading those customers who could benefit from them. A large industry in New Zealand which has the potential to benefit from these technologies is farming and agriculture. Farmers need to monitor large farms, which comes at the cost of a lot of time. A system has been developed which sensors can be added to, allowing for farmers to monitor their farms from mobile devices, hence optimizing their workload. To test this system, an analog capacitive moisture sensor was added to the sensor nodes which measures the moisture content in soil throughout a farm and transmits it back to a base station (coordinator) which processes data and uploads it to a cloud server, allowing for remote farm monitoring. Using the wireless sensor network structure designed through this project, many sensor types can be added to these sensor nodes and provide the relevant data as required. These devices have been designed to

transmit wireless data across long distances, as found in farms and have a lifespan of 5+ years, requiring extremely low upkeep. The solution has been made using low-cost equipment, meaning they are appealing to farmers in comparison to previous solutions. Due to these benefits, farmers would find these systems attractive and more likely to be applied.

Statement of Originality

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another individual (except where explicitly defined in the acknowledgements or applied references in literature), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institute of higher learning.

Name

James Lee Michael Howard

Signature

Date

31/08/2022

Table of Contents

Acknowledgements.....	2
Abstract.....	3
Statement of Originality.....	5
Table of Contents.....	6
Nomenclature	11
List of Figures	14
List of Tables	17
Chapter 1 Introduction	18
1.1 Aim	20
1.2 Scope	20
Chapter 2 Literature Review	22
2.1 Industry 4.0	22
2.2 Cyber Physical Systems	24
2.2.1 CPS Overview	24
2.2.2 Usage of CPS in Industries	25

2.3 Cloud Computing.....	27
2.4 Wireless Communications.....	29
2.5 Sensor Technologies.....	31
2.6 Internet of Things	32
2.7 Smart Industries	34
2.7.1 Smart Manufacturing	34
2.7.2 Smart Grids	36
2.7.3 Smart Farming	37
2.8 Research Gaps, Questions and Objectives of the Current Work	39
Chapter 3 Concept, Considerations and Specifications.....	41
3.1 Chosen Concept	42
3.2 Design Considerations.....	43
3.3 Design Specifications.....	44
Chapter 4 Design of WSN and Structure for Moisture Monitoring System	46
4.1 Design Schematic	46
4.2 Wireless Communications.....	48
4.3 XBee Programming.....	50
4.4 Arduino	53

4.5 Raspberry Pi.....	58
4.5.1 RPi Schematic	60
4.5.2 Python Programming.....	62
4.5.3 Cloud Servers	64
4.5.4 Remote Devices	67
4.5.5 XBee setup with XCTU	70
4.5.6 Reading Analog Data	74
4.5.7 Data Processing	78
4.6 Sensor Node Battery Life.....	82
Chapter 5 Design Validation, Results and Discussion	85
5.1 Communications Validation	85
5.1.1 Analog Voltage Correction.....	87
5.2 Data Interpretation	90
5.3 Cost Analysis.....	92
5.3.1 Wireless Sensor Node Cost.....	92
5.3.2 Coordinator Cost.....	94
5.4 Final Design	95
5.4.1 Data Readings	95

5.4.2 Wireless Sensor Node Final Schematic.....	97
5.4.3 Moisture Readings.....	98
5.4.4 Range Testing	101
Chapter 6 Conclusion	104
6.1 Objectives and Achievements	104
6.2 Critical Results and Inferences	105
6.3 Future Work	107
References	111
Appendices.....	118
Appendix A – XBee S2C Datasheet.....	118
Appendix B – Arduino Datasheet	120
Appendix C – Raspberry Pi 3B Datasheet.....	121
Appendix D – Capacitive Sensor Datasheet	122
Appendix E – Python Code for Coordinator	123
XBee Read Test 1	123
XBee Read Test 2	124
XBee Read Test 3	124
XBee Read Test 4	125

XBee with Voltage level adjustment 126

XBee Final Script 127

Appendix F – XBee Profiles..... 129

Coordinator Profile 129

Sensor Node Profile 131

Appendix G – Part List 135

Nomenclature

CPS	Cyber Physical System
IoT	Internet of Things
WSN	Wireless Sensor Network
DT	Digital Twin
JIT	Just in Time
RPi	Raspberry Pi
I4.0	Industry 4.0
AI	Artificial Intelligence
6LoWPAN	IPv6 over Low-Power Personal Area Networks
IPv6	Internet Protocol version 6
LoRa	Long Range
API	Application Programming Interface
IDE	Integrated Development Environment
GPIO	General Purpose input/output
RF	Radio Frequency
LAN	Local Area Network
OS	Operating System

ID	Network Identifier
NI	Node Identifier
SH	Serial Number High
SL	Serial Number Low
DH	Destination Serial Number High
DL	Destination Serial Number Low
NI	Node Identifier
SM	Sleep Mode
SP	Sleep Period, ms
BD	Baud Rate, bps
IR	Input/Output Sampling Rate, ms
ADC	Analog-Digital Converter
VREF	Reference Voltage, V
V_{in}	Voltage in, V
V_{out}	Voltage out, V
TX	Transmit Data
RX	Receive Data
GND	Ground
HEX	Hexadecimal Numbering
NOOBS	New Out Of Box Software
Pip	Package Installer for Python

DBX	Dropbox
CAD	Computer Aided Design
FDM	Fused Deposition Modeling
CNC	Computer Numerical Control

List of Figures

Figure 4.1 Raspberry Pi	49
Figure 4.2 XBee Module and Pin Configuration (Electronic Wings, 2018)	51
Figure 4.3 Networking & Security parameters of XBee	51
Figure 4.4 I/O Analog sampling configuration	52
Figure 4.5 I/O Sampling Rate	52
Figure 4.6 Arduino	53
Figure 4.7 Arduino Connector Pins	54
Figure 4.8 Arduino with XBee Shield	57
Figure 4.9 Raspberry Pi GPIO Pinout	59
Figure 4.10 RPi Configuration (Jain, 2019).....	60
Figure 4.11 Raspberry Pi connected to XBee through GPIO pins	61
Figure 4.12 Raspberry Pi Connected to XBee Explorer USB	62
Figure 4.13 Setting the workspace in the RPi Terminal.....	63
Figure 4.14 Cloud server test upload text file.....	65
Figure 4.15 Dropbox test upload text file, second script.....	66
Figure 4.16 Result Test text file on dropbox server	67
Figure 4.17 Initial Sensor Schematic	68
Figure 4.18 Wireless sensor node with voltage divider applied	69
Figure 4.19 XBee Default parameters.....	70

Figure 4.20 Setting modules to XB24C DigiMesh 2.4 TH 9002 Firmware.....	71
Figure 4.21 Frame Log Terminal	73
Figure 4.22 Reading serial data through USB port	75
Figure 4.23 Query Remote XBees for data	76
Figure 4.24 Analog Result from test script	78
Figure 4.25 Wireless Sensor Node in Glass of water	79
Figure 4.26 Moisture sensor in atmospheric air	80
Figure 4.27 Python definitions to process moisture readings.....	80
Figure 4.28 Sleep Mode Settings	83
Figure 4.29 Wireless Sensor Node with Sleep Mode enabled.....	84
Figure 5.1 Capacitive Moisture sensor in Glass of water	86
Figure 5.2 Wireless Sensor Moisture Readings - Raw data	86
Figure 5.3 AA Battery Voltage across its life	87
Figure 5.4 Callback method when data received with voltage level reading.....	88
Figure 5.5 Data samples adjusted to account for low battery voltage	89
Figure 5.6 Moisture Sensor in air (left) and glass of water (right).....	90
Figure 5.7 Wireless Moisture sensor used on recently watered plant.....	91
Figure 5.8 Moisture readings for recently watered plant	92
Figure 5.9 Information from remote XBee	95
Figure 5.10 Uploaded text file onto dropbox server	96
Figure 5.11 Wireless Sensor Node Schematic	97

Figure 5.12 Indoors XBee Range (Google, 2022)	101
Figure 5.13 Line of Sight XBee Range (Google, 2022).....	102

List of Tables

Table 2.1 Wireless Communication Descriptions 30

Table 4.1 Table of Wireless Communication Protocols for IoT 47

Table 4.2 Example Frame Received from Remote XBee Device 55

Table 5.1 Wireless Sensor Node Cost Analysis 93

Table 5.2 Coordinator Cost Analysis 94

Table 5.3 Verification of system applied to atmospheric air 98

Table 5.4 Verification of system applied to glass of water 99

Table 5.5 Verification of system applied to freshly watered plant 99

Chapter 1

Introduction

Industry 4.0 is a term which has been thrown around production industries and is the concept of optimization in the appropriate field, through the implementation of emerging technologies and concepts, including Cyber Physical Systems, Internet of Things, Wireless Sensor Networks and Cloud-computing. When these are used in unison, they can create ways to increase production rates, lower costs or find flaws in setups. The goal is to centralize data and observations to find improvements. I4.0 is essentially the introduction of these technologies to the existing I3.0, which includes the implementation of computers and automation, and enhancing it to create smart systems which are enabled by the gathered data. The project is to be broken down into 4 key sections, which each will allow a final design to be found, firstly, Literature Review.

Literature Review will be conducted on these technologies, the fields they are applied to and how much they can optimize production. Through this literature review, a field is to be identified which can be focused on, and find the problems with current solutions, identifying why they are not widely used throughout the field, and make a solution to this problem, promoting them to the user and giving very little reason of why not to use them. With information found throughout literature review, the benefits and problems can be identified in existing solutions which can then

be used as cornerstones for this project to overcome and design a system which addresses these issues.

Once the Literature Review has been completed, it should become quite clear which direction to steer the project in, and what sort of design is required. A concept can be created based on information found throughout the literature review targeting the specific field in which the project will focus. With the chosen concept, things to be considered throughout the design process will be established, followed by the specifications of the design, which will target getting the final product to meet the overarching goal of this project and will fill the gap found in the literature review.

Thirdly, the design process will begin which will take the chosen concept to a physical system which can be employed into the field. This includes trying to find the parts which work well in existing systems, while ignoring the parts which dissuade the user from implementing them into their work environment. Schematics, processors, and wireless communication protocols will be examined and implemented into the design, then used together with wireless sensor networks to create devices which can be easily implemented into the design which allows it to meet the specifications.

Finally, with a completed design, the design is to be validated to check that it meets the requirements and specifications found earlier in the project. All the components of the system are to be analyzed to ensure correct functionality and meeting criteria including communications, data readings, cost analysis, and final system testing.

1.1 Aim

The aim of this project is to research I4.0 technologies, fields in New Zealand in which they can be applied and design a solution to the chosen field. Currently, I4.0 solutions to problems are often difficult to install and setup, are often quite expensive and not easily accessible or well-known, meaning production workers don't seek to implement such systems. It is because of this that a key aim throughout the project is to develop a system which provides more incentive to use in practice as opposed to existing solutions.

1.2 Scope

The scope of this project is:

- Perform research on Industry 4.0, the technologies making it possible, and how they can be applied to optimize production.

- Gather information on the industries in which they have potential, and focus on one, conducting literature review on it and generate ideas on how it could possibly be optimized.
- Conceptualize a system which applies I4.0 technology which has potential to be used in the production environment.
- Design a production monitoring system which implements these technologies, meeting the requirements which are to be established in Chapter 3.
- Create a full structure which can be easily adapted to situations as required, and scalable to allow for different scenarios and different magnitude production, so the design is applicable to various situations.
- Make the design easily accessible and cheap which encourages the production worker to apply the system to their work environment.

This thesis will summarize the approach taken to literature review regarding the concepts involved in I4.0, and where they can be applied, design a product which can be used in practical scenarios, and validate the system.

Chapter 2

Literature Review

The following section will discuss research undergone before, and throughout the course of the project. The aim of this is to determine what sort of technologies make Industry 4.0 possible, what impact they have on optimizing production, saving resources, and how these techniques are implemented. Different industries in which these concepts can be applied to will be researched, finding one suitable to conduct this project on.

2.1 Industry 4.0

I4.0 is the concept of a fourth industrial revolution in the production and manufacturing sector. It can be applied through the implementation of Cyber Physical Systems (CPS) to pre-existing production equipment and end-to-end integration of digital systems by seeking fully integrated solutions (Xu, Xu, & Li, 2018). The main components of Industry 4.0 include CPS, Internet of Things (IoT), Sensors, Wireless communication, and Cloud services. I4.0 combines these technologies to make advancements in terms of production rates, time efficiency and saving companies more money. Wireless sensor networks (WSN) are used in their appropriate fields,

with wireless data sent to a network, which can then uploaded to a cloud service, and be processed into usable data for decision-making, automation and saving time.

The focus is to shift away from embedded systems and transition to CPS, IoT and cloud-based computing. This allows machines in the manufacturing sector to communicate with one another and decentralize control systems which can improve and optimize production. Recent developments in IoT have made I4.0 possible. IoT being developed in conjunction with existing technology such as sensors, actuators, or other smart devices, make for intelligent equipment which can optimize manufacturing and production in real-time. Cloud-computing offers high performance, low-cost resource sharing and dynamic allocation, which is used with wireless communication, decentralizing computational tools, and sharing services (Xu, Xu, & Li, 2018).

In one application, an automotive manufacturing company applied these technologies to optimize the configuration of production lines, balancing the load between different workstations. This enabled them to use labour efficiently with production rates also increasing. Applying these technologies enabled the manufacturer to reduce operating costs and capital investments by about 10%. Another case in automotive manufacturing also followed this 10% reduction in costs by using planning tools to optimize plant capacity, allowing them to bring new car models into production (Xu, Xu, & Li, 2018). To enable I4.0 to be openly available and popular, technologies including both hardware and software need to be developed into standard solutions, rather than privately owned and proprietary (Muhuri, Shukla, & Abraham, 2019).

2.2 Cyber Physical Systems

2.2.1 CPS Overview

The main key technology to unlocking I4.0 are CPS, which enable the connections between the cyber world and the physical world, allowing for physical interaction. CPS are applied in real world concepts, including robotics, autonomous vehicles, medical monitoring, pilot avionics and smart manufacturing.

With CPS, systems in the physical world can exchange data with each other, and access internet services (Bhrugubanda, 2015), allowing them to access online data or upload sensor data and communicate with one another. Various CPS machines can work and communicate to form complex systems, unlocking new potential and capabilities. CPS can create virtual copies of their physical components and systems, known as digital twins (DT), through the digitalization of collected data. Integrating cloud computing services and WSNs are key to CPS, as the network can control system dynamics, middleware, and software (Bhrugubanda, 2015). DTs can be formed with the data received through WSNs which can generate virtual production lines, and process data to make correct actions on both the DT, and physical systems. CPS are made up of microcontrollers, sensors and potentially actuators which interact with physical objects and process data. Smart machine technology paired with the potential to communicate with each

other allows for planning tasks and choose or generate new optimized production strategies. Modelling a human perception of the system and various environmental conditions which it may undergo are important to decision making, and these CPS can evolve to operate under new conditions and unreliable environments (Bhrugubanda, 2015)

2.2.2 Usage of CPS in Industries

CPS can be applied in diverse ways to many industries, and benefit from them in multiple different ways, a few examples of these being agriculture, energy management, environment monitoring, transport, medical, and in manufacturing. In agriculture, CPS can increase food consumption efficiency and production capabilities with the use of precision agriculture, intelligent water management, and more efficient food distribution (Chen, 2017). Through the constant monitoring of WSNs on crops and their environmental conditions, maximum output can be achieved. Smart pest control and rat detection systems can be used in the agriculture sector which significantly reduce wastage, contamination and cost, potentially increasing supply-demand for food.

Most CPS devices require lower power than alternative design choices, making them an attractive option, however supply and demand of required energy is often inconvenient (Shi, Wan, Yan, & Suo, 2011). Smart grids are a large advance in the implementation of CPS, providing more

adaptive and optimized power generation, meaning power can be distributed and consumed efficiently. CPS can be used in a large variety of environments including mountains and rivers, and can monitor them to quickly respond when needed, such as in the case of disasters, giving very quick warning in such situations. When exposed to these environments, these conditions may damage equipment and influence the safety and reliability of the system (Chen, 2017). These systems should be able to withstand long durations of time without any human interaction. Intelligent transport which implements CPS can improve the safety and coordination of traffic through the use of sensor technology, communications and control systems. Communication among vehicles with these technologies can result in far lower fatalities. A big challenge faced in the development of autonomous vehicles is the accuracy requirement is high. Existing GPS-based localization provides an accuracy of 5m, whereas the requirement in vehicular communication environments is 50cm (Alam, Balaei, & Dempster, 2013). In the healthcare industry, CPS play an increasingly significant role, focusing on smart sensors for real-time patient health monitoring and warning (Chen, 2017), giving continuous quality care to patients. Current challenges being faced in this sector include context-aware intelligence, autonomy, security and privacy, and device certifiability (Lee, et al., 2012).

CPS are key technologies for the development and improvement of smart manufacturing. Smart manufacturing means the use of embedded software and hardware to optimize productivity in the manufacturing of goods or delivery of services (Gunes, Peter, Givargis, & Vahid, 2014).

Centralized control systems are key to optimizing manufacturing, focusing heavily on modelling, conceptualization, utilization, and automated warehousing systems.

2.3 Cloud Computing

Another cornerstone of I4.0 is cloud-computer. Essentially, cloud-computing is the access to computer services on demand, through the Internet. Companies can rent storage or applications as they are needed from a cloud service provider, instead of committing their own investment or infrastructure, allowing for companies to pay for what they need when they need it as opposed to paying for an entire system and upkeeping the maintenance themselves (Ranger, 2022). Cloud-computing infrastructure makes up over a third of IT spending in the world, proving its importance to industries in the modern world. Cloud-computing can be characterized into three different categories: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS).

IaaS is when computer-based infrastructure can be rented over the Internet, including storage, networking or servers. Oracle's research found two thirds of users find IaaS as an easier alternative to innovate and significantly reduce maintenance costs. The largest problem with IaaS is the security of sensitive information being hosted by external companies. PaaS includes the infrastructure included in IaaS but also allows for applications and software to be developed,

specific to the situation in which the service will be applied. This may include programming software, along with database management or operating systems. SaaS includes the delivery of custom applications, often with hardware, storage, or operating systems irrelevant, this allows for these services accessed through apps or web browsers (Watts & Raza, 2019).

Due to companies which use these services not having to invest in computing infrastructure or servers, large amounts of money can be saved, allowing for them to use computers extensively throughout their means of production with lower costs and yielding greater returns. Not being forced to buy and upkeep servers while maintaining software and applications is an extremely large benefit as it is taken care of by the supplier. These services can be especially beneficial to smaller companies which have limited access to and use of computers, meaning they would not need to hire in-house talent to do these jobs and most likely have less skills (Ranger, 2022). This allows for companies to put more focus and investment into their projects without worrying about big upfront costs and allowing for easy access new software without IT procurement. Depending on the requirement and current demand, it is easy to upscale or downscale as needed. The largest factor which pushes companies from implemented cloud-computing into their work environment is that companies are reluctant to share sensitive information with external parties as it may also be used by competitors. Using the same cloud services as competitors may also make it more challenging to gain business advantages. Another factor which could push companies from using cloud services is that moving pre-existing data to a cloud infrastructure

may be complicated, and potentially expensive, which may outweigh the benefits which they provide, dissuading companies from doing so.

A key concern companies have with cloud-computing is their security, even when these security breaches are extremely rare. In order to increase levels of security, devices which need to be readily accessible over the Internet should be segmented into its own personal network and then have the network restricted. Any intrusions can be detected by monitoring the network and identifying unknown traffic, informing the user that there are potential security breaches. Before a company adopts cloud computing technology, they should know that any of their sensitive information being hosted by third-party providers may come with risk, and great care must be taken when adopting this technology (Stergiou, Psannis, Kim, & Gupta, 2016).

2.4 Wireless Communications

Another key concept to I4.0 is wireless communication. In the modern day, most communications are wireless and involve the transmission of data without wires and cables. There have been many fast advancements to wireless technology due to their benefits to both business and personal use. Their speed, flexibility and efficiency have made them one of the most important tools in I4.0, allowing devices to connect to the Internet while roaming. Recently, this technology has become cheap, high speed and easy to install. Networks are able to be accessed just about

anywhere without requiring cables, allowing for devices and work to be connected to the Internet from remote areas. As mentioned in section 2.3, the biggest downside is the threat to security, with data able to be stolen if not properly secured (ASM Technologies). There are various types of wireless communication technology, as shown in Table 2.1.

Table 2.1 Wireless Communication Descriptions

Wireless Communication Protocol	Description
Infrared (IR) wireless	Used for short to medium range communications, data is transferred as IR radiation
Satellite	Data is sent to satellite, amplifying signal and sending back to earth
Wi-Fi	Uses routers to send and receive wireless data and access to the Internet
Mobile communication	Single frequency band communication
Bluetooth	Allows devices to connect and transfer data wirelessly, commonly for music, headphones and peripherals
Zigbee	Transmits data across long distances at lower speeds and low power

Wi-Fi is a key wireless communication protocol in I4.0 as it provides devices with access to the Internet allowing them to access cloud-based services. The technology uses transmission and reception of radio and electromagnetic waves (McFadden, 2019). These networks have frequencies of 2.4GHz and 5GHz, with data rates of roughly 11mbps (1.375MB/s) and 54mbps (6.75MB/s) respectively.

2.5 Sensor Technologies

Sensors are another key cornerstone to I4.0, without them I4.0 would not be possible. Sensors read the desired property of an entity or environment, such as: temperature, humidity, vehicular movement, lightning condition, pressure, soil makeup, noise levels, presence or absence of substances, mechanical stress levels, characteristics such as speed, direction or size of an object (Akyildiz, Su, Sankarasubramaniam, & Cayirci, 2002). Sensors quantify these properties by converting them into digital signals which can be used for readings and further computer processing (Yokogawa).

In I4.0, sensor technologies are often used for monitoring, identifying issues, analysis and decision making, meaning they can improve efficiency of different processes or ensure product quality. Sensors are commonly used in I4.0 in combination with wireless communications, to

form Wireless Sensor Networks (WSNs), which collect data from points of interest in the field and send them back to the user through wireless communications.

When using multiple sensor types in unison on one specific sensor node, a lot of information can be gathered about the subject, allowing DTs to be formed. In manufacturing processes, this allows for operators to measure accurately and make advised decisions on the processes to ensure products are being procured efficiently and in an appropriate manner (Yokogawa).

2.6 Internet of Things

Internet of Things (IoT) is essentially the term used for everything that can connect to the Internet, communicate with each other, commonly used to describe devices which wouldn't be expected to connect to the Internet. In I4.0, this covers devices including sensor technologies which connect to the Internet providing monitoring to give information for an appropriate action or automation via a feedback loop. Just about any physical device can be effectively turned into an IoT device when connected to the Internet for communications or control, including simple objects like lightbulbs, which can be connected to control via Wi-Fi (Ranger, 2020). Large assemblies may contain multiple IoT components, such as new and modern engines which can contain hundreds of sensors and be provided with feedback to ensure its functionality.

Recent developments in cheap microchips and microprocessors allow IoT to take off and become cost effective for use in various applications. Businesses can access data analytics easily with IoT, allowing them to make changes and gain a deeper understanding of their devices. When employed in manufacturing environments, sensors can be added to systems which can receive performance data, and notify the company when things are likely to fail, reducing maintenance costs and lowering downtime.

IoT devices can generate extremely large amounts of data, and provide information on all sorts of potential improvements, including physical layouts, methods and techniques. Due to the large amounts of data, metadata also becomes a factor, as data must be well managed. This can be done in SQL databases to bring more structure to unstructured information. Data needs to be managed and processed in ways which it can be useful to people and bring more understanding. Due to the large amounts of data produced by IoT devices, many companies who employ such technology often choose to process their data through cloud-computing services for savings. There are many different ways in which IoT devices gather data, typically involving Wi-Fi, and ZigBee or Bluetooth, however, it is expected in the next few years that 5G networks are going to become a staple part of IoT and I4.0. 5G networks allow for over a million devices to be connected per square kilometer, meaning very high numbers of sensors and devices are able to be applied in small areas, allows for high scalability of IoT in industry (Ranger, 2020). AI systems can be trained through analysing data and give future predictions based on it.

2.7 Smart Industries

2.7.1 Smart Manufacturing

In the manufacturing sector, the implementation of I4.0 technologies can provide enormous benefits. Smart factories are a large step up from factories with simple automation methods, by adapting operations to one large fully interconnected system. By collecting data from WSNs, smart factories are able to learn and change according to the requirements of the system. Floor usage in these factories can be fully optimized as DTs are formed and can be viewed from computers, making it easy to read information related to manufacturing processes and how components move through factories, and enable maximum production capacity. Implementing WSNs and IoT into smart factories allows for accurate use of the Just in Time (JIT) operations idea for deliveries of materials and products, maximizing cost savings. Machines can communicate with one another through a centralized control system, continuously feeding data to one another, letting them know estimated cycle times, task completions and inventory levels, helping to make predictions about actions to take and provide the operators with information. These factors help to make factories more agile and fast-moving, with lower downtime and making facility adjustments easier.

The largest issue faced by manufacturers who are looking to implement I4.0 to upgrade to a smart factory is how complicated and expensive the initial setup of equipment, technologies and new infrastructure can be. Typically, manufacturers start off by implementing these systems to small operations and tasks, then scaling them up over time as they can. High levels of automation are already very key in modern-day factories; however, these automated tasks are often limited to single or defined methods. Through the implementation of I4.0 and AI technology, CPS can evolve systems to make more optimal decisions and conduct themselves in a human-like manner. I4.0 provides smart factories with a high level of adaptability, meaning there is a large potential to make changes as required or upscale due to high demand. Operations can be almost entirely automated with very little to no human interaction at all, while still maintaining high levels of reliability (Burke, Laaper, Hartigan, & Sniderman, 2017).

Audi opened up a smart factory for the new model Q5 in San Jose Chiapa, Mexico, with employees monitoring development of cars through centralized control systems (Bose-Munde & Finus, 2019). The focus is trials and experiments which are conducted for driverless trucks and automated forklifts, which when combined with transport drones, urgent parts are able to be delivered to the production line quickly. The processing time and delivery times of automatic orders are digitally visualized for employees (Wagner, Herrmann, & Thiede, 2017).

2.7.2 Smart Grids

IoT, CPS and WSNs can be integrated into grid networks to create smart grids, which move towards more reliable, efficient and safe energy. Smart meters will monitor the consumption, storage and creation of energy, which can then be reported to the service providers, providing them with information regarding the energy demands of customers and price accordingly. Through the implementation of fog/edge computing, data is collected from smart meters then stored and processed, later used for the optimization of energy dispatch. These steps improve the overall resource utilization and reduce costs, saving both the company and the user money (Lin, et al., 2017). The use of multiple sensors throughout the smart grid means failures and blackouts can be detected, and the grid can then employ adaptive and islanding techniques to continue energy delivery until the issue has been fixed, whereas a traditional grid system would have a hard defined way of power delivery, and if this fails, there is no easy way around it. With these pieces of information, there are effectively three key levels to a smart grid: smart electricity generation, delivery methods and the way energy is consumed (Fang, Misra, Xue, & Yang, 2012).

Distributed power generation will use small scale systems including solar panels and wind turbines to improve the reliability and quality levels of energy usage. These micro-grids can disconnect from macro-grids, lowering disturbances improving quality in both internal and external power supplies. Distributed generators are not easy to put into practice however, as

they require large scale rollouts of renewable resource power generation, solar/wind, which are extremely reliant on weather and therefore subject to large fluctuations in output. On top of this issue, the initial capital costs for these implementations can be quite high. A key problem faced in traditional grids is infrastructure challenges, which could include quickly aging components and the increasing demand/load. The scalability of traditional grids is also low, meaning new materials, electronics and technologies cannot be easily implemented, further driving the development towards smart grids.

2.7.3 Smart Farming

Another large industry found which has great benefits for I4.0 technology is farming. Farming processes are very large in variety and are very different depending on the type of production being undergone. Livestock farming and arable (crop) farming may appear to have different factors at play at first thought, however, there are many similarities between the two. Common factors are at play which have large effects on both these types of farming, such as temperature, soil, pests, diseases, and weather conditions. Smart farming features two types of potential solutions to these issues: active farm monitoring, and feedback loops (Wolfert, Ge, Verdouw, & Bogaardt, 2017).

There are many different sensor technologies which can be implemented into a WSN to provide information regarding the state of the farm, most importantly, data regarding the quality and speed of production, in this case, crop health and growth. Information which can be monitored may include:

- Water and Nutrition Monitoring
- Diseases and Bug Monitoring
- Soil Monitoring
- Crop Health Monitoring
- Environment

The data should be received from remote sensor nodes and transmitted to a processing device which allows the farmers to view this data and make use of it.

Based on the information farmers gather from potential monitoring techniques listed above, they may address the readings with appropriate responses including but not limited to the use of irrigation, pesticides, fungicides, herbicides, fertilization, soil preparation, yield condition and yield storage (Ayaz, Ammad-Uddin, Sharif, Mansour, & Aggoune, 2019). Centralized feedback loops may be used on the data being fed into the system, provided direct feedback on the action needing to be taken, or automated responses.

2.8 Research Gaps, Questions and Objectives of the Current Work

The industry on which the design of the thesis will be based on is I4.0 in agriculture. New Zealand's primary industries include farming and forestry (Nana, 2010), dominated by dairy, beef and sheep farming, forestry, fisheries, and horticulture. Due to the important nature of horticulture to New Zealand, and it being such a large industry with room for improvement, it was decided to implement Industry 4.0 concepts into devices which can make room for more optimized farming.

Based on the Literature Review, it seems that there is a large potential for increasing efficiency in farming, however current work and solutions come with flaws. The largest being that I4.0 solutions to this industry do not have benefits which outweigh the costs and difficulties of implementation of these systems. Typically, these solutions are very proprietary and closed source, meaning they are very hard to expand on or customize to fit certain purposes, and therefore can cost far more than needed. In their current state, I4.0 farming solutions all try to contain the most features and highest level of polishing, forgetting the purpose that they need to be beneficial to farmers in a way that they are highly sought after in a way that farmers who do not implement them are missing out. The costs of these technologies can be extremely high to the point farmers who employ them are the ones losing, and therefore they do not use them. This leaves a big gap in the sense that farmers need to have encouragement to employ a system

which proves to be very beneficial to them, giving them all of the benefits they require from these existing systems, so they can use the feedback provided from them to their advantage, and a customizable program which can be easily edited to suit the purpose they require before they are implemented. This way farmers will only pay for what they need.

This leads to the question, can these I4.0 solutions be applied to the farming sector for a cost that allows for farmers to implement them into their farms, and gain the benefits that existing solutions have without paying costs which outweigh the benefits gained from these systems? The work done throughout this project will aim to fill this gap and give farmers a good reason to use I4.0 solutions, making them easily accessible and implement them into their daily lives.

Chapter 3

Concept, Considerations and Specifications

The aim of this chapter will be defining a way how I4.0 technologies discussed in Chapter 2 can be implemented into a design which can be used in a New Zealand industry. There is a very large scope for ideas and potential solutions which can be used, and many different decisions to be made. The design does not necessarily have to contain all the technologies to implement I4.0, however a few concepts such as WSNs and Cloud servers are key to success. The design should implement these key ideas to an industry which can lead to optimization and save resources, are easily accessible and installed by those who can benefit from them. As discussed in Chapter 2, big problems involved with applying I4.0 ideas comes with a large cost and requires a lot of work to get going, and software often proprietary leading to ongoing costs, dissuading those who benefit from using them. Due to this, a key to this system and project going forward will be to overcome these issues and provide the end-user with benefits which outweigh the price to get them going, and upkeep costs, with minimal maintenance and input required.

3.1 Chosen Concept

Technologies which were discussed in the Literature Review chapter will be used to design these devices and allow for improvement. The devices which are designed will provide a backbone for continuous improvement and changes. This means that when a change wants to be made, or a new type of sensor to be added, it can be done so with relatively low effort and cost. Throughout the development of these devices, soil moisture sensors will be used to send data related to moisture levels in crops. Quite often watering equipment on large farms are automatic and run periodically, this means they even run when they are not needed, and soil moisture levels are sufficient – such as the days following rain. This can be seen as space for improvement and resources can be saved, in this case water.

Potential space for improvements could include observation of crops, livestock, soil and atmosphere. The monitoring of water quality and soil health with the constant entry of measurements to an online database can improve animal health, lower death rates, reduce waste, save resources, provide insight into water usage and lower labour input (Fancom, n.d.). Farm processes vary based on the type of production (livestock, arable farming etc.), however there are common factors including conditions such as temperature, soil, pests, diseases and weather. Sensors/monitoring, analysis and decision making ensure correct systematic behaviour (Wolfert, Ge, Verdouw, & Bogaardt, 2017).

3.2 Design Considerations

There are multiple components operating on different levels that are required to design an IoT-based smart farming device.

These include:

1. Physical Structure – this includes all physical components which are to be implemented enabling smart farming.
2. Data Acquisition – this includes all the components and programming involved in the collection of data from sensor nodes.
3. Data Processing – this is what the devices do with the acquired data, this will involve receiving data on a coordinator (base station) which receives signals from all nodes then processing the data into useful information.
4. Data Analytics – this involves uploading the data to a designated cloud-server which can be accessed by the end user on any computer or phone device which has access to the Internet. It is then up to the end user to make decisions based on the data which has been received from the system. (Farooq, Riaz, Abid, Abid, & Naeem, 2019)

A bottom-up approach will be taken to achieve these requirements, starting from the end-goal then working backwards on how to achieve it. As discussed in Data-Analytics, the end user must be able to access data on any computer or phone device, assuming they have Internet access. This means that data must be uploaded to a cloud-server, so the design must include a way to upload data to this server, this could potentially be a computer or device which can connect to Wi-Fi.

3.3 Design Specifications

Typically, Wi-Fi routers operating on the 2.4GHz band can reach up to 150 feet (45.7m) indoors and 300 feet (91.4m) outdoors, 5GHz bands reach approximately a third of these distances (Mitchell, 2020). Farms are likely to require data from longer distances than 91.4m away but must use Wi-Fi to upload data to cloud-servers, therefore another type of wireless communication must be considered. This wireless communication should be able to send wireless data from sensor nodes across a long distance.

Based on these factors, a list of design requirements can be produced:

- There must be an intermediate device which receives data and uploads in via Wi-Fi to a cloud server.

- Must run wireless sensor nodes which require as little upkeep as possible, and as little human interference as practicably possible.
- Must be able to receive data from as long distances as possible (within reason).
- Must be able to read accurate farm data from smartphone to be used to decision-making.
- Must be an economically viable option that has benefits which outweigh the cost and provide farmers incentive to implement.

Chapter 4

Design of WSN and Structure for Moisture Monitoring System

The aim of this chapter will be to take the concept of a WSN with cloud server functionality to a physical system. This includes steps taken to achieve a functioning device which can acquire remote farm data, receive it on a coordinator, process it and upload it to a cloud server, providing the farmer with useful data which can then be used for decision making, improving efficiency such as saving resources, money, and time.

4.1 Design Schematic

A few questions come up in order to meet the requirements of the design. The first question is what wireless communication protocol is suitable for this problem? Table 4.1 analyses different wireless communication protocols suitable for IoT devices, their range, data rate and power.

Table 4.1 Table of Wireless Communication Protocols for IoT

Protocol Name	Range	Data rate	Power Usage
Zigbee	900 ft. (274m)	250kbps	Low
Bluetooth	300 ft. (91m)	2 Mbps	Low
LoRa	10 miles (16.1km)	50kbps	Low
Thread	230 ft. (70.1m)	250kbps	Low

Based on this information, the two most appropriate options would be Zigbee or LoRa. A closer look is taken at previous sensors in the farming industry to see and understand which protocols are being used. One case uses 6LoWPAN gateway nodes together with a Raspberry Pi device with a CC2531 for 802.15.4 protocol used to interface with the local gateway nodes (Ahmed, De, & Hussain, 2018).

6LoWPAN is used to communicate information from different technologies and make it interoperable, with data being collected from sensors and sent to IoT cloud (Ahmed, De, & Hussain, 2018). Another case consists of Raspberry Pi as a base station and several wireless sensor nodes using the Zigbee protocol (Nikhade, 2015). The sensor nodes are made up of a combination of sensors, microcontroller and a Zigbee transceiver (XBee module).

In a third case, it is noticed that XBee devices and the Zigbee protocol were also used, this is likely because the protocol fits the situation appropriately, using low power and sending data across long distances. Due to the nature of sensor data, high data transfer rates are unnecessary, which is the largest con of this protocol, making it very suitable for this application. XBee devices have an API developed for them, meaning they can be programmed easily through different IDEs on computer OS.

Each WSN will consist of one coordinator, a number of routers (if required for range expanding), and endpoints, which in this case are our wireless sensor nodes. The coordinator will receive data from endpoints and routers, process it into useful information, then upload to a cloud server. The coordinator must be able to connect to the Wi-Fi, so a suitable coordinator would include a computer, Raspberry Pi, or an Arduino with a Wi-Fi shield. For the purpose of this project, a capacitive moisture sensor will be used to test the WSN, and system developed.

4.2 Wireless Communications

As seen in the XBee datasheet, XBee S2C devices have an outdoor RF line-of-sight range of up to 1200m, making them ideal for farms. These devices cost \$29.10 NZD each, which is quite cheap considering the transmission range. They can also be used as intermediate routers, receiving data from end points, and transmitting it to the coordinator, further increasing the range. They contain

APIs for each of the following programming languages: C++, Java, Python and for Arduinos, this will make them interface well with modern computer technologies.

For sensor data to be accessed from a mobile device, it must be uploaded to a cloud server through the Internet. Modern computers all have Internet access, whether it be through Wi-Fi or LAN, making them a suitable option for the coordinator. They are however largely unnecessary to upload small amounts of data to a cloud server and will consume large amounts of power for little reason. Raspberry Pi, as seen in Figure 4.1, is another option, with Wi-Fi built into a LINUX based OS, they are small devices which consume low power, cheap and can be programmed through IDEs and easy to interface with GPIO pins.



Figure 4.1 Raspberry Pi

The last choice which is suitable for our coordinator is an Arduino, these are very cheap and easy to program. No OS is installed, and they run one program on repeat which is flashed to the device through their PC application. They can easily receive data from XBees through a shield. Arduinos

do not have Wi-Fi built into them; however, it can be added through a shield which can be installed on top of the board.

4.3 XBee Programming

XBees can be programmed through Digi proprietary software, XCTU. When connecting an XBee to a computer through USB port using an adapter, RF devices can be searched for in XCTU. Parameters for the individual XBee modules are configured through XCTU, such as ID, DH and DL. All XBee modules consist of Serial numbers: high and low (SH and SL), which are used as unique identifiers to each module, and can be used to identify where sensor data is coming from. The DH and DL parameters of any Endpoint/Router devices should be set to the SH and SL of the intended receiving module, enabling a data path to be created. Figure 4.2 shows an individual XBee module, the pins as they are defined by the manufacturer, and the pin numbers.

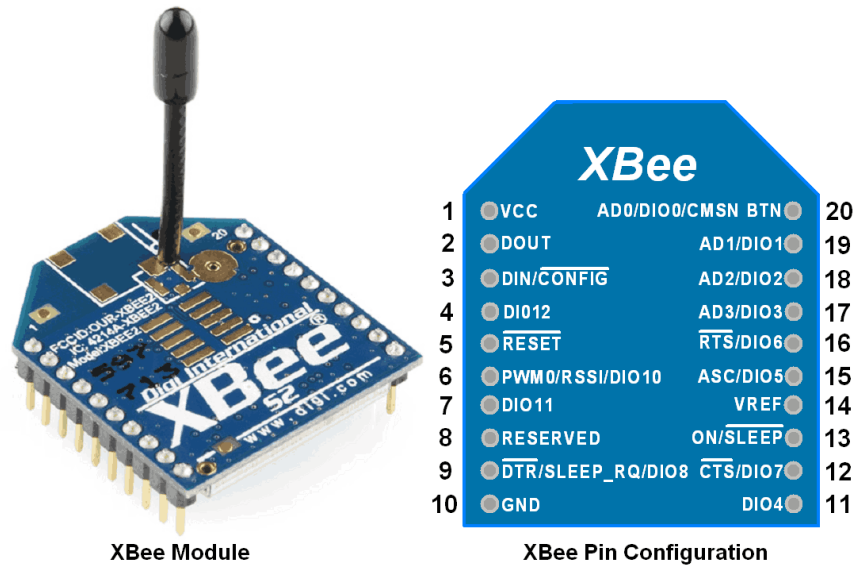


Figure 4.2 XBee Module and Pin Configuration (Electronic Wings, 2018)

▼ **Networking & Security**
Modify networking settings

i	CH Channel	D		
i	ID Network ID	3398		
i	MT Broadcast Multi-Transmits	3		
i	PL TX Power Level	Highest [4]		
i	PM Power Mode	Boost Mode Enabled [1]		
i	RR Unicast Retries	A	Retries	
i	CA CCA Threshold	0	-dBm	

Figure 4.3 Networking & Security parameters of XBee

Figure 4.3 shows a few parameters which can be changed on the XBee module, including CH (Channel) and ID (Network ID). These will be key pieces of information contained in the XBee which allow it to connect on the same network to other XBees which will be included in the WSN.

Parameters can be changed and written to the device to suit the needs of the user. In this case,

we will be sampling for data, so I/O sampling will need to be programmed. XBee modules have a built in ADC which can be used when working with analog data, as will be performed in this design. Figures 4.4 and 4.5 show the configuration settings for I/O sampling, with pins 17-20 used as ADC for analog samples.

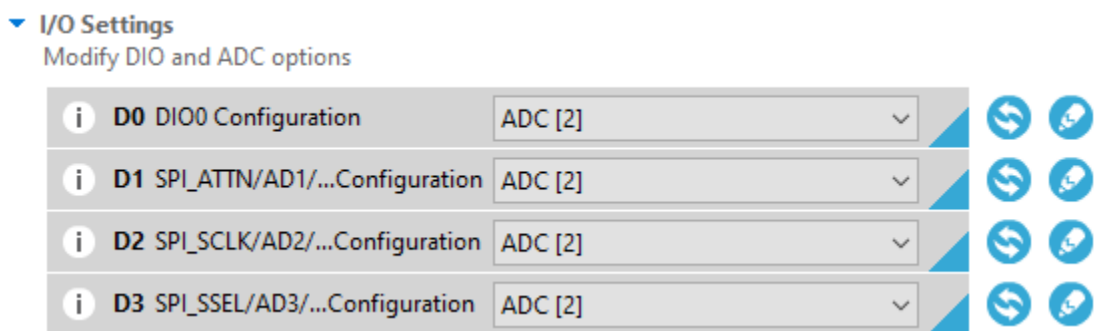


Figure 4.4 I/O Analog sampling configuration

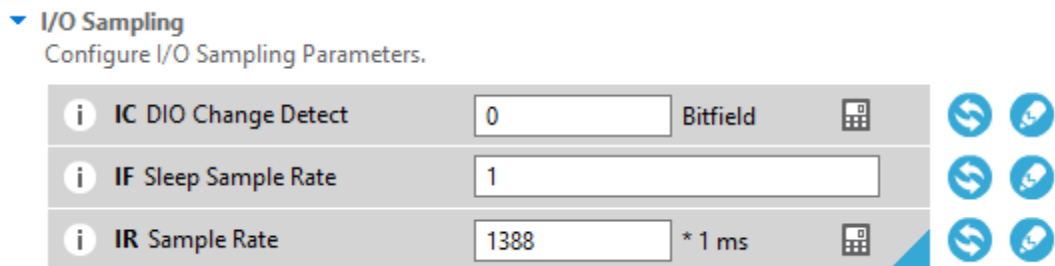


Figure 4.5 I/O Sampling Rate

Refer to Appendix F for full XBee Configuration profiles.

4.4 Arduino

The first device which was used to attempt to create a network was the Arduino, as shown in Figure 4.6.



Figure 4.6 Arduino

This device is essentially a flash-able microcontroller, which uses a ATmega328P microcontroller combined with a ATmega 16U2 microprocessor. The Arduino comes equipped with 32KB of RAM and 16MHz CPU, so it will not be able to support large programs or big data, potentially limiting future scalability. A Pinout schematic is shown in Figure 4.7, refer to Appendix B for pin functions and descriptions.

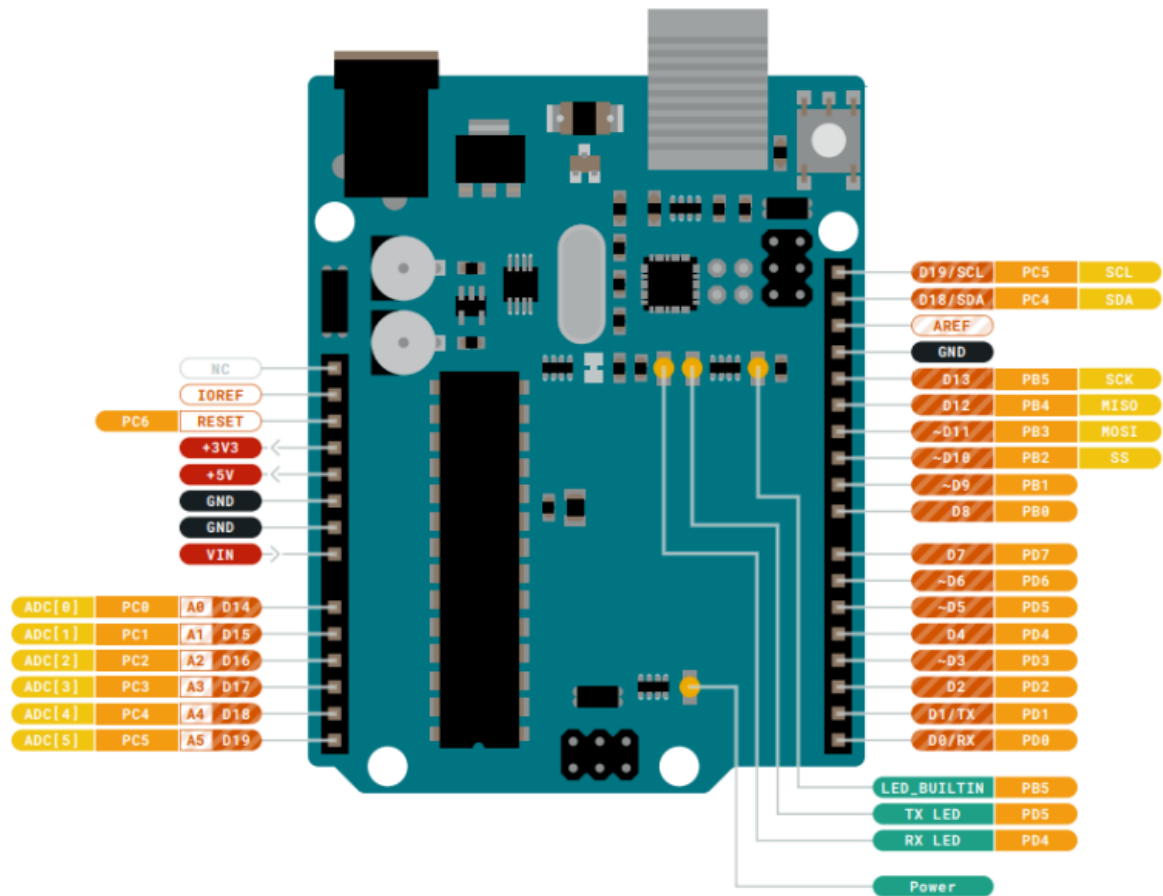


Figure 4.7 Arduino Connector Pins

Arduino boards can be powered through USB connectors to a PC, or through the V_{in} pin connected to 2.7V-5.5V. Analog data can be received through the A0-A5 pins, where digital data can be received through the D0-D9 pins. XBee modules output digital data, so they can be input to the appropriate digital pins. This will however only be able to generate digital data, in the case for XBees, 10-bit. This means that one digital pin can only receive 10 bits of data, and when multiple sensors, sensor types and sensor nodes are introduced, the data will become very low

resolution. When the TX pin on the XBee is connected to the RX pin on the Arduino, an API frame will be sent to the Arduino. The frame contains HEX data which represents information about the incoming data. Here is an example of a HEX frame: 7E 00 18 92 00 13 A2 00 41 C1 3E 63 FF FE C1 01 00 00 0F 00 10 03 59 02 29 01 00 97. Table 4.2 shows this can be interpreted as:

Table 4.2 Example Frame Received from Remote XBee Device

Field Name	Example HEX data	Description
Start delimiter	7E	All XBee API frames begin with 7E
Length	0018	Length of the frame
Frame Type	92	92 indicates that the frame is a generated I/O sample
64-bit source address	0013A20041C13E63	64-bit address of XBee which sample was received from
16-bit source address	FFFE	16-bit address of XBee which sample was received from
Receive options	C1	Additional info about the packet
Number of samples	01	Indicates number of samples included in the frame
Digital channel mask	0000	Bitfield mask indicating which I/O lines set as digital

Analog channel mask	0F	Bitfield mask indicating which I/O lines set as ADC
Digital Samples	0010	Bitfield containing digital sample data. This field not included if 0000
Analog Sample 1	0359	Value of ADC inputs, each ADC will have 2-byte field based on number of inputs
Analog Sample 2	0229	
Analog Sample 3	0100	
Checksum	97	Checksum of API Frame

The key data gathered from these frames in this case would be the analog sample data, and the source addresses, as we want to know which sensor node is sending the data. The XBee for the Arduino can be easily programmed by mounting an XBee shield onto the board, as seen in Figure 4.8. When the Arduino is connected to the PC using a standard USB cable, the Arduino will naturally try to run the program to which it has been flashed, and the XBee is unable to be programmed. To get around this, the GND pin is wired to the RESET pin on the Arduino, this means the board will not run its program, and is essentially bypassed, allowing access to the XBee device.

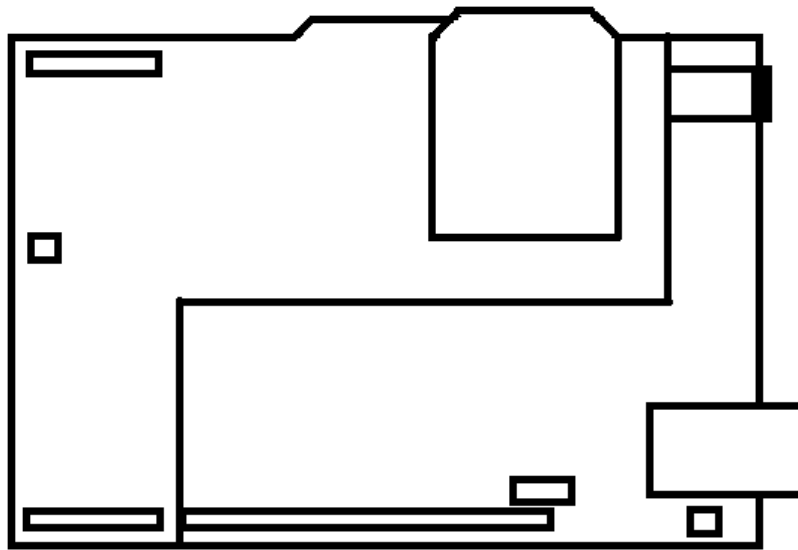


Figure 4.8 Arduino with XBee Shield

XBee frames can easily be read and interpreted on the Arduino as an API has been developed to support these types of setups, this library is written in the C++ language. This library is accessed on the Arduino documentation pages (Rapp, 2015), however, one key problem is encountered when trying to integrate the Arduino board with the Wi-Fi chip, as both the Wi-Fi chip and XBee require access to the TX and RX pins. When the XBee shield is mounted on the Arduino board, both these pins are being used, so the design inherently will not function. The Arduino also has very low scalability in terms of processing data and may encounter problems in the future when it comes to increasing WSN functionality and features. It is for these two key reasons that this potential solution to the coordinator is not to be further used and the RPi will be used instead.

4.5 Raspberry Pi

The second device which was used the RPi. This is much more powerful than the Arduino with 1GB of RAM and a LINUX-based OS. There is an initial setup off an SD card, which is later used as its memory, in which RPi OS Raspian is installed through NOOBS installer. The RPi is used with standard PC peripherals and must be connected to a display to use the OS, however, by enabling VNC on the RPi, and using a VNC viewer application on regular PC, the RPi can be accessed with an IP address on the LAN. There are three programming languages which have an XBee API which can be support on the RPi – C++, JAVA and Python. Python will be used exclusively through this project as RPi comes with a built-in IDE for Python, Python files are very easy to execute as they do not need compiling with any changes, hence making changes very easy to make, and potential future AI functionality. Thonny Python IDE comes preinstalled, however various packages and changes are made to the RPi. The RPi must be interfaced There are two ways which serial data can be accessed in the RPi: through GPIO pins or through USB. Figure 4.9 shows a pinout diagram to be used for GPIO interfacing.

3v3 Power	1		2	5v Power
GPIO 2 (I2C1 SDA)	3		4	5v Power
GPIO 3 (I2C1 SCL)	5		6	Ground
GPIO 4 (GPCLK0)	7		8	GPIO 14 (UART TX)
Ground	9		10	GPIO 15 (UART RX)
GPIO 17	11		12	GPIO 18 (PCM CLK)
GPIO 27	13		14	Ground
GPIO 22	15		16	GPIO 23
3v3 Power	17		18	GPIO 24
GPIO 10 (SPI0 MOSI)	19		20	Ground
GPIO 9 (SPI0 MISO)	21		22	GPIO 25
GPIO 11 (SPI0 SCLK)	23		24	GPIO 8 (SPI0 CE0)
Ground	25		26	GPIO 7 (SPI0 CE1)
GPIO 0 (EEPROM SDA)	27		28	GPIO 1 (EEPROM SCL)
GPIO 5	29		30	Ground
GPIO 6	31		32	GPIO 12 (PWM0)
GPIO 13 (PWM1)	33		34	Ground
GPIO 19 (PCM FS)	35		36	GPIO 16
GPIO 26	37		38	GPIO 20 (PCM DIN)
Ground	39		40	GPIO 21 (PCM DOUT)

Figure 4.9 Raspberry Pi GPIO Pinout

Serial communication is disabled by default and is enabled through the terminal by running the command “raspi-config” then selecting Interfacing Options as shown in Figure 4.10 and enabling P6 Serial. As XBee devices do not use standard pin spacing, breakout boards are used to connect the modules with breadboards or other standard 0.1” spacing.

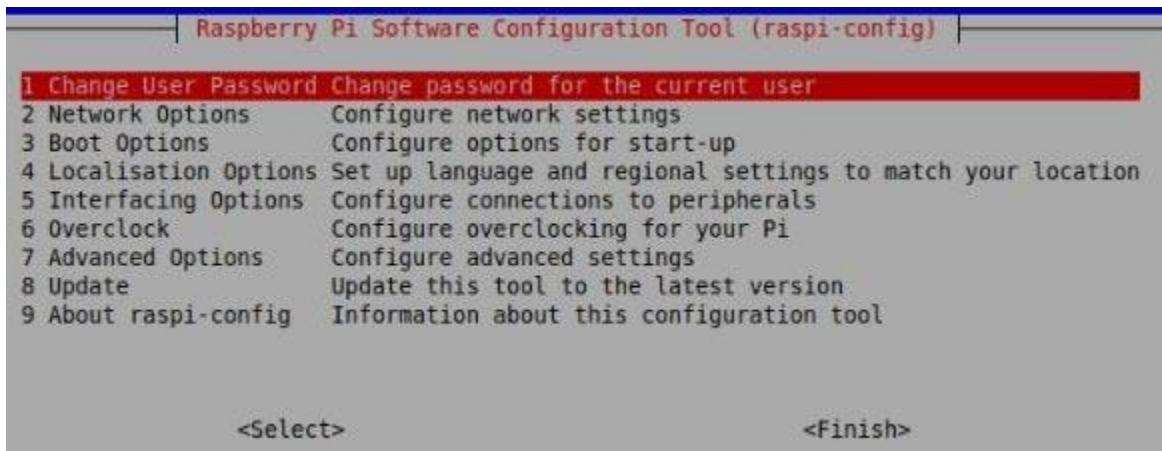


Figure 4.10 RPi Configuration (Jain, 2019)

4.5.1 RPi Schematic

The RPi has two potential ways to interface with XBee: GPIO and USB. The first design which will be attempted will use the GPIO pins for interfacing as shown in Figure 4.11:

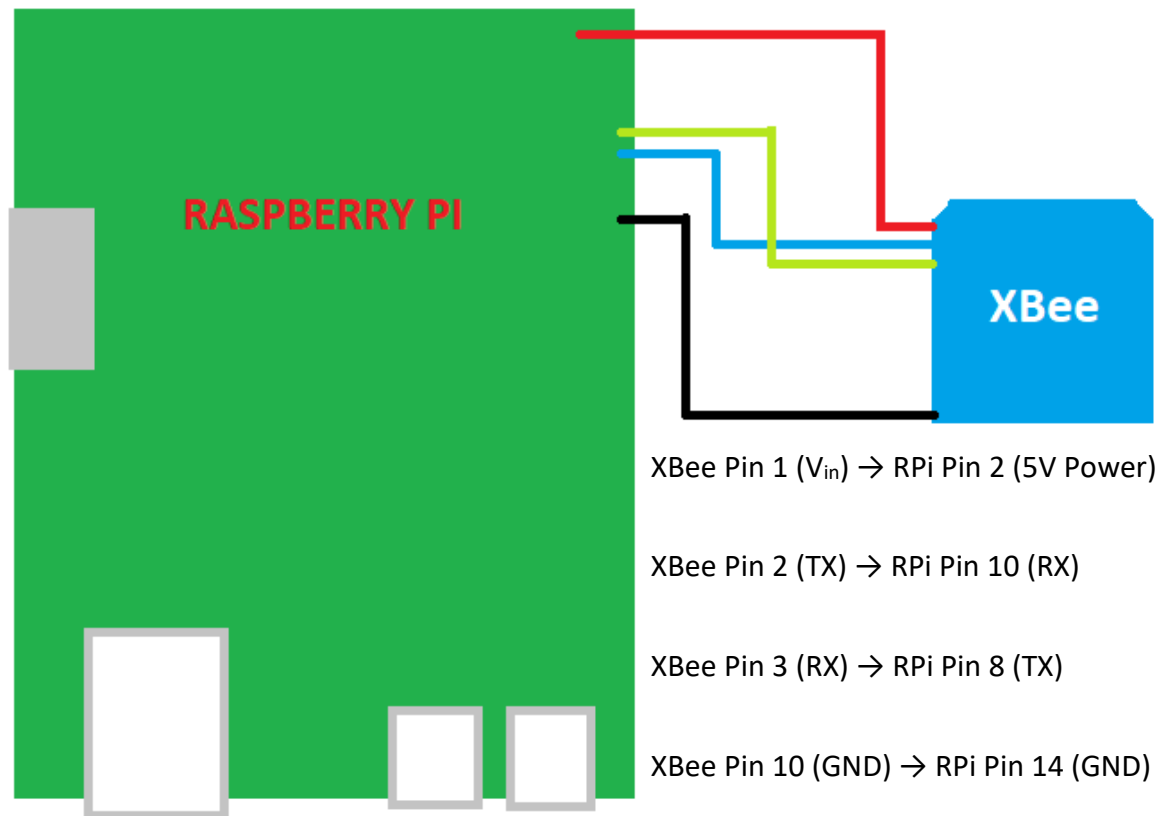


Figure 4.11 Raspberry Pi connected to XBee through GPIO pins

The second potential way is by connecting the XBee to an XBee explorer board, which can plug into the USB port on RPi directly, as shown in Figure 4.12. Of the two potential ways, both should have the same outcome, just reading Serial data through different RPi files. The first method, using GPIO pins requires more wiring and may look a bit cluttered, whereas the second way requires extra parts. Both methods will be used in this design and the more suitable one will be used.

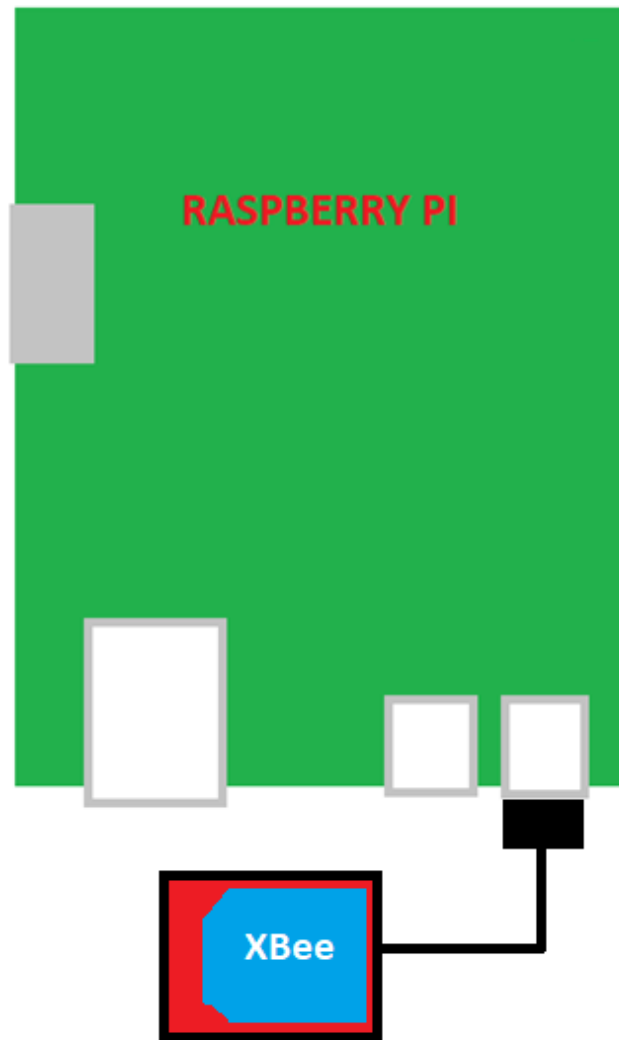
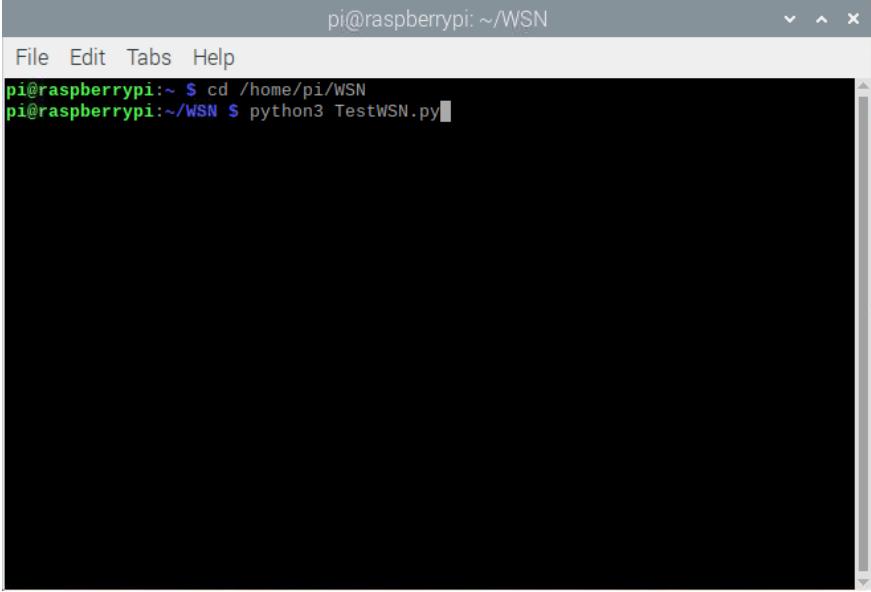


Figure 4.12 Raspberry Pi Connected to XBee Explorer USB

4.5.2 Python Programming

To design the program to be used on the Raspberry Pi, a programming language must be used which contains both an API for the cloud services, and the XBee devices. Potential languages

supported by both devices are C++, JAVA and Python. As discussed in Chapter 4.5: Raspberry Pi, Python 3 will be used exclusively throughout this project. It comes pre-installed on RPi devices, however not all the packages are included. There is a XBee package which must be installed, which can be done so using Pip through the following terminal command: “pip3 install xbee” (Malmsten, et al., 2018). Pip3 is used as normal pip will lead to installing xbee packages to an earlier version of python, with this installation, XBee API can be accessed in Python 3. The python library for XBee can be accessed through the documentation pages (Faludi, 2017). For the purpose of this project, all python script files will be stored in the “/home/pi/WSN” folder. Python script files will be run using the following commands in the terminal, as shown in Figure 4.13:

A screenshot of a terminal window on a Raspberry Pi. The title bar at the top reads "pi@raspberrypi: ~/WSN". Below the title bar is a menu bar with "File", "Edit", "Tabs", and "Help". The terminal content shows two commands being entered at the prompt "pi@raspberrypi:~". The first command is "cd /home/pi/WSN" and the second is "python3 TestWSN.py". The cursor is at the end of the second command.

```
pi@raspberrypi:~ $ cd /home/pi/WSN
pi@raspberrypi:~/WSN $ python3 TestWSN.py
```

Figure 4.13 Setting the workspace in the RPi Terminal

4.5.3 Cloud Servers

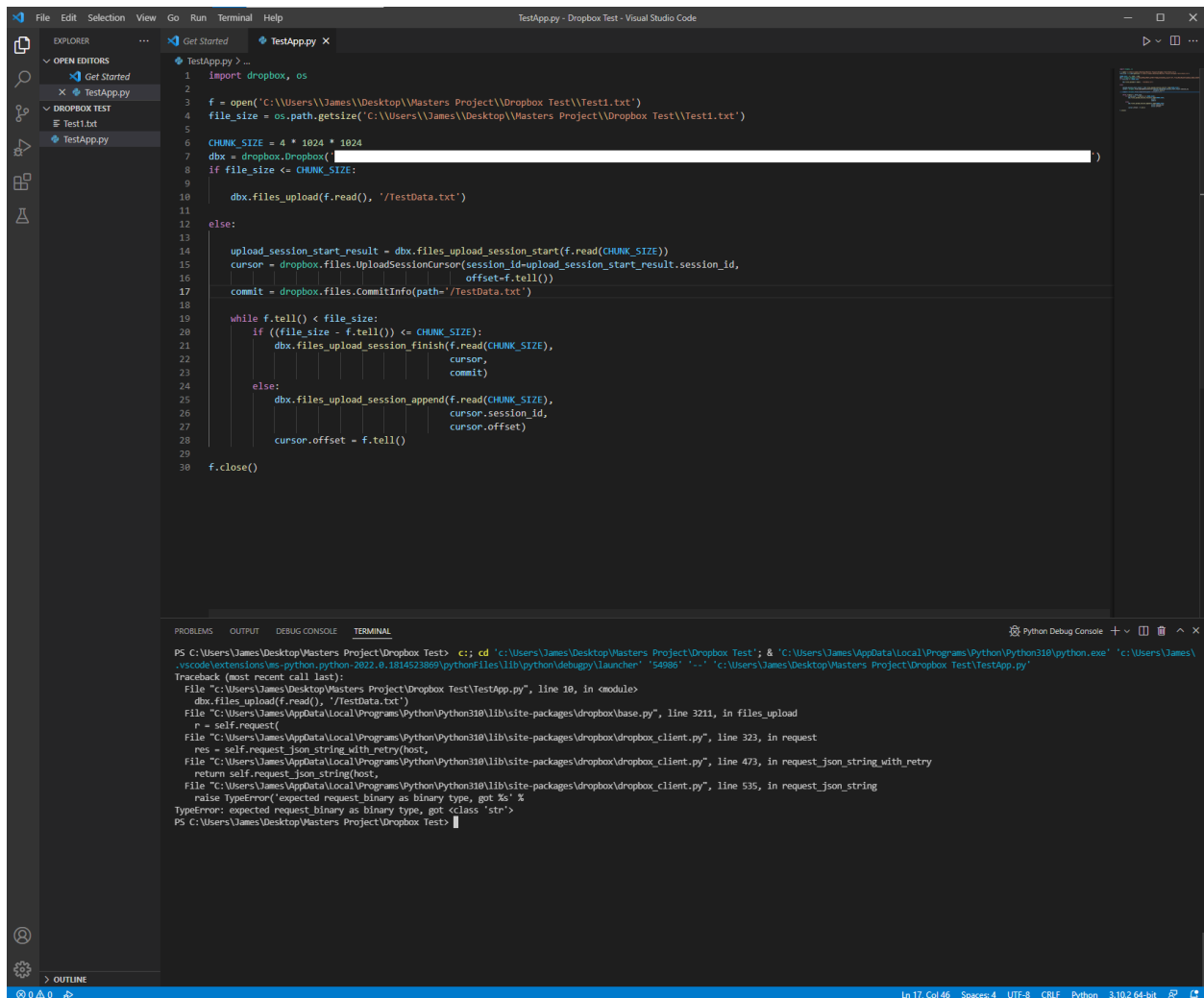
To meet the design requirement of farmers able to access farm data remotely from mobile devices, farm data gathered by sensors must be uploaded to a cloud server, a key component of I4.0. Wireless sensor data for moisture sensors will be relatively small, as any historical sensor data can be deleted with the latest up to date information. Due to this, it is currently unlikely we will run into storage problems as total required cloud storage will be quite small. The important factor for selecting a cloud server to be used in this design simply comes down to the compatibility between the API provided for the cloud server, and the RPi + Python programming language. It is for this reason, that Dropbox is chosen as the cloud server that will host files to be viewed by the end-user. Dropbox has a python API which installed using the following command in the terminal:

`“pip3 install dropbox”`

A private DBX account must be created, which hosts the files to be uploaded containing sensor data, then an API must be created.

After creating an API, an access token is generated, using this token in a python script to create a cloud server object will allow all access to files for uploading and editing. Text files can be created on the local machine (RPi) then uploaded to the server containing the desired data. Due to the intention of the WSN, old data can be overwritten to avoid creating Big Data containing a whole bunch of unnecessary historical information. The script below, shown in Figure 4.14, was

the first attempt to upload a file using the DBX API, the uploaded file was empty and used purely as a test file, however, errors occurred, and they were unable to be troubleshooted, so the script started from scratch.



```
1 import dropbox, os
2
3 f = open('C:\\Users\\James\\Desktop\\Masters Project\\Dropbox Test\\Test1.txt')
4 file_size = os.path.getsize('C:\\Users\\James\\Desktop\\Masters Project\\Dropbox Test\\Test1.txt')
5
6 CHUNK_SIZE = 4 * 1024 * 1024
7 dbx = dropbox.Dropbox('')
8 if file_size <= CHUNK_SIZE:
9
10     dbx.files_upload(f.read(), '/TestData.txt')
11
12 else:
13
14     upload_session_start_result = dbx.files_upload_session_start(f.read(CHUNK_SIZE))
15     cursor = dropbox.files.UploadSessionCursor(session_id=upload_session_start_result.session_id,
16                                                offset=f.tell())
17     commit = dropbox.files.CommitInfo(path='/TestData.txt')
18
19     while f.tell() < file_size:
20         if ((file_size - f.tell()) <= CHUNK_SIZE):
21             dbx.files_upload_session_finish(f.read(CHUNK_SIZE),
22                                           cursor,
23                                           commit)
24         else:
25             dbx.files_upload_session_append(f.read(CHUNK_SIZE),
26                                           cursor.session_id,
27                                           cursor.offset)
28             cursor.offset = f.tell()
29
30 f.close()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Python Debug Console

```
PS C:\Users\James\Desktop\Masters Project\Dropbox Test> cd 'C:\Users\James\Desktop\Masters Project\Dropbox Test' & 'C:\Users\James\AppData\Local\Programs\Python\Python310\python.exe' 'C:\Users\James\
.vscode\extensions\ms-python.python-2022.0.1014521809\python\lib\python\debugpy\launcher' '54986' '-' 'C:\Users\James\Desktop\Masters Project\Dropbox Test\TestApp.py'
Traceback (most recent call last):
  File "C:\Users\James\Desktop\Masters Project\Dropbox Test\TestApp.py", line 10, in <module>
    dbx.files_upload(f.read(), '/TestData.txt')
  File "C:\Users\James\AppData\Local\Programs\Python\Python310\lib\site-packages\dropbox\base.py", line 3211, in files_upload
    r = self.request(
  File "C:\Users\James\AppData\Local\Programs\Python\Python310\lib\site-packages\dropbox\dropbox_client.py", line 323, in request
    res = self.request_json_string_with_retry(host,
  File "C:\Users\James\AppData\Local\Programs\Python\Python310\lib\site-packages\dropbox\dropbox_client.py", line 473, in request_json_string_with_retry
    return self.request_json_string(host,
  File "C:\Users\James\AppData\Local\Programs\Python\Python310\lib\site-packages\dropbox\dropbox_client.py", line 535, in request_json_string
    raise TypeError('expected request_binary as binary type, got %s' %
TypeError: expected request_binary as binary type, got <class 'str'>
PS C:\Users\James\Desktop\Masters Project\Dropbox Test>
```

Figure 4.14 Cloud server test upload text file

A simpler approach to uploading files to the DBX cloud server was taken in the second test script, as seen in Figure 4.15, which worked successfully. Writing simple definitions and running the script step-by-step will make it easier to implement into the coordinator with sensor data. Figure

4.16 shows a successfully uploaded test text file, which was uploaded to the dropbox cloud-server via python script, being accessed through a web browser. This simple method works and may also be used in the uploading of data from sensors.

```

1 import time
2 import dropbox
3 from dropbox.files import WriteMode
4 from dropbox.exceptions import ApiError, AuthError
5
6 dbx = dropbox.Dropbox('')
7
8
9 def main():
10     i = 1
11     while True:
12         writetext(str(i))
13         uploadfile()
14         time.sleep(30)
15         i = i+1
16
17 def writetext(s):
18     with open('Test1.txt', 'w') as f:
19         f.write(s)
20
21 def uploadfile():
22     with open('Test1.txt', "rb") as f:
23         dbx.files_upload(f.read(), '/test.txt', mode=WriteMode('overwrite'))
24         f.close()
25
26 if __name__ == "__main__":
27     main()
28

```

Figure 4.15 Dropbox test upload text file, second script

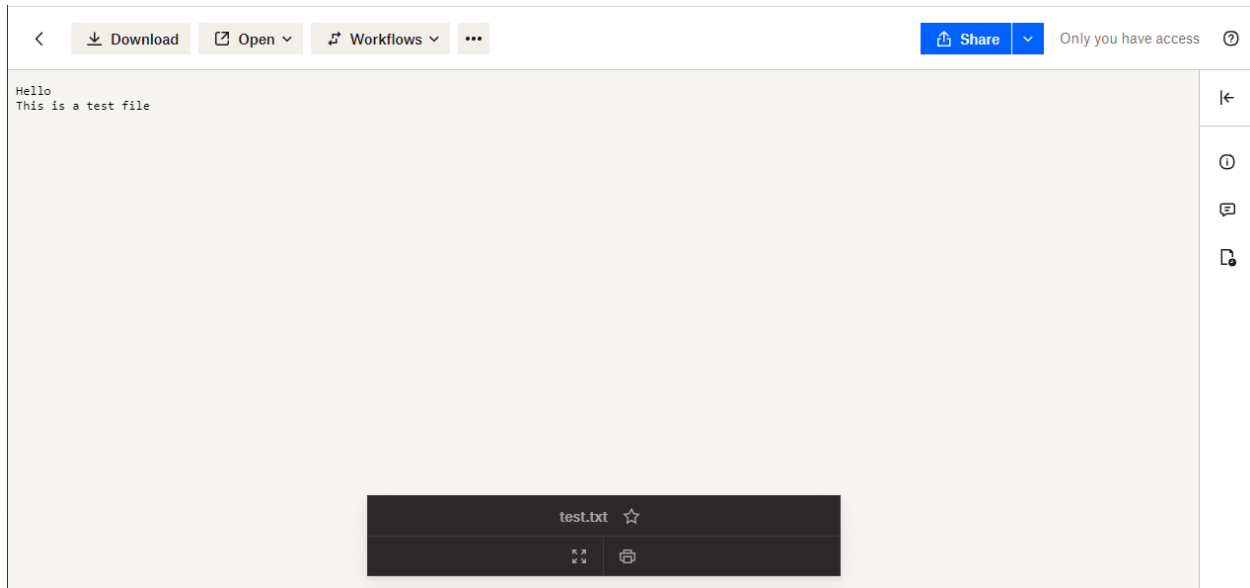


Figure 4.16 Result Test text file on dropbox server

4.5.4 Remote Devices

The remote wireless sensor nodes are the endpoints which collect data from the farms. They must collect this through sensor devices interfaced with the XBee which then transmits this signal back to the coordinator and processed on the RPi. The sensor should ideally run on the same voltage as the XBee so they can be hooked up to the same battery without the need for a voltage divider or voltage regulator, as these lower current and increase power usage respectively. Two AA batteries will supply a voltage of 3V to the XBee and the sensor. A capacitive moisture sensor is used for this as opposed to a resistive sensor, to avoid corrosion interfering with data. The sensor sends an analog signal through its Pin 1 into the XBee, which uses ADC to convert it into a 10-bit digital signal to be transmitted. Figure 4.17 shows the setup of the wireless sensor node.

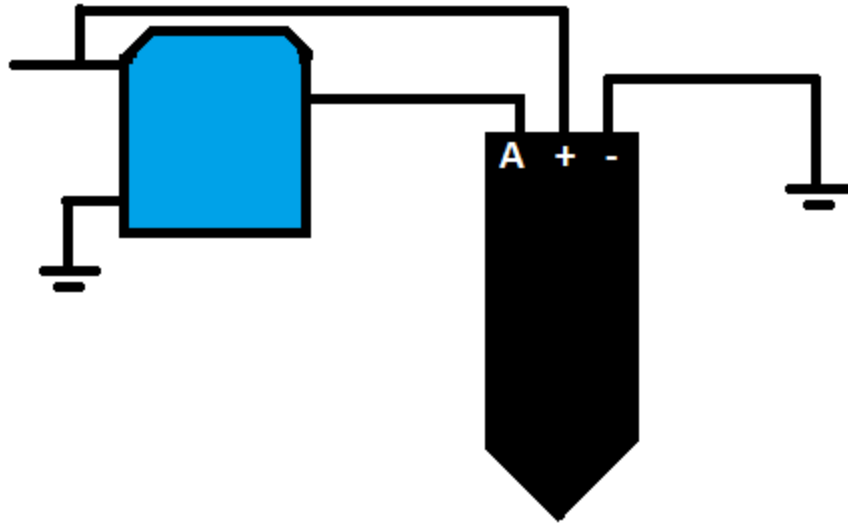


Figure 4.17 Initial Sensor Schematic

XBee Pin 1 (V_{in}) \rightarrow +3V

XBee Pin 10 (GND) \rightarrow GND

XBee Pin 17 (AD3) \rightarrow Sensor Pin 1 (V_{out})

Sensor Pin 2 (V_{in}) \rightarrow +3V

The V_{out} of the sensor is in a range of 0-3V, as per the datasheet, however the ADC on the XBee goes off a reference voltage of 1.2V, so a voltage divider must be used to scale the voltage down going into Pin 17 on the XBee. Applying the voltage divider formula as shown below, Resistor values R_1 and R_2 can be found.

$$V_{out} = V_{in} \left(\frac{R_2}{R_1 + R_2} \right)$$

With the intended V_{out} being 1.2V, and V_s being 3V, R_1 and R_2 can be 10k Ω and 25k Ω respectively as shown in Figure 4.18, the drop in current does not matter as the XBee is only reading voltage. Now that the data has been divided to fit the XBee reference voltage, it can be sent to the coordinator. 10-bit means the voltage of 0-12V will be scaled to the integer values of 0-1023 when received; this resolution is enough to provide sufficient moisture data as high levels of accuracy is unnecessary.

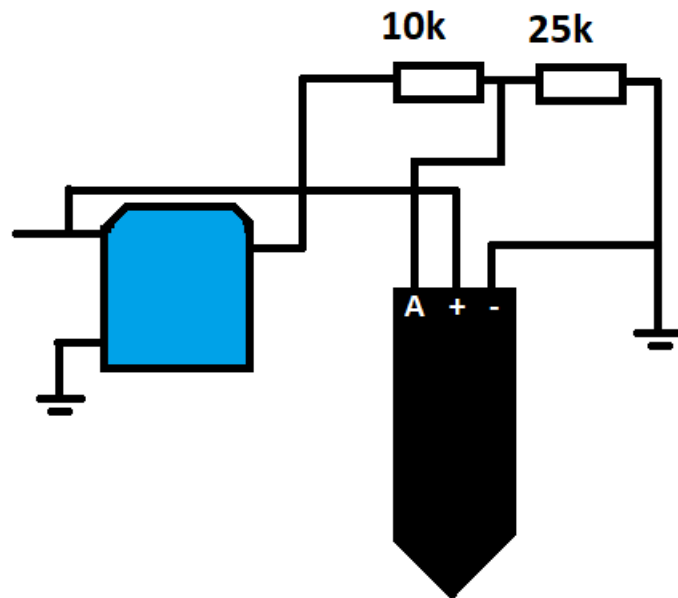


Figure 4.18 Wireless sensor node with voltage divider applied

4.5.5 XBee setup with XCTU

When connecting the XBee to a PC through USB, the RF module can be found in XCTU. The radio module appears on the left, and when selected, the configuration profile window is displayed. This contains all the parameters which can be programmed into the connected XBee module, as seen in Figure 4.19.

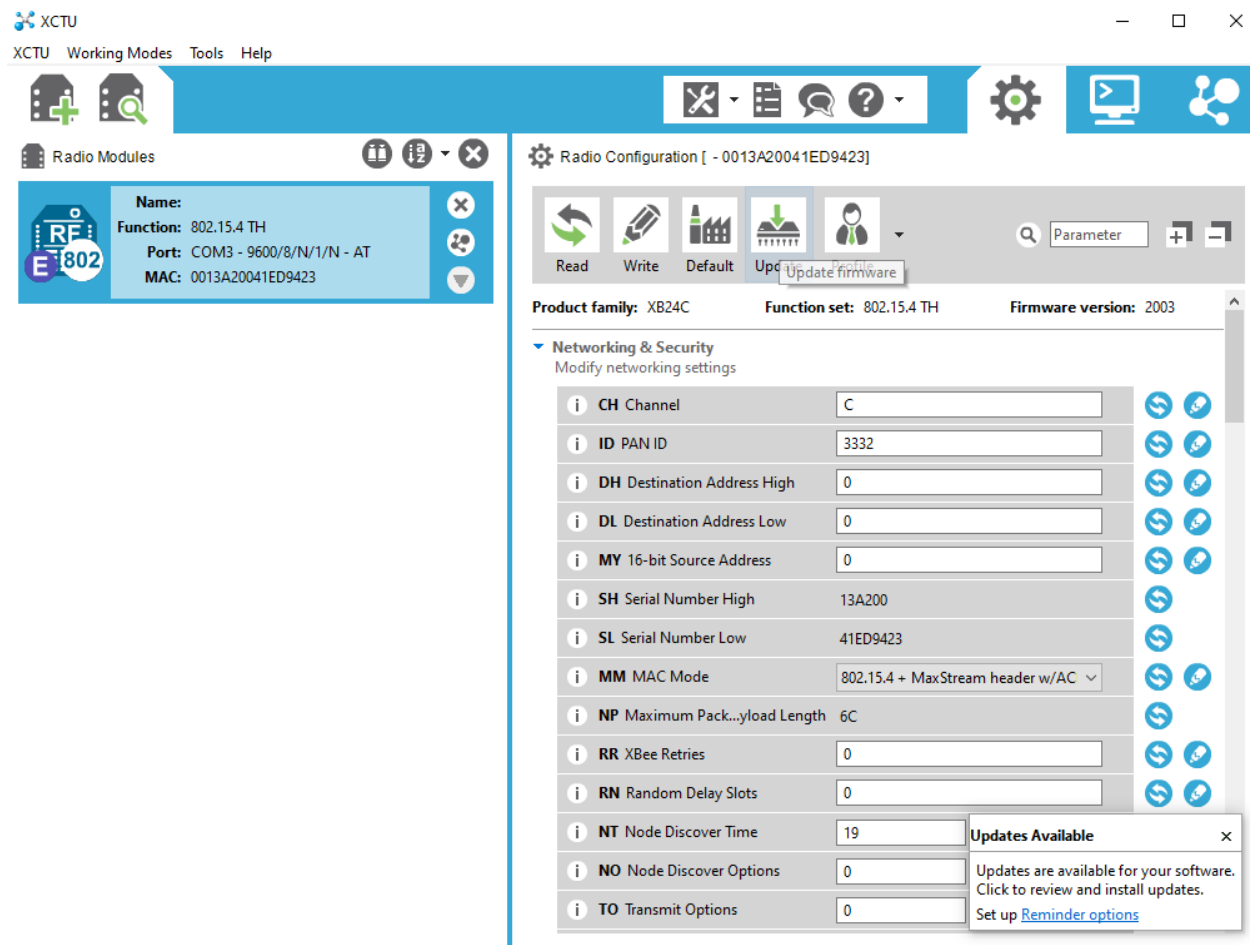
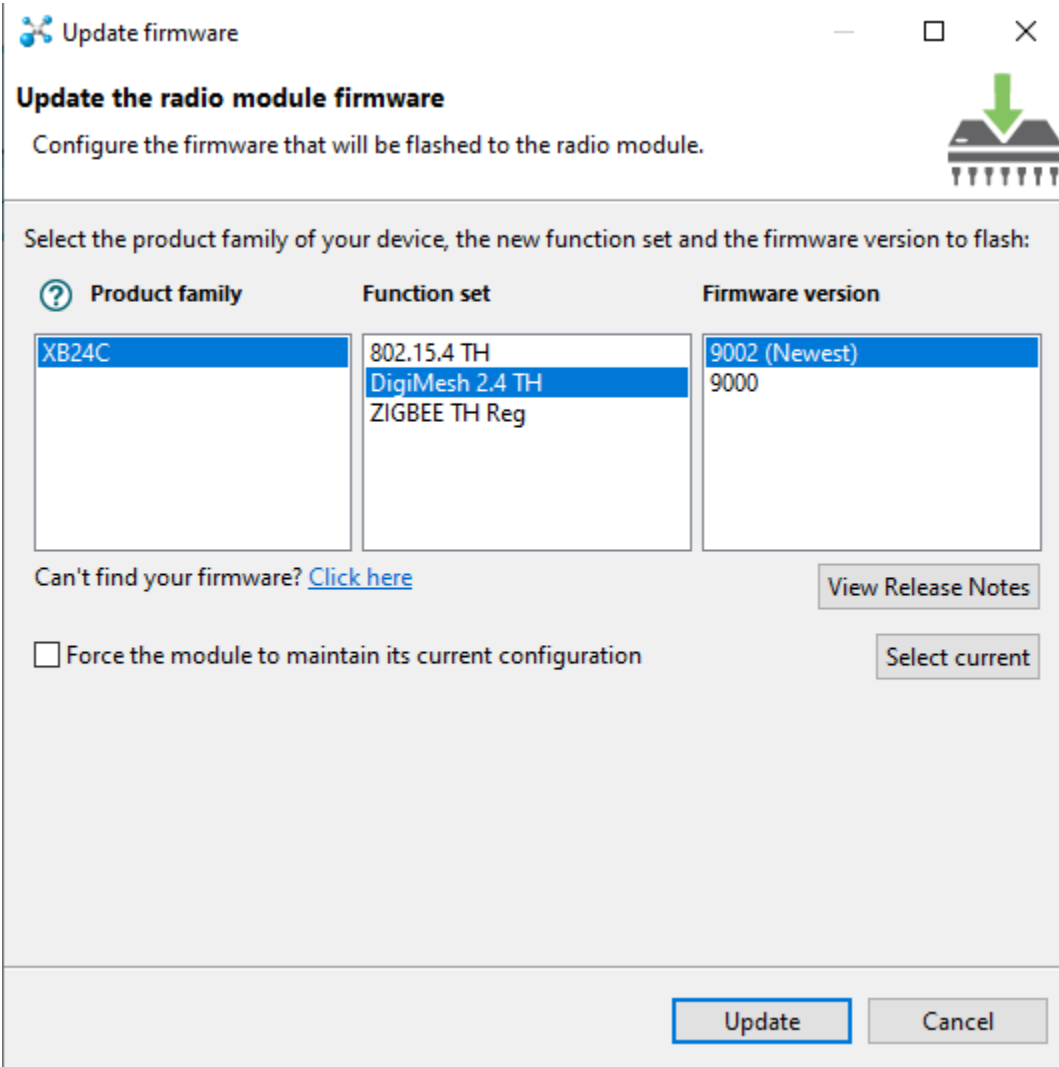


Figure 4.19 XBee Default parameters

All XBee modules in the network are to be used on the same network with the same firmware, so they are all updated to DigiMesh 2.4 TH version 9002, as shown in Figure 4.20. The DigiMesh function is set as it enables meshing to transmit data, and all devices can connect with one another.



Update firmware

Update the radio module firmware
Configure the firmware that will be flashed to the radio module.

Select the product family of your device, the new function set and the firmware version to flash:

Product family	Function set	Firmware version
XB24C	802.15.4 TH	9002 (Newest)
	DigiMesh 2.4 TH	9000
	ZIGBEE TH Reg	

Can't find your firmware? [Click here](#)

☐ Force the module to maintain its current configuration

[View Release Notes](#)

[Select current](#)

[Update](#) [Cancel](#)

Figure 4.20 Setting modules to XB24C DigiMesh 2.4 TH 9002 Firmware

Finally, all devices are setup on the same firmware and can be programmed to communicate with each other. First, the coordinator is programmed. All devices used in the WSN must have the

same Channel, CH, and Network ID, ID. The values of these parameters don't have to mean anything, but must be consistent among all XBee devices, so they are set to "D" and "3398" respectively. The Node Identifier, NI, is set as "Coordinator", which will be used to identify the module in the python program. AP is set to 1, meaning API Mode is Enabled, allowing the python script to read and write XBee Data through their documentation. These are the only parameters which need to be changed for the coordinator as it is only receiving data, not transmitting.

All XBees to be used in wireless sensor nodes must also be configured. With CH set to "D" and ID set to "3398", they are already connected to the network, but destination addresses must be defined and data sampling for the pins enabled. SH for the coordinator is "13A200" and SL "41C13DAF". DH and DL must be configured to these values respectively, which essentially tells the network that this data is intended to be sent to the coordinator. NI, Node Identifier can also be used to identify which node is sending data, so it is defined for each XBee module. AP must also be enabled on these nodes to allow the python script to read node identifiers. I/O settings must be changed to allow data into pins 17-20. For this specific case, we are sending analog data from the sensor into pin 17, so D3, its respective parameter, is set to 2, which means this pin uses a 10-bit ADC. The final parameter which must be set for the wireless sensor nodes is IR, the sample rate. Say we are to sample the I/O pins every 5 seconds, or 5000ms, IR must be set to 1388, which means 5000 in HEX.

With all these parameters set for both the coordinator and nodes, we can now begin sampling for data. When the coordinator is still connected to a PC and connected to XCTU, the terminal is opened and frames start appearing from sensor nodes, as seen in Figure 4.21.

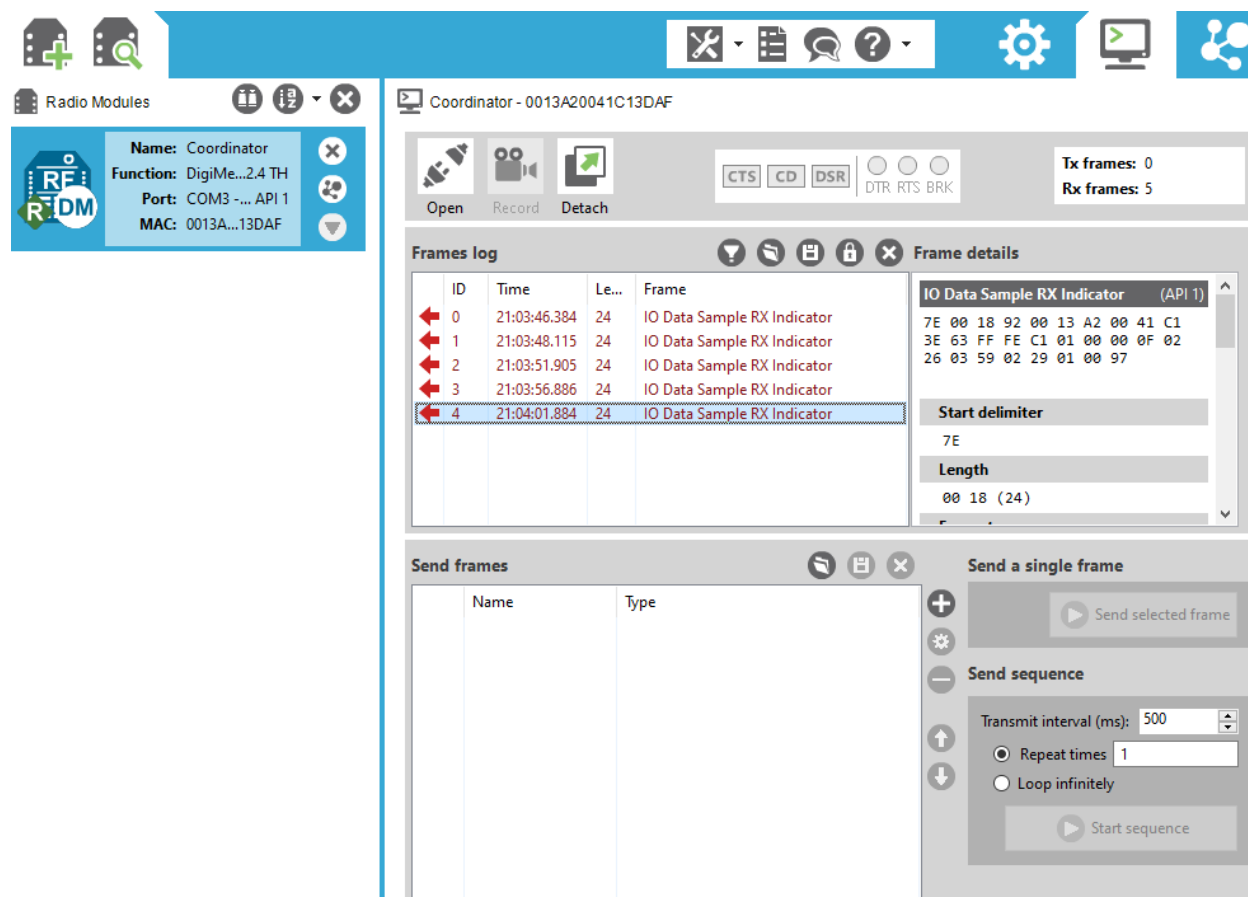


Figure 4.211 Frame Log Terminal

Every 5 seconds, a sample is sent from the wireless sensor node to the coordinator, and it can be seen in the frame log window. It contains all data received from the sensor node, most importantly, the analog sample value and source address. In the case seen in Figure 4.21, the 64-bit address of the node which sent the sample is 0013A20041C13E63, this can be used to identify which XBee sent the message, and therefore location on the farm. The data received in this frame

is seen as 0229, which is a HEX number, and has the value 553 when converted to base 10. As the maximum value of the 10-bit ADC is 1023, this means the voltage into pin 17 of the XBee which sent the sample is $553/1023 * 1.2V = 0.65V$.

4.5.6 Reading Analog Data

With XBee modules configured, and the coordinator receiving data frames in XCTU, a program should be written on the RPi to read this data. There are multiple potential ways to receive said data, the first being through the GPIO pins in the RPi board. The first attempt at reading serial data was done through serial ports, few packages need to be installed to the RPi before these pins can be accessed, which can be done by using the following command in the terminal:

```
“sudo apt-get install rpi.gpio”
```

The serial port used to access RPi GPIO pins is “/dev/ttyS0”, which must be used when trying to open the connection. The BR must be consistent through the design, it is 9600 in the XBee parameters, and is to be kept as 9600 in any python scripts. When using the first test script using GPIO pins with serial port /dev/ttyS0 as seen in Appendix E – XBee Read Test 1, the script could not detect the serial device or any incoming data, throwing errors on every run, so this method was scrapped.

The second test script to read analog data on the RPi involved using the XBee explorer USB board, connected directly to the RPi USB port. The aim of this second test script was to read any incoming frames every 5 seconds. As seen in Figure 4.22, the output of this script, when requesting data gives something that looks like the following:

“b’~\x00\x18\x92\x00\x13\xa2\x00A\xc1>c\xff\xfe\xc1\x01\x00\x00\x0f\x02\$\x02-\x02,\x01\xa7\x1c”

```

1 import serial
2 import RPi.GPIO as GPIO
3 import os, time
4
5 GPIO.setmode(GPIO.BOARD)
6 GPIO.setwarnings(False)
7
8 port = serial.Serial("/dev/ttyUSB0", baudrate = 9600, timeout = 1)
9 while True:
10     rcv = port.readline()
11     print(rcv)
12     time.sleep(5)

```

```

b'\x00\x1c'
b'~\x00\x18\x92\x00\x13\xa2\x00A\xc1>c\xff\xfe\xc1\x01\x00\x00\x0f\x02$\x02-\x02,\x01\xa7\x1c~\x00\x18\x92\x00\x13\xa2\x00A\xc1>c\xff\xfe\xc1\x01\x00\x00\x0f\x02&\x02,\x02+\x01\xa7\x1c'
b'~\x00\x18\x92\x00\x13\xa2\x00A\xc1>c\xff\xfe\xc1\x01\x00\x00\x0f\x02&\x02,\x02+\x01\xa7\x1c'
b'~\x00\x18\x92\x00\x13\xa2\x00A\xc1>c\xff\xfe\xc1\x01\x00\x00\x0f\x02&\x02,\x02+\x01\xa7\x1c'
b'~\x00\x18\x92\x00\x13\xa2\x00A\xc1>c\xff\xfe\xc1\x01\x00\x00\x0f\x02%\x02-\x02,\x01\xa7\x1b'
b'~\x00\x18\x92\x00\x13\xa2\x00A\xc1>c\xff\xfe\xc1\x01\x00\x00\x0f\x02$\x02-\x02,\x01\xa7\x1d~\x00\x18\x92\x00\x13\xa2\x00A\xc1>c\xff\xfe\xc1\x01\x00\x00\x0f\x02\x02,\x02+\x01\xa7\x1e'
b'~\x00\x18\x92\x00\x13\xa2\x00A\xc1>c\xff\xfe\xc1\x01\x00\x00\x0f\x02%\x02+\x02+\x01\xa7\x1e'
b'~\x00\x18\x92\x00\x13\xa2\x00A\xc1>c\xff\xfe\xc1\x01\x00\x00\x0f\x02%\x02+\x02+\x01\xa7\x1e'

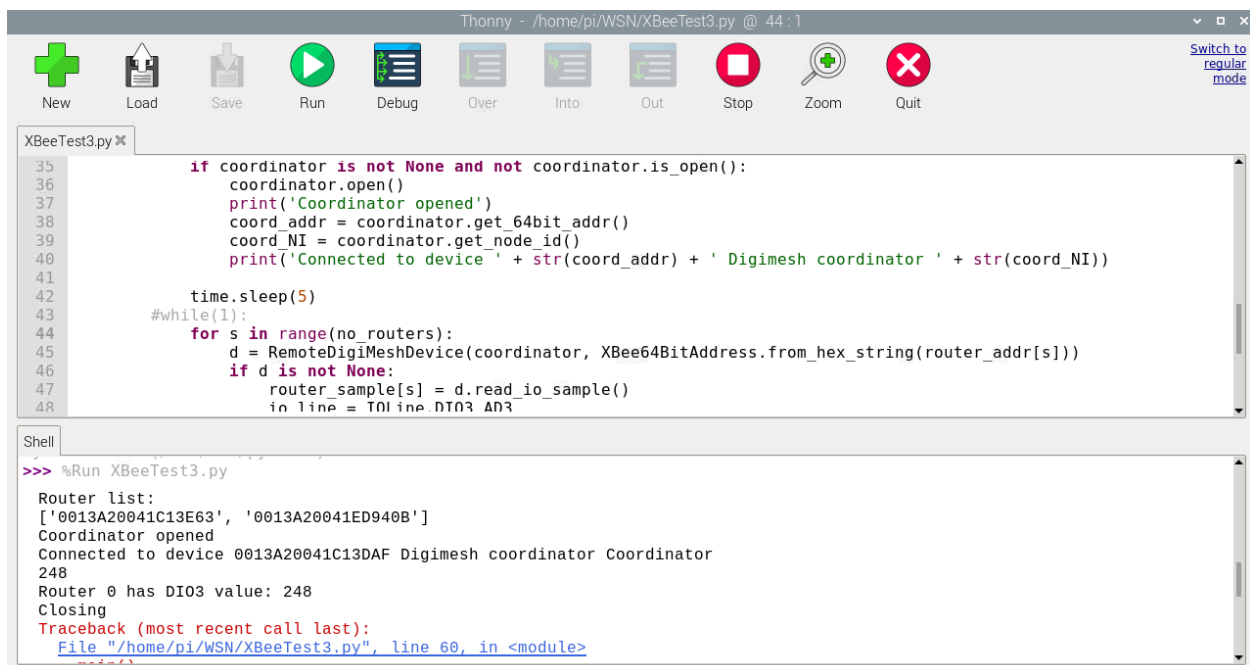
```

Figure 4.22 Reading serial data through USB port

It is observed that this data received is similar to that of the frame received in XCTU from an XBee node, with 0013A200, the SH of the sending XBee being displayed, however the SL is not being received well, all that is being received is “c1”, which is the first HEX data contained in SL. This is not helpful as it does not provide enough information about where the data frame is coming

from. It is also noted that all analog data signals were not included in this frame, hence rendering it useless.

A completely different approach is used for the third attempt at reading XBee data, taking advantage of the XBee API and the methods involved, the XBee USB explorer is still used to access the XBee serial data. All XBees used for sensor nodes have their unique 64-bit source addresses, found in XCTU, which are listed in a .txt file in the workspace folder. A list of router objects which the coordinator receives data from is created using these addresses and each router is queried for analog data being sent through the IO line in which data is being sent, in this case AD3 (Pin 17 ADC). The result of this method can be seen in Figure 4.23 below.



```
Thonny - /home/pi/WSN/XBeeTest3.py @ 44:1
New Load Save Run Debug Over Into Out Stop Zoom Quit Switch to regular mode

XBeeTest3.py
35     if coordinator is not None and not coordinator.is_open():
36         coordinator.open()
37         print('Coordinator opened')
38         coord_addr = coordinator.get_64bit_addr()
39         coord_NI = coordinator.get_node_id()
40         print('Connected to device ' + str(coord_addr) + ' Digimesh coordinator ' + str(coord_NI))
41
42     time.sleep(5)
43     #while(1):
44     for s in range(no_routers):
45         d = RemoteDigiMeshDevice(coordinator, XBee64BitAddress.from_hex_string(router_addr[s]))
46         if d is not None:
47             router_sample[s] = d.read_io_sample()
48             io_line = IO_line.DIO3_AD3

Shell
>>> %Run XBeeTest3.py
Router list:
['0013A20041C13E63', '0013A20041ED940B']
Coordinator opened
Connected to device 0013A20041C13DAF Digimesh coordinator Coordinator
248
Router 0 has DIO3 value: 248
Closing
Traceback (most recent call last):
  File "/home/pi/WSN/XBeeTest3.py", line 60, in <module>
    ...
```

Figure 4.23 Query Remote XBees for data

As seen in the shell, the router list is shown; in this case, two routers were used. The first router queried returned a sample value of 248, meaning the voltage into the XBee's pin 17 was $248/1023 * 1.2V = 0.291V$. This is a good sign as it shows that the remote XBee is sending data, and it is able to be received in a useful form which can be processed in the coordinator. Due to this method involving querying multiple XBees continuously, errors are run into when more than one module is used. This method also requires the Queried XBee to be on, in range and awake at the time data is queried, if any of these criteria are not met, the program will crash. As remote devices are likely to require sleep mode later in the design process, this method will not work for sleeping devices, and therefore is unlikely to be used. The full third test script which uses this method can be found in Appendix E: XBee Read Test 3.

A fourth script is made which can overcome problems encountered in the third script: accommodating for multiple sensor nodes and only running the code when data is received from the Router, rather than requesting it. This is done by adding a callback method, which can be found in the XBee Python API, to the coordinator. The callback definition will only be run any time the coordinator receives a message from any XBee, meaning it will not query for any data which does not exist. The program can then request the 64-bit source address of the XBee which sent the message, and any analog data on IO lines which have been defined. Figure 4.24 shows the result of this script.



```
Shell
>>> %Run XBeeDropbox.py
Router List: |
['0013A20041C13E63', '0013A20041ED940B']

Python 3.7.3 (/usr/bin/python3)
>>> %Run XBeeDropbox.py
Router List:
['0013A20041C13E63', '0013A20041ED940B']
Sample received from Node 1 - 221
```

Figure 4.24 Analog Result from test script

As seen in Figure 4.25, a sample is being received from Node 1, the “NI” parameter defined for the node in XCTU, which is used to identify where the sample came from. The analog sample received was 221, a 10-bit result. This means that the voltage output from the sensor was 0.259V. The received data is now received in a form which is easy to access and process further.

4.5.7 Data Processing

Due to data received from the XBee on the coordinator being in 10-bit format, it is not very useful to us. For useful data to be viewed, it should be converted into a form which is easy to read and provide information regarding the data type. In the case of a moisture sensor, 0-1023 is not very useful, and is even inversed with 0 being high moisture levels and 1023 being low levels. A useful way to read this information would be to convert the 10-bit data into a moisture percentage form, this is very easy to understand for the farmer and shall be applied for this sensor. If a pH sensor were to be used, this form would not be useful and a pH level of 0-14 should be read. To convert this moisture data to a percentage, we must first find the 10-bit reading when the sensor

is being used in a glass of water and take the reading, as seen in Figure 4.25, which will be used as a reference point for 100% moisture.

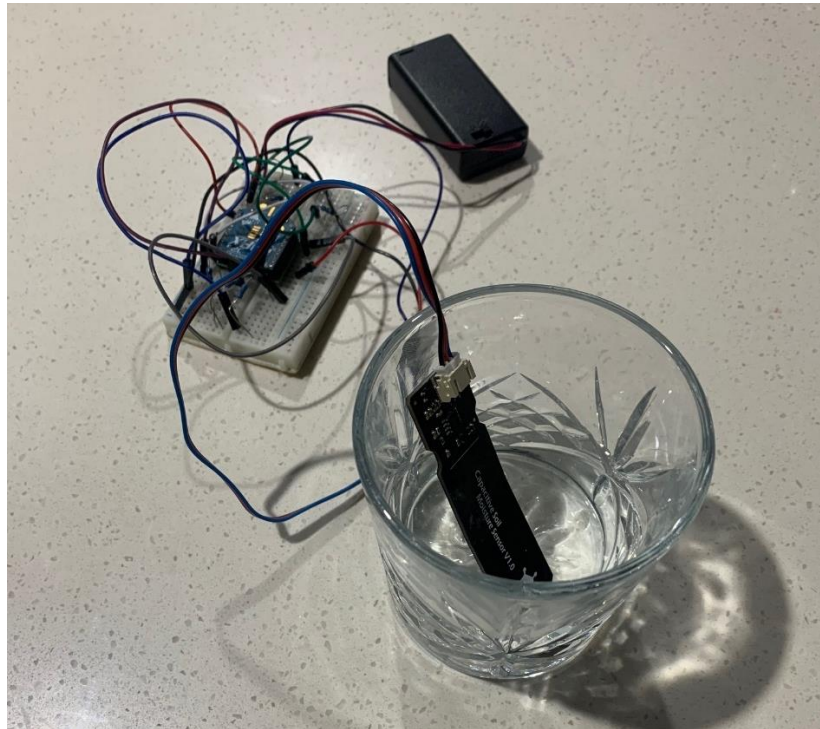


Figure 4.25 Wireless Sensor Node in Glass of water

The result received on the coordinator for 100% moisture was about 153, which will be used as the reference for maximum moisture level.

The same shall be done with atmospheric air, in which we assume that at the reading represents zero moisture, as seen in Figure 4.26.

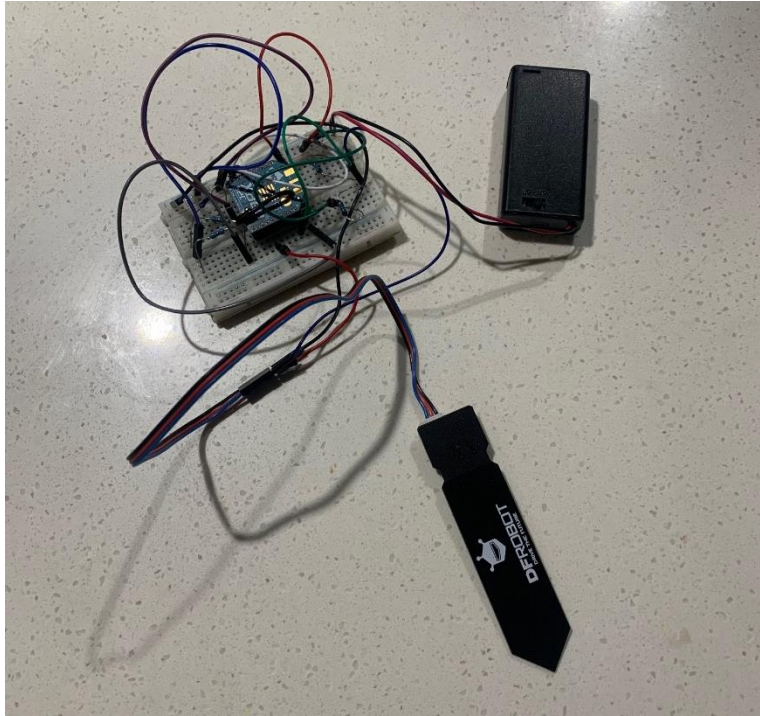


Figure 4.26 Moisture sensor in atmospheric air

The reading on the coordinator for the sensor in atmospheric air was about 295, which is used as the reference point for zero moisture level. With these two values, we can scale any reading between these numbers to a percentage using the python code definitions seen in Figure 4.27.

```
def _map(x, in_min, in_max, out_min, out_max):  
    return int((x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min)  
  
def getMoistureLevel(reading):  
    if reading >= MOISTURE_0:  
        return 0  
    if reading <= MOISTURE_100:  
        return 100  
    if reading < MOISTURE_0 and reading > MOISTURE_100:  
        Moisture_percentage = _map(reading, MOISTURE_0, MOISTURE_100, 0, 100)  
        return Moisture_percentage
```

Figure 4.27 Python definitions to process moisture readings

The first definition will scale a quantity, x , to fit the required data range. “in_min” and “in_max” will be the values which we found the sensor to discover, so will be 295 (min) and 153 (max) respectively. “out_min” and “out_max” represent the values to which the data shall be scaled to, in this case 0-100, as our data should be viewed as a percentage. If for example, a pH sensor was to be installed, these values would be 0 and 14 respectively.

The second definition will be called when a data sample is to be processed, in which it feeds the 10-bit data into the first definition and returns the moisture level in a percentage. This processed data being in a useful form, may now be used by the end-user to help make decisions. In order to make this data even more helpful, it would be good to show the time and date when this sample was received. There are many basic ways to request time and date from the RPi through python, which can be formatted as required. The method used requests the time and date from the device, and then formats it into “dd-mm-yyyy hh:mm:ss” format. The python library datetime must be imported to access this method.

4.6 Sensor Node Battery Life

Part of the design criteria is that the remote sensor nodes can remain in service without maintenance or human interference for as long as practicably possible. Being conservative, according to the datasheet the XBee can drain charge at a rate of 45mA, and the sensor at a rate of 5mA. When combined, the sensor node draws current at a rate of 50mA. Standard alkaline AA batteries have a capacity of ~2500mAh (Microbattery, n.d.), which means the sensor node has a battery life of $2500\text{mAh}/50\text{mA} = 50$ hours, until the farmer must change the battery. This is about 2 days, which is not good and far from optimal, therefore won't be desired by farmers. For this reason, sleep mode must be activated on all XBee endpoints and routers. According to the datasheet, XBees running on sleep mode use $\sim 1\mu\text{A}$, much lower than 45mA when on. This is negligible so essentially only the component drawing current is the sensor node at 5mA. This means the sensor draws ten times less power, and will survive for 500 hours, or 3 weeks. While this is a large improvement to 2 days, it is still not enough to be considered desirable to farmers as they would need to replace the batteries on every sensor node every 3 weeks.

▼ **Sleep Commands**
Configure Sleep Parameters

SM Sleep Mode	Async. Cyclic Sleep Pin Wake [5]		
SO Sleep Options	0	Bitfield	
SN Number of Cy...een ON_SLEEP	1		
SP Sleep Time	41EB00	* 10 ms	
ST Wake Time	2710	* 1 ms	
WH Wake Host Delay	0	* 1 ms	

Figure 4.28 Sleep Mode Settings

To enable sleep mode, the XBee router/endpoint must be connected to the PC and reconfigured in XCTU. The parameter SM is set to 5, as seen in Figure 4.28, which correlates to Cyclic sleep with pin wake-up, meaning the device wakes once every cycle, in which it must send data to the coordinator, then go back to sleep. The pin wake-up allows the device to be forced awake, which is very helpful if the module is ever reconfigured. SP (Sleep time) is set to 41EB00, which is HEX for 4,320,000, or 12 hours, means the module will wake twice per day. 2710 is HEX for 10,000, meaning the module will stay awake for 10 seconds per wake cycle.

To put the sensor to sleep along with the XBee module, the current into the sensor should only flow when the device is awake, therefore a current out of the XBee should be enabled when it wakes up and disabled when it is asleep. Pin 13 (ON_SLEEP) on the XBee module is used as an indicator for the sleep status, meaning it sends a current in the awake state. When connecting

the positive pin (V_{in}) on the sensor to pin 13 on the XBee as shown in Figure 4.29, the sensor falls asleep with the XBee device.

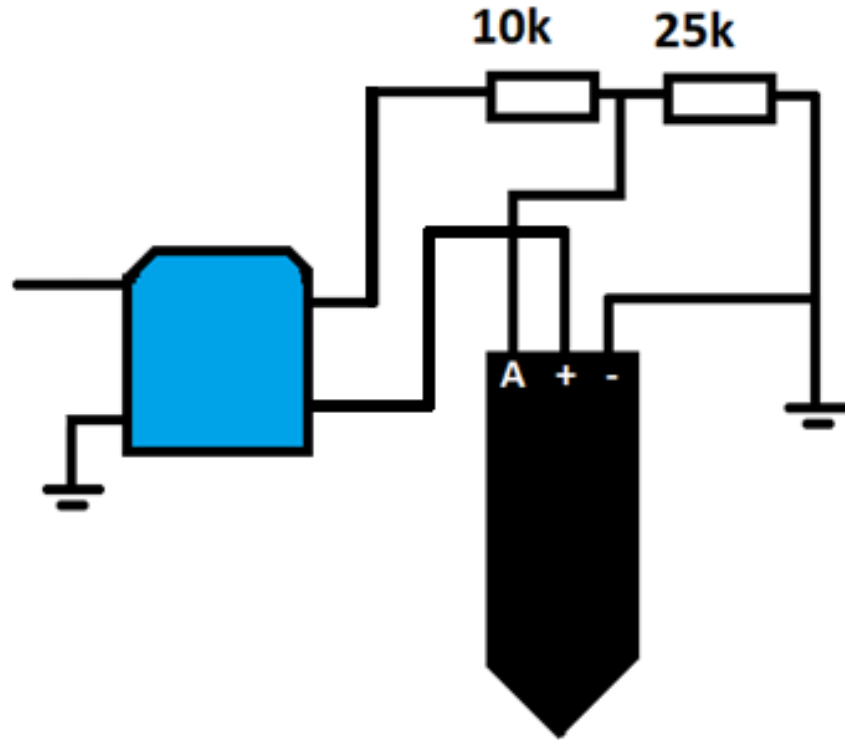


Figure 4.29 Wireless Sensor Node with Sleep Mode enabled

The sensor node will drain at 50mA for 20 seconds for day, and 1 μ A for the rest of the day. This means that the node drains 0.2778mAh per day of wake time, and 0.024mAh per day of sleep time, or 0.3018mAh total per day. Given that the battery has 2500mAh, this equates to 22.7 years of battery life. Alkaline batteries have a shelf life of 5-10 years, which is shorter than the duration it would take to run the battery dry and will be treated as the battery life of the wireless sensor nodes.

Chapter 5 Design Validation, Results and Discussion

This chapter will discuss how the designs in Chapter 4 are used in conjunction to form the functioning WSN and coordinator systems. The concepts will be related back to the design requirements and considerations, and ensure features are working as anticipated and robust.

5.1 Communications Validation

To ensure the correct readings are being sent and received, a test run is performed on one of the wireless sensors and sent across the network to the coordinator. Initially, the moisture sensor was left in the atmosphere, a reference point for zero moisture, and the reading taken, which read at 295 on the coordinator. The same sensor was then placed in a glass of water (100% moisture) as shown in Figure 5.1, which read about 153 on the coordinator, as shown in Figure 5.2.

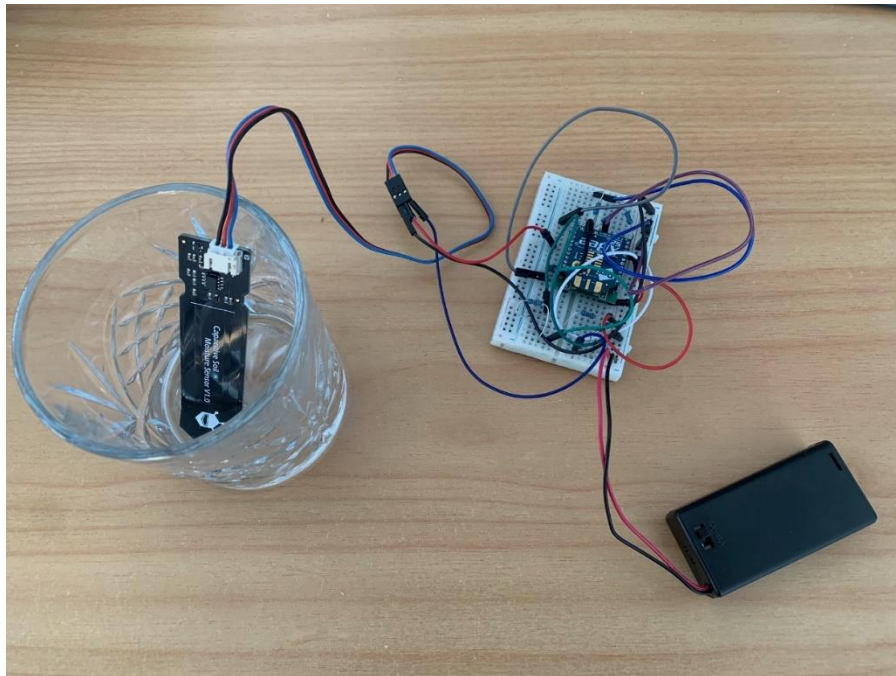


Figure 5.1 Capacitive Moisture sensor in Glass of water

Thonny - /home/pi/WSN/XBee4.py @ 31:1

New Load Save Run Debug Over Into Out Stop Zoom Quit [Switch to regular mode](#)

```

XBee4.py
17 print(ADDRESS_LIST)
18
19 def main():
20     coordinator = DigiMeshDevice(port,baudrate)
21
22     while True:
23         try:
24             coordinator.open()
25             def io_sample_callback(io_sample, remote_xbee, send_time):
26                 Node_ID = remote_xbee.get_parameter('NI').decode()
27                 Sample = io_sample.get_analog_value(IOLINE_IN)
28                 voltage_level = io_sample.get_analog_value(IOLINE_VOLTAGE)/1023
29                 Sample_adjusted = Sample/voltage_level

```

Shell

```

Router List:
['0013A20041C13E63', '0013A20041ED940B']
Sample received from Node 1 - 282
Sample received from Node 1 - 293
Sample received from Node 1 - 292
Sample received from Node 1 - 295
Sample received from Node 1 - 154
Sample received from Node 1 - 153
Sample received from Node 1 - 153
Sample received from Node 1 - 152

```

Figure 5.2 Wireless Sensor Moisture Readings - Raw data

This verifies that the coordinator is receiving correct data, with samples closer to ~153 being pure moisture, and samples being closer to ~295 being no moisture.

5.1.1 Analog Voltage Correction

During the design validation, it is noticed that the voltage of the batteries being used for the sensor nodes begin to drain, losing voltage. The voltage assumed through the design was 3V (2x AA batteries), however the voltage being supplied was ~2.5V. Figure 5.3 shows the voltage level after running at 100mA drain rate of an AA battery over time.

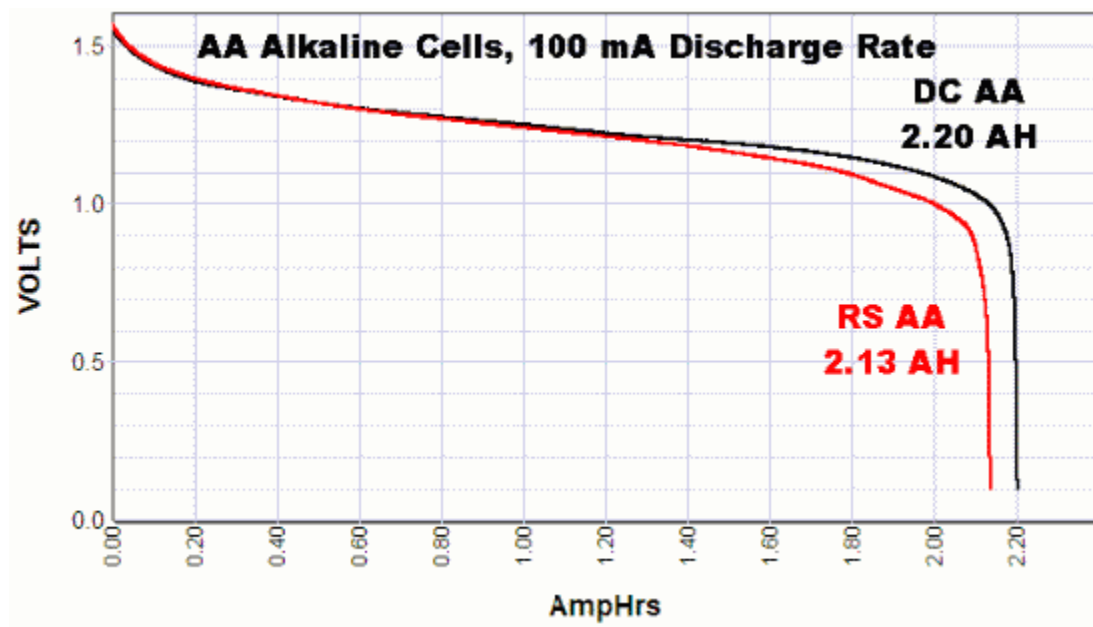
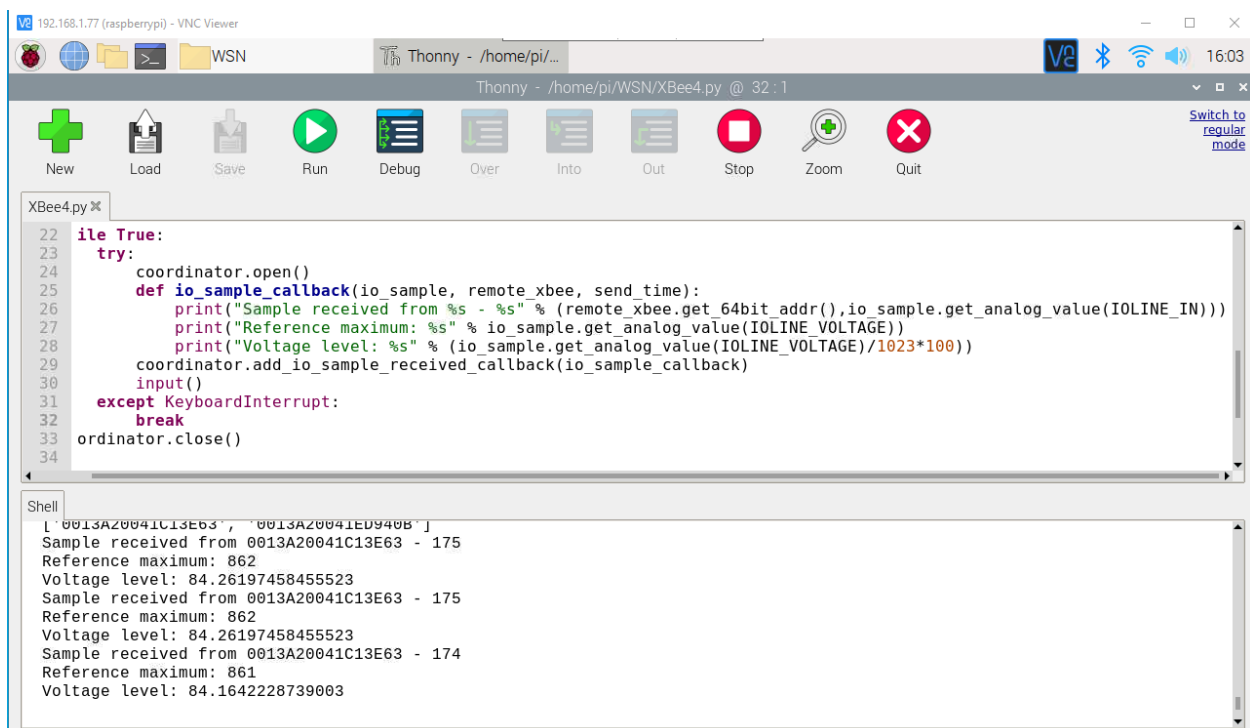


Figure 5.3 AA Battery Voltage across its life

Due to this low battery voltage, the sensor would be sending incorrect analog signals to the XBee, which can lead to incorrect readings and inappropriate or unnecessary actions taken due to this.

To counter this, the coordinator should read what voltage level the XBees on the sensor nodes are running at. A method in the python XBee API “get_power_level()”. It was attempted to implement this into the script as it could supply a reference to which a correction can be made, however, this method only returned either PowerLevel.HIGH or PowerLevel.LOW. This does not provide enough detailed information to make accurate voltage level corrections, so the battery voltage level must be sent through one of the analog pins on the XBee.

To get a reference voltage, another voltage divider was added and transmitted in GPIO pin 19, which is added to the coordinator script to read the IO line, as seen in Figure 5.4. This will once again give a 10-bit signal on a scale of 0-1023, 1023 being maximum battery voltage (3V).

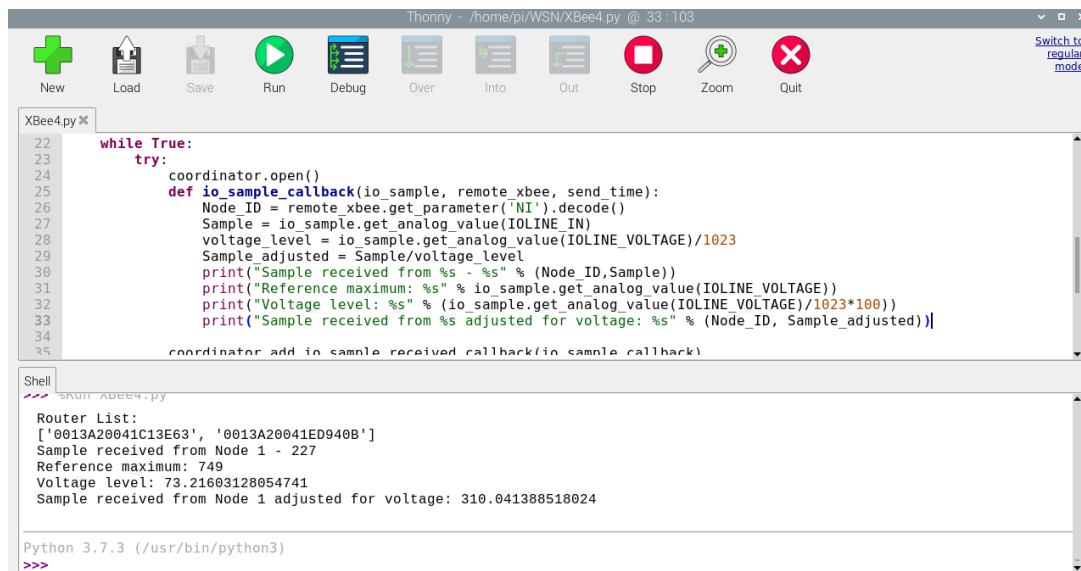


```
XBee4.py
22 if True:
23     try:
24         coordinator.open()
25         def io_sample_callback(io_sample, remote_xbee, send_time):
26             print("Sample received from %s - %s" % (remote_xbee.get_64bit_addr(), io_sample.get_analog_value(IOLINE_IN)))
27             print("Reference maximum: %s" % io_sample.get_analog_value(IOLINE_VOLTAGE))
28             print("Voltage level: %s" % (io_sample.get_analog_value(IOLINE_VOLTAGE)/1023*100))
29             coordinator.add_io_sample_received_callback(io_sample_callback)
30             input()
31         except KeyboardInterrupt:
32             break
33     coordinator.close()
34
```

```
Shell
['0013A20041C13E63', '0013A20041E0940B']
Sample received from 0013A20041C13E63 - 175
Reference maximum: 862
Voltage level: 84.26197458455523
Sample received from 0013A20041C13E63 - 175
Reference maximum: 862
Voltage level: 84.26197458455523
Sample received from 0013A20041C13E63 - 174
Reference maximum: 861
Voltage level: 84.1642228739003
```

Figure 5.4 Callback method when data received with voltage level reading

As seen in Figure 5.4, a 10-bit reading of 861 is being received from the remote XBee, meaning that the battery is running at 84.16% of its maximum voltage, or 2.52V. The sensor sample received from the remote XBee can then be divided by 0.8416 to provide an accurate sensor reading, regardless of battery life remaining, so long as the XBee is being supplied enough to run. As per the datasheet, the XBee requires a minimum of 2.1V to run, or 70% of battery life, after this, the battery will not send signals. As shown in figure 5.3, the battery will reach 70% voltage (1.05V in the figure) when 95% of its charge has been used. Figure 5.5 shows the modified method to adjust voltage level.



The screenshot shows the Thonny IDE interface. The top toolbar includes icons for New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, and Quit. The main editor window displays a Python script named 'XBee4py.py' with the following code:

```

22 while True:
23     try:
24         coordinator.open()
25         def io_sample_callback(io_sample, remote_xbee, send_time):
26             Node_ID = remote_xbee.get_parameter('NI').decode()
27             Sample = io_sample.get_analog_value(IOLINE_IN)
28             voltage_level = io_sample.get_analog_value(IOLINE_VOLTAGE)/1023
29             Sample_adjusted = Sample/voltage_level
30             print("Sample received from %s - %s" % (Node_ID, Sample))
31             print("Reference maximum: %s" % io_sample.get_analog_value(IOLINE_VOLTAGE))
32             print("Voltage level: %s" % (io_sample.get_analog_value(IOLINE_VOLTAGE)/1023*100))
33             print("Sample received from %s adjusted for voltage: %s" % (Node_ID, Sample_adjusted))
34
35         coordinator.add_io_sample_received_callback(io_sample_callback)

```

The bottom shell window shows the output of the script:

```

Router List:
['0013A20041C13E63', '0013A20041ED940B']
Sample received from Node 1 - 227
Reference maximum: 749
Voltage level: 73.21603128054741
Sample received from Node 1 adjusted for voltage: 310.041388518024

```

The shell prompt indicates the Python version is 3.7.3 and the path is /usr/bin/python3.

Figure 5.5 Data samples adjusted to account for low battery voltage

5.2 Data Interpretation

As the data voltage correction is complete, and accurate data readings are being obtained on the coordinator, this data must be interpreted for the user. As discussed in Chapter 4, reference values are used when converting the received 10-bit data into a percentage, however, these values found may not be accurate as they were not accounting for the voltage level correction. New values of 348 and 178 were used respectively for minimum and maximum moisture content, as shown in Figure 5.6.

Reference maximum: 865	Router List:
Voltage level: 84.55522971652005	['0013A20041C13E63', '0013A20041ED940B']
Sample received from Node 1 adjusted for voltage: 347.70173410404624	Sample received from Node 1 - 149
Sample received from Node 1 - 294	Reference maximum: 854
Reference maximum: 865	Voltage level: 83.47996089931574
Voltage level: 84.55522971652005	Sample received from Node 1 adjusted for voltage: 178.48594847775175
Sample received from Node 1 adjusted for voltage: 347.70173410404624	Sample received from Node 1 - 143
Sample received from Node 1 - 293	Reference maximum: 819
Reference maximum: 864	Voltage level: 80.05865102639295
Voltage level: 84.4574780058651	Sample received from Node 1 adjusted for voltage: 178.61904761904762
Sample received from Node 1 adjusted for voltage: 346.92013888888886	

Figure 5.6 Moisture Sensor in air (left) and glass of water (right)

These values will be used as the reference points for data processing hard coded into the python script for this sensor module, stating that 10-bit values of 348 is 0% moisture, and 178 is 100% moisture. To test this system, a wireless sensor node was used on a recently watered plant, as shown in Figure 5.7.

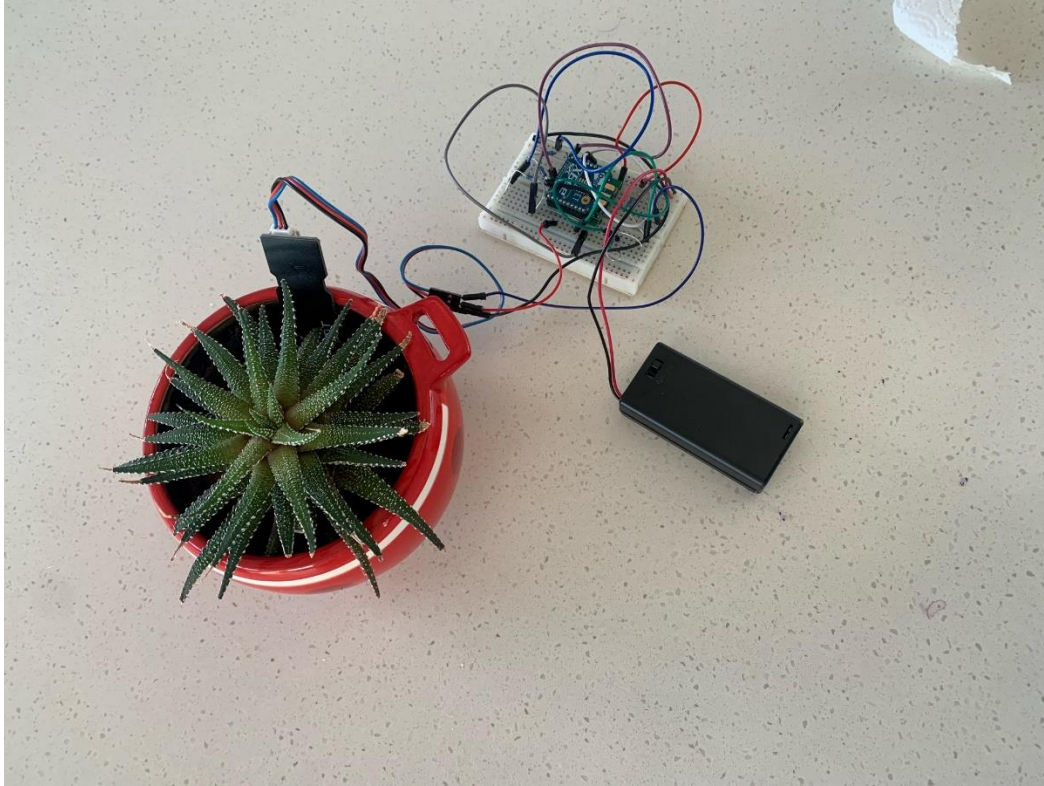
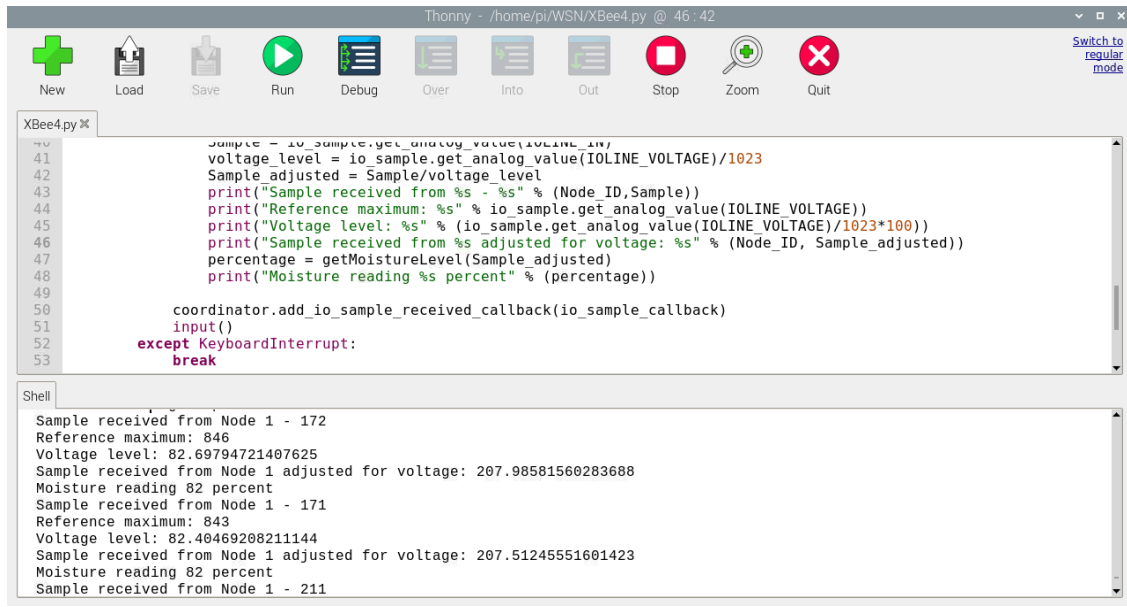


Figure 5.7 Wireless Moisture sensor used on recently watered plant

The coordinator received the results shown in Figure 5.8 with this wireless sensor node in place.



```
Thonny - /home/pi/WSN/XBee4.py @ 46:42
New Load Save Run Debug Over Into Out Stop Zoom Quit
XBee4.py
40 sample = io_sample.get_analog_value(IOLINE_IN)
41 voltage_level = io_sample.get_analog_value(IOLINE_VOLTAGE)/1023
42 Sample_adjusted = Sample/voltage_level
43 print("Sample received from %s - %s" % (Node_ID, Sample))
44 print("Reference maximum: %s" % io_sample.get_analog_value(IOLINE_VOLTAGE))
45 print("Voltage level: %s" % (io_sample.get_analog_value(IOLINE_VOLTAGE)/1023*100))
46 print("Sample received from %s adjusted for voltage: %s" % (Node_ID, Sample_adjusted))
47 percentage = getMoistureLevel(Sample_adjusted)
48 print("Moisture reading %s percent" % (percentage))
49
50 coordinator.add_io_sample_received_callback(io_sample_callback)
51 input()
52 except KeyboardInterrupt:
53     break
Shell
Sample received from Node 1 - 172
Reference maximum: 846
Voltage level: 82.69794721407625
Sample received from Node 1 adjusted for voltage: 207.98581560283688
Moisture reading 82 percent
Sample received from Node 1 - 171
Reference maximum: 843
Voltage level: 82.40469208211144
Sample received from Node 1 adjusted for voltage: 207.51245551601423
Moisture reading 82 percent
Sample received from Node 1 - 211
```

Figure 5.8 Moisture readings for recently watered plant

The moisture reading was 82 percent for the recently watered plant, accurate to what we would expect. When holding the moisture sensor with hands, the moisture reading is 20 percent, also what we would expect.

5.3 Cost Analysis

5.3.1 Wireless Sensor Node Cost

The wireless sensor node consists of the following:

1. Sensors

2. XBee
3. 2x AA batteries
4. Breadboard
5. 4x Resistors
6. Jumper Leads

For the design used in this project, a simple capacitive moisture sensor was used, however, many different sensors could be applied to these devices, analog or digital. Up to four different sensors may be used as there are 4 pins on the XBee device which can be configured accordingly, however, as per this design, one of those pins is being used to transmit a voltage from the batteries to make reading corrections. Table 5.1 shows the price of the parts used in the node at the time of writing. Refer to Appendix G for detailed part list.

Table 5.1 Wireless Sensor Node Cost Analysis

Part	Price (NZD)
Capacitive moisture sensor	\$12.10
XBee S2C	\$29.10
Breadboard	\$9.50
2x AA batteries	\$3.00
AA (x2) battery holder	\$3.00
Total	\$56.70

Each wireless sensor node costs \$56.70 NZD at the time of writing. This is considered quite cheap, for the complete system, which will benefit farmers long-term, providing money savings with more optimized decision making and efficiency. The capacitive moisture sensor was only \$12.10, while other sensors, such as pH sensors for ~\$20, can be added to the system for low costs.

5.3.2 Coordinator Cost

Table 5.2 shows the price of the parts used in the coordinator at the time of writing. Only one coordinator is required per WSN. Refer to Appendix G for detailed part list.

Table 5.2 Coordinator Cost Analysis

Part	Price (NZD)
Raspberry Pi 3B	\$84.90
XBee Explorer USB	\$50.26
XBee S2C	\$29.10
Total	\$164.26

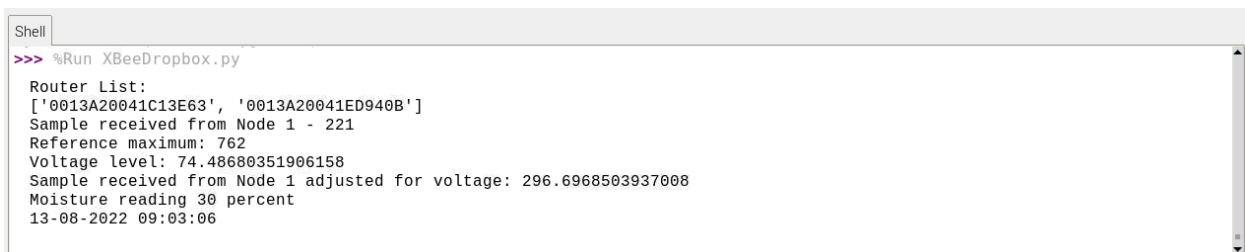
At a total cost of \$164.26 NZD for the coordinator, this is also considered cheap for this kind of system, only one of these is required in the network. Considering its scalability for future projects, with an OS and the ability to easily modify python scripts with a bit of knowledge, making changes to the network, including sensors or feedback loops, should be relatively simple to implement. Other options could be considered to reduce this price; however, it is unlikely to find

replacements for these parts which maintain functionality and unnecessary as it is already cheap for the benefits it provides. This is likely to be a big factor in what determines whether the solution will be chosen to be implemented into farms by farmers.

5.4 Final Design

5.4.1 Data Readings

Once all parts of the python script are complete, they can be implemented together to achieve the final goal of uploading useful sensor data from wireless sensor nodes to a cloud server. This is done by making a combined string in the python script which includes the moisture level in percentage form, the remaining battery charge, and the time of the reading. This string is then saved into a local text file under the name of the node which it was received from, or the “NI” parameter of the XBee. Figure 5.9 shows all the information gathered when it receives a data packet from the remote XBee.



```
Shell
>>> %Run XBeeDropbox.py
Router List:
['0013A20041C13E63', '0013A20041ED940B']
Sample received from Node 1 - 221
Reference maximum: 762
Voltage level: 74.48680351906158
Sample received from Node 1 adjusted for voltage: 296.6968503937008
Moisture reading 30 percent
13-08-2022 09:03:06
```

Figure 5.9 Information from remote XBee

With this data received, and the text file created on the local machine (RPI), the data can easily be uploaded to the cloud server, in this case dropbox, using their API. This essentially means the data is online and accessible through any computer or mobile device, by logging into their account. Figure 5.10 shows an example of the uploaded text file on the dropbox server.

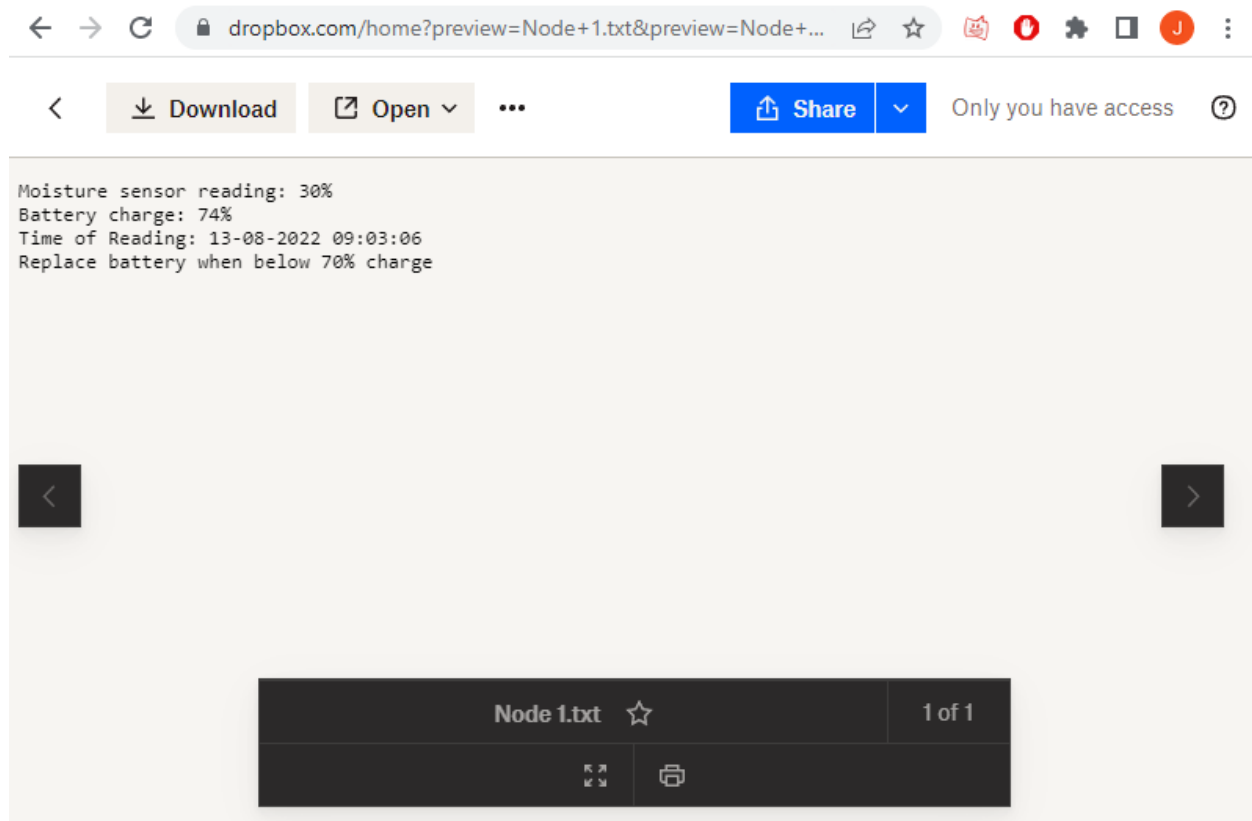


Figure 5.10 Uploaded text file onto dropbox server

Each wireless sensor node will have its own unique “NI” identifier as defined in XCTU, under which the node files are named. The file shown in Figure 5.10 above is named Node1.txt, showing where the data was received from.

With the remote wireless sensor nodes complete, a schematic is shown in Figure 5.11, displaying the wiring setup of the node and the appropriate pins.

5.4.2 Wireless Sensor Node Final Schematic

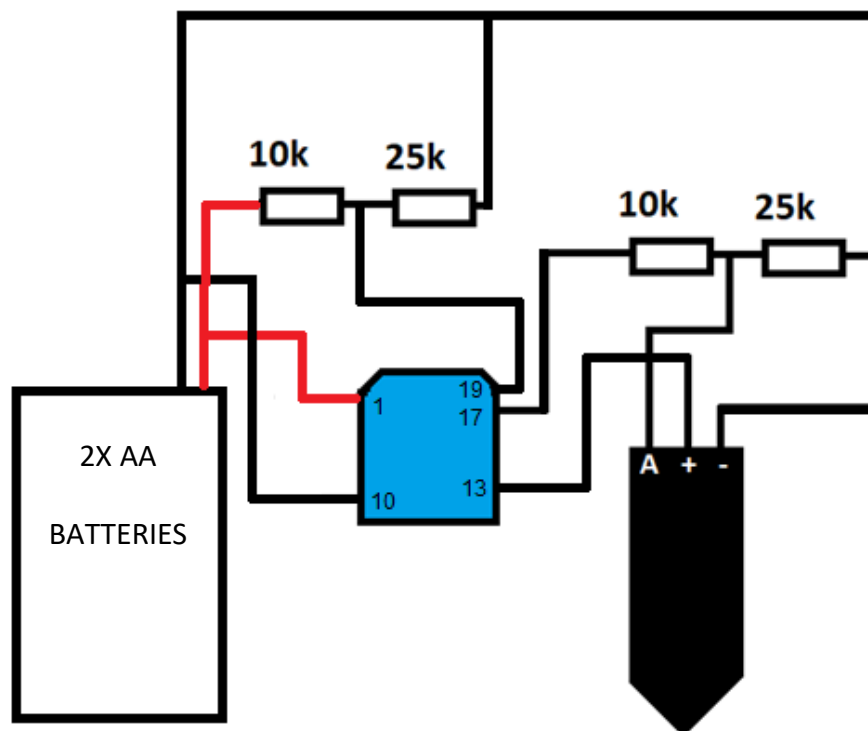


Figure 5.11 Wireless Sensor Node Schematic

As shown in Figure 5.11 above, pins 1 and 10 of the XBee in blue are connected directly to the 2x AA batteries, providing up to 3V of power. Pin 19 is also connected to the battery directly, with a

voltage divider so that it can transmit the battery voltage level for the relevant data adjustments to be made. Pin 17 is connected to the analog output of the sensor, with a voltage divider to drop the voltage so that it fits the appropriate XBee analog reference voltage of 1.2V. Pin 13 is the sleep mode pin which is connected to the positive end of the sensor and will only send a current when the device is awake, greatly reducing charge consumption. As discussed in chapter 4, the power consumption of the device is so low, that it is essentially just the shelf life of the battery – typically 5-10 years. Due to time constraints, this cannot be validated.

5.4.3 Moisture Readings

To verify the readings of the system, sleep mode was disabled on an XBee and was tested in three scenarios: atmospheric air, in a freshly watered plant, and in a glass of water. The sampling rate was set to 5 seconds and 4 readings of each scenario were taken with the final copy of the python program. Tables 5.3, 5.4 and 5.5 show the output of these runs respectively.

Table 5.3 Verification of system applied to atmospheric air

	Reading 1	Reading 2	Reading 3	Reading 4
Moisture Sensor	18%	19%	19%	20%
reading	(317)	(315)	(315)	(314)
Battery Charge	82.8%	82.7%	82.5%	82.5%
reading				

Time of reading	04-08-2022 11:43:22	04-08-2022 11:43:27	04-08-2022 11:43:32	04-08-2022 11:43:37
------------------------	------------------------	------------------------	------------------------	------------------------

A fresh set of AA batteries was used when measuring the moisture level in a glass of water to ensure the functionality of the battery charge readings, and hence analog data corrections.

Table 5.4 Verification of system applied to glass of water

	Reading 1	Reading 2	Reading 3	Reading 4
Moisture Sensor	100%	100%	100%	100%
reading	(157)	(162)	(162)	(162)
Battery Charge	94.8%	95.4%	95.1%	95.4%
reading				
Time of reading	04-08-2022 11:55:29	04-08-2022 11:55:33	04-08-2022 11:55:38	04-08-2022 11:55:43

A third set of AA batteries were used to test the final readings, the system applied to a freshly watered plant.

Table 5.5 Verification of system applied to freshly watered plant

	Reading 1	Reading 2	Reading 3	Reading 4
Moisture Sensor	74%	75%	74%	74%
reading	(221)	(219)	(222)	(222)

Battery Charge reading	81.5%	81.2%	82.1%	82.4%
Time of reading	04-08-2022 12:11:00	04-08-2022 12:11:05	04-08-2022 12:11:10	04-08-2022 12:11:15

The results for each of these cases clearly showcases that the wireless sensor nodes function exactly as they are intended to. The moisture readings are low when exposed to atmospheric air, they are 100% when exposed to water, and ~74% when exposed to soil. It is noticed that the readings for atmospheric air were higher than those which the device was calibrated to, the 10-bit reading was 317 whereas the calibration was set to 348. This must mean that there was a lower moisture content in the air at the time when these values were calibrated. This is unlikely to affect the stability of the system however, as the sensors will not be exposed to air humidity when used practically. These values are however close, verifying that the readings are functional. When the batteries were changed between runs, the battery level reading also changed as intended, and the results still appear to be accurate. It also seems like the battery charge reading has roughly a 1% fluctuation, or ~0.03V. This would be very difficult to diagnose or correct and is negligible so it will be ignored. All the data from these tables was collected from the cloud server text file.

5.4.4 Range Testing

According to the XBee S2C datasheet in Appendix A, the range of data transmission is different for indoor and outdoor applications. The datasheet states the indoor range of the XBee is 60m and the outdoor range is 1200m. This may however be different to the device which has been made as batteries may not be running at the full voltage, and current may be lower than optimal, potentially lowering the range of the device. Firstly, the device was tested for indoor range. The coordinator was placed inside with a Wi-Fi connection and a Wireless Sensor Node was slowly walked down the street until signals were no longer being received. Figure 5.12 shows the range when the coordinator was placed indoors.

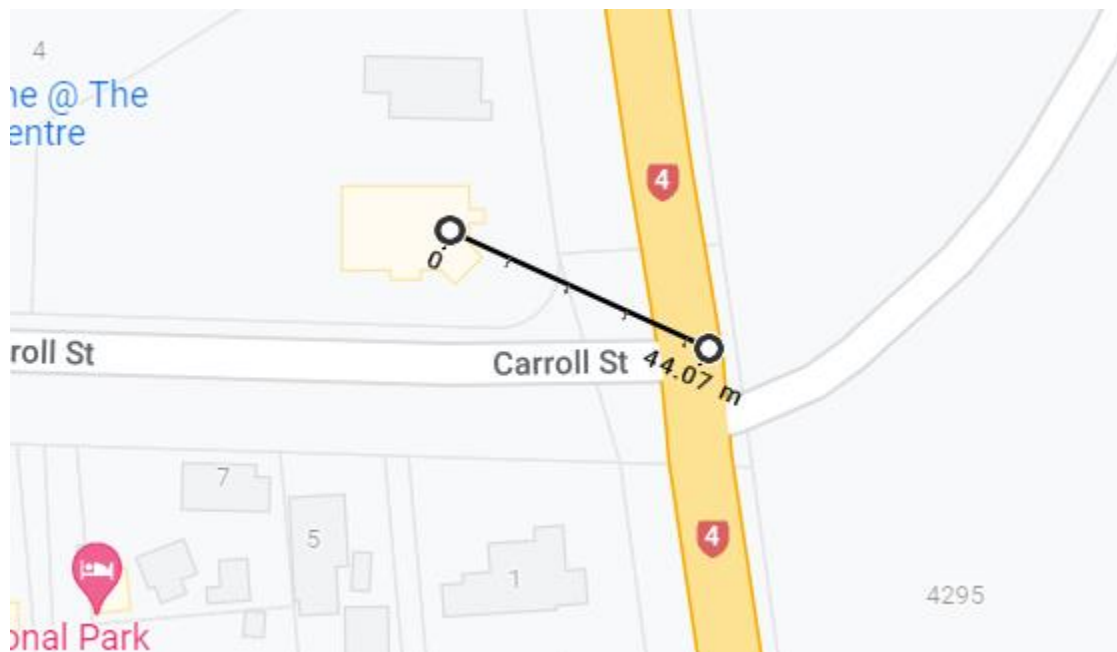


Figure 5.12 Indoors XBee Range (Google, 2022)

As seen in Figure 5.12, the indoor range is measuring ~44m, less than the 60m the devices are rated for in the spec sheet. The same test is done for line of sight (outdoor), Figure 5.13 shows the range.

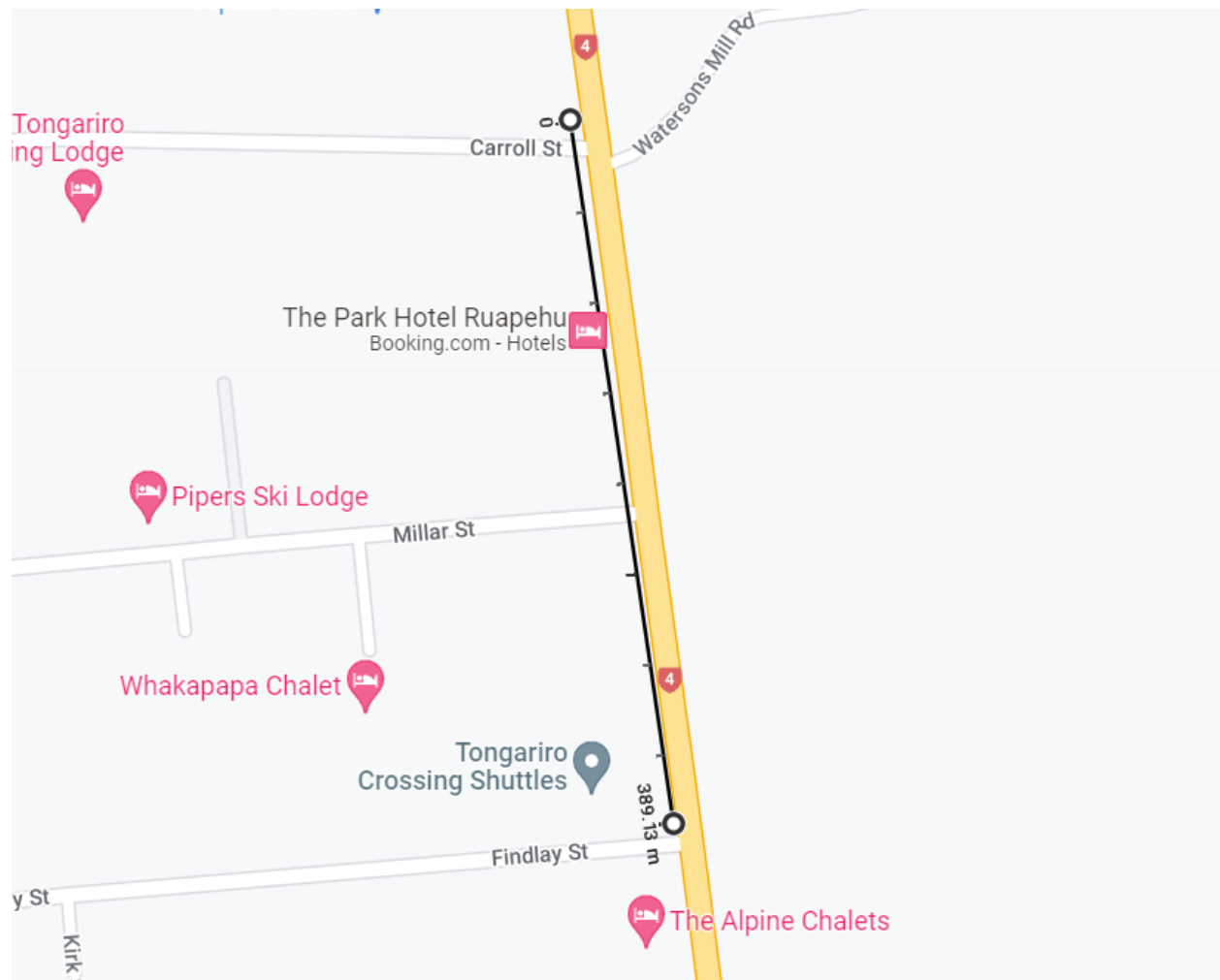


Figure 5.13 Line of Sight XBee Range (Google, 2022)

As seen in Figure 5.13, the range is also less than the datasheet states, considerably. This means in their current state, the devices are potentially not able to reach their full power levels. The most likely reason for this is lack of current because the datasheet states the devices run all the way down to 2.1V, while the batteries used in this test were providing 2.8V. With a refined circuit

board made to provide the device continuously with enough current, these ranges should increase to what they have been rated for. However, this is not a limitation to the design, as XBees can be programmed as routers which act as an intermediate between the wireless sensor node and the coordinator. These can be quite simply set up as they receive data and can be programmed to retransmit, allowing up to a maximum number of hops (depending on how far the devices are from each other).

Chapter 6

Conclusion

6.1 Objectives and Achievements

It is believed that a system design which can be implemented into farms in New Zealand and meets the requirements discussed in Chapter 3 has largely been completed with the framework setup complete. The design and functionality of the system can be used as a proof of concept that I4.0 can be implemented to farms in New Zealand providing data based on wireless sensors throughout the farm and results easily accessible and able to be monitored on the farmers computer or phone. The WSN has been developed from scratch and is now able to take accurate wireless readings, and able to upload them to a cloud server for access anywhere. Now that the technical side of the design has been finished, including setting up the electrical components, programming the devices and successfully sending and receiving long distance wireless signals, there is space for refinement and making them accessible in farms, such as developing a physical structure for the nodes, essentially allowing for outdoor implementation and usage.

A simple moisture level sensor was used throughout the project, as a base to build off, and prove that the system functions as intended, allowing for many future developments as many types of sensors able to be implemented, and data processed using the same methods as discussed in Chapter 4. There is a very large scope for the future of the project, as the system can now be scaled up in many ways, including sensor improvements, automated feedback loops and AI. With these features added to the completed WSN structure and programs developed through this project, it should be simple to implement features as they are needed.

6.2 Critical Results and Inferences

Throughout the literature review conducted in this project, it is noticed that I4.0 is a big step up from technologies used in I3.0, which includes the implementation of computers and simple automated tasks, whereas I4.0 vastly integrates itself into the system to the point that AI can be implemented into production through continuous feeds of data to greatly improve the levels of automation and increase productivity while finding weaknesses or areas of potential improvement. Typically, farming wouldn't be considered an industry which is highly integrated with such technology, but the research conducted shows that there can be large benefits to it even in areas where it isn't expected.

The design of the system is very robust, providing continuous feedback and information about the farm for years of its life, with a python script which only runs when called for, meaning the chance of crashing is extremely unlikely. The only big reason for potential crashing would be if the coordinator does not have Internet access when it tries to upload data, however this can easily be noticed as timestamps appear on readings, and if it has been more than 12 hours (or a defined period) then there must have been an Internet disconnection or outage. The exact same system can be applied to different sensors as they are added to the wireless sensor node by following the steps taken between sections 4.5.5 and 4.5.8 followed by adjustments taken in sections 5.1.2 and 5.2.

At a price of ~\$56.70 per wireless sensor node and a \$164 coordinator, the goal of making a solution to the wireless sensor network applied in farms which encourages farmers to implement can be considered successful. With these low prices and ease of installation and implementation, without the need to pay ongoing costs, it is believed that the system meets the specifications established in Chapter 3. The wireless sensor nodes require minimal upkeep or interaction and can be left for very long periods of time before requiring any sort of maintenance. The devices are sending data across ~400m through the validation, but it is likely this range can be easily extended as discussed in Chapter 5. Data which is collected from sensors across the farm send data back to the coordinator which then processes and uploads this information to a cloud server which can be easily accessed from smartphones or computers to aid in decision making or monitoring of farms. All the specifications are considered to be met and therefore a success.

6.3 Future Work

There is very large scope for the future of this project, with space for improvements and scalability on many aspects and components included in the design, which encourages farmers to implement these designs to their farms, and benefit from them.

Firstly, the physical design structure of the wireless sensor node needs to be made for the electronic components, to protect them from the wind and rain, allowing these devices to be applied in open environments, rather than just sheltered areas. This will be fairly simple as a 3D CAD model can be designed to equip the sensor node, and then manufactured with waterproofing, such as through FDM 3D printing with thermoplastics or CNC machining a mould to make the casing with.

Secondly, more sensors could be implemented to give more relevant information to what the farmer is trying to achieve with these devices. Examples of sensor types which can be applied to farms include optical sensors, which can determine clay or organic matter content. Electrochemical sensors could potentially be deployed to test soil for nutrients, specifically pH level, these could be very easily added to the XBee devices as pH data is analog and applied the same way as moisture sensors used in this project. Small low power cameras could potentially

be employed to obtain a physical view of the farm and provide imagery of leaf health, ripeness, or cattle.

Implementing feedback loops which make decisions based on received data from the WSN completes the CPS, and could potentially have large benefits for farmers, as it can save them resources, optimize time usage and improve quality farm output. Farms often cover large areas and having automated systems can save farmers time of having to get around their farm to all the areas which need attention.

AI is a potential concept which can be implemented to this system, often used in other industries which use I4.0 ideas, it could be suitable for this application. AI would greatly improve decision making (and hence, automation if implemented) especially when used with multiple sensors in conjunction with one another, AI would figure out how to get the highest quality product, and/or get that product in the quickest time possible. AI will be tricky to implement into this design however, as they require training periods based on the data being fed into them. They are trained by a few potential methods including:

- Based on historical data. This method of training AI won't work because we do not have databases with sensor data and product output, therefore AI cannot find or optimize patterns to relate the two.

- Real-time training. AI will learn in real-time as farming is done and product output is achieved. This is tricky because it takes a very long time for each system to produce an output (i.e. crops, fruit, cattle) for the AI to relate quality and time to the inputs (sensor data and decisions made based on them).

Implementing a GUI would be necessary with these proposed features. The complexity of the system will get high fast, and in order to maintain usability, there will need to be a level of human interaction with the program. As the number of sensor types is increased, with each type of data requiring different processing, the scalability of the system is limited unless a GUI is used. Features which may potentially be used in the GUI to improve the level of usability may include:

- Activating and deactivating each sensor type for every node – selecting which pins each different sensor is connected to. Data processing would then be allocating accordingly to these settings.
- Location data – with a GUI, each sensor node may be allocated with its own metadata, which can interpret the source address to include location data, names, images, labels, and notes. These pieces of information can be uploaded to the cloud-server too which will make it easier for the user to know which sensor is which and increase usability.
- Data calibration – in order to ensure that all sensor readings are accurate, a calibration mode can be created. The GUI provides options to calibrate and provide instructions to place the sensor node in each extreme condition (in moisture sensor case, glass of water

and air-conditioned air). The program then automatically scales this data based on the sensor readings using the method described in Chapters 4.5.7 and 5.2.

- Parameter adjustment – the farmer may adjust certain parameters of the sensor network such as the reading times – if the farmer requires more accurate data than the default two readings per day, they may increase the frequency. With feedback loops and automation in place, farmers may adjust thresholds of data before the corrective action is taken or adjust the level of the corrective action.

References

- Ahmed, N., De, D., & Hussain, I. (2018). Internet of Things (IoT) for Smart Precision Agriculture and Farming in Rural Areas.
- Akyildiz, I., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002). Wireless sensor networks: a survey.
- Alam, N., Balaei, A. T., & Dempster, A. G. (2013). Relative Positioning Enhancement in VANETs: A Tight Integration Approach.
- Arduino CC. (2017, July). *XCTU and XBee shield communication*. Retrieved from Using Arduino/Networking, Protocols and Devices: <https://forum.arduino.cc/t/xctu-and-xbee-shield-communication/469367>
- ASM Technologies. (n.d.). *Introduction to Wireless and Telecommunication*. Retrieved from ASM Technologies Engineering Innovation: <https://www.asmltd.com/introduction-wireless-telecommunication/#:~:text=Wireless%20communication%20technology%20transmits%20information,and%204G%20networks%2C%20and%20Bluetooth>
- Ayaz, M., Ammad-Uddin, M., Sharif, Z., Mansour, A., & Aggoune, E.-H. M. (2019). Internet-of-Things (IoT)-Based Smart Agriculture: Toward Making the Fields Talk.

Berkley. (n.d.). *Cyber-Physical Systems*. Retrieved from Ptolemy Berkley Cyber-Physical Systems - A Concept Map: <https://ptolemy.berkeley.edu/projects/cps/>

Bhrugubanda, M. (2015). A Review on Applications of Cyber Physical Systems.

Bose-Munde, A., & Finus, F. (2019, May 3). *What is Smart Factory? Definition, exmaples & industry 4.0 technologies*. Retrieved from ETMM: <https://www.etmm-online.com/what-is-smart-factory-definition-examples-industry-40-technologies-a-825861/#:~:text=Another%20prime%20example%20of%20Smart,to%20follow%20a%20specific%20sequence>

Burke, R., Laaper, S., Hartigan, M., & Sniderman, B. (2017, Aug 31). *The smart factory*. Retrieved from Deloitte Insights: <https://www2.deloitte.com/us/en/insights/focus/industry-4-0/smart-factory-connected-manufacturing.html>

Chen, H. (2017). Applications of Cyber-Physical System: A Literature Review.

DelftStack. (2021, March 12). *Overwrite a File in Python*. Retrieved from DelftStack: [https://www.delftstack.com/howto/python/python-overwrite-file/#:~:text=Overwrite%20a%20File%20in%20Python%20Using%20the%20file.,-truncate\(\)%20Method&text=truncate\(\)%20method.,truncate\(\)%20method.](https://www.delftstack.com/howto/python/python-overwrite-file/#:~:text=Overwrite%20a%20File%20in%20Python%20Using%20the%20file.,-truncate()%20Method&text=truncate()%20method.,truncate()%20method.)

Digi. (2018, August 23). *Digital and Analog sampling using XBee radios*. Retrieved from Digi International: <https://www.digi.com/support/knowledge-base/digital-and-analog-sampling-using-xbee-radios>

Dropbox. (2015, June 14). *Dropbox for Python Developers*. Retrieved from Dropbox:
<https://www.dropbox.com/developers/documentation/python>

Electrical Engineering. (2014, October 14). *When will the AA battery voltage drop?* Retrieved from
Stack Exchange: <https://electronics.stackexchange.com/questions/134143/when-will-the-aa-battery-voltage-drop>

Electronic Wings. (2018). *XBee Module Sensors & Modules*. Retrieved from Electronic Wings:
<https://www.electronicwings.com/sensors-modules/xbee-module>

Faludi, R. (2017, November 6). *Introducing the Official Digi XBee Python Library*. Retrieved from
Digi International: <https://www.digi.com/blog/post/2017/introducing-the-official-digi-xbee-python-library>

Fancom. (n.d.). *Smart Farming for superior farm conditions*. Retrieved from Fancom:
<https://www.fancom.com/smart-farming>

Fang, X., Misra, S., Xue, G., & Yang, D. (2012). Smart Grid - The New and Improved Power Grid: A
Survey.

Farooq, M. S., Riaz, S., Abid, A., Abid, K., & Naeem, M. A. (2019). A Survey on the Role of IoT in
Agriculture for the Implementation of Smart Farming.

Ferdoush, S., & Li, X. (2014). Wireless Sensor Network System Design Using Raspberry Pi and
Arduino for Environmental Monitoring Applications.

- Google. (2022). *Google Maps*. Retrieved from Google Maps: <https://www.google.com/maps/@-36.8774253,174.8444205,19.01z>
- Gunes, V., Peter, S., Givargis, T., & Vahid, F. (2014). A Survey on Concepts, Applications, and Challenges in Cyber-Physical Systems.
- Guo, P., Dusadeerungsikul, P. O., & Nof, S. Y. (2018). Agricultural cyber physical system collaboration for greenhouses stress management.
- Haque, S. A., Aziz, S. M., & Rahman, M. (2014). Review of Cyber-Physical System in Healthcare.
- Jain, R. (2019, April 2). *How to Interface XBee Module with Raspberry Pi*. Retrieved from Circuit Digest: <https://circuitdigest.com/microcontroller-projects/raspberry-pi-xbee-module-interfacing#:~:text=So%2C%20before%20using%20the%20XBee,the%20laptop%20using%20USB%20cable.>
- Kim, S., & Park, S. (2017). CPS(Cyber Physical System) based Manufacturing System Optimization.
- Lee, I., Sokolsky, O., Chen, S., Hatcliff, J., Jee, E., Kim, B., . . . Venkatasubramanian, K. K. (2012). Challenges and research directions in medical cyber-physical systems.
- Lin, J., Yu, W., Zhang Nan, Yang, X., Zhang Hanlin, & Zhao, W. (2017). A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications.
- Malmsten, P., Rapp, G., Brian, Brackert, C., Synderman, A., Sangalli, M., . . . Walker, D. (2018, April 4). *Python tools for working with XBee radios*. Retrieved from PyPI: <https://pypi.org/project/XBee/>

- McFadden, C. (2019, Nov 18). *How Exactly Does Wi-Fi Work?* Retrieved from Interesting Engineering: <https://interestingengineering.com/innovation/how-exactly-does-wi-fi-work>
- Microbattery. (n.d.). *AA Battery: Everything You Need To Know About The AA Battery*. Retrieved from Microbattery: <https://www.microbattery.com/blog/post/battery-bios:-everything-you-need-to-know-about-the-aa-battery/>
- Mitchell, B. (2020, November 5). *What is the Range of a Typical Wi-Fi Network?* Retrieved from Lifewire Tech For Humans: <https://www.lifewire.com/range-of-typical-wifi-network-816564#:~:text=A%20general%20rule%20of%20thumb,indoors%20and%20300%20feet%20outdoors.>
- Muhuri, P. K., Shukla, A. K., & Abraham, A. (2019). Industry 4.0: A bibliometric analysis and detailed overview.
- Nana, G. (2010, March 11). *Industrial sectors*. Retrieved from Te Ara - the Encyclopedia of New Zealand: <http://www.TeAra.govt.nz/en/industrial-sectors>
- Nikhade, S. G. (2015). Wireless Sensor Network System using Raspberry Pi and Zigbee for Environmental Monitoring Applications.
- Ojha, T., Misra, S., & Raghuwasnshi, N. S. (2015). Wireless sensor networks for agriculture: The state-of-the-art in practice and future challenges.

pythontutorial. (n.d.). *Python Write Text File*. Retrieved from Python Tutorial:
<https://www.pythontutorial.net/python-basics/python-write-text-file/>

Ranger, S. (2020, Feb 3). *What is the IoT?* Retrieved from ZDNet:
<https://www.zdnet.com/article/what-is-the-internet-of-things-everything-you-need-to-know-about-the-iot-right-now/>

Ranger, S. (2020, Feb 3). *What is the IoT? Everything you need to know about the Internet of Things right now*. Retrieved from ZDNet: <https://www.zdnet.com/article/what-is-the-internet-of-things-everything-you-need-to-know-about-the-iot-right-now/>

Ranger, S. (2020). *What is the IoT? Everything you need to know about the Internet of Things right now*.

Ranger, S. (2022, Feb 25). *What is cloud computing?* Retrieved from ZDNet:
<https://www.zdnet.com/article/what-is-cloud-computing-everything-you-need-to-know-about-the-cloud/>

Rapp, A. (2015, September 16). *XBee-Arduino Library*. Retrieved from Arduino Reference:
<https://www.arduino.cc/reference/en/libraries/xbee-arduino-library/>

Raspberry Pi GPIO Pinout. (n.d.). Retrieved from Raspberry Pi Pinout: <https://pinout.xyz/>

Shawn. (2020). *How to use Raspberry Pi GPIO Pins - Python Tutorial*. Retrieved from Seeedstudio:
<https://www.seeedstudio.com/blog/2020/02/19/how-to-use-raspberry-pi-gpio-pins-python-tutorial/>

Shi, J., Wan, J., Yan, H., & Suo, H. (2011). A Survey of Cyber-Physical Systems.

Stergiou, C., Psannis, K. E., Kim, B.-G., & Gupta, B. (2016). Secure integration of IoT and Cloud Computing.

Top 6 IoT Communication Protocols. (2020, August 25). Retrieved from Wisilica:
<https://wisilica.com/company/top-6-iot-communication-protocols/>

Tractor Junction. (2021, June 3). *Types of Smart Sensors in Agriculture For Farming in India*. Retrieved from Tractor Junction: <https://www.tractorjunction.com/blog/types-of-smart-sensors-in-agriculture-for-farming-in-india/>

Wagner, T., Herrmann, C., & Thiede, S. (2017). Industry 4.0 impacts on lean production systems.

Watts, S., & Raza, M. (2019, June 15). *SaaS vs PaaS vs IaaS: What's the difference & How to choose*. Retrieved from BMC Software: <https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/>

Wolfert, S., Ge, L., Verdouw, C., & Bogaardt, M. J. (2017). Big Data in Smart Farming - A review.

Xu, L. D., Xu, E. L., & Li, L. (2018). Industry 4.0: state of the art and future trends.

Yokogawa. (n.d.). *Sensor Technology and its Applications*. Retrieved from Yokogawa Co-innovating tomorrow: <https://www.yokogawa.com/special/sensing-technology/usage/>

Zigurat. (2019). *What Do the Next Five Years Hold For the IoT?* Retrieved from Zigurat Innovation & Technology Business School: <https://www.e-zigurat.com/innovation-school/blog/what-do-the-next-five-years-hold-for-the-iot/>

Appendices

Appendix A – XBee S2C Datasheet

Received from <https://www.digi.com/resources/documentation/digidocs/pdfs/90001500.pdf>

Specification	XBee value	XBee-PRO value
Indoor / urban range	Up to 200 ft (60 m)	Up to 300 ft. (90 m)
Outdoor RF line-of-sight range	Up to 4000 ft (1200 m)	Up to 2 miles (3200 m)
Transmit power output (software selectable)	6.3 mW (8 dBm), Boost mode ¹ 3.1 mW (5 dBm), Normal mode Channel 26 max power is 0.3 mW (-5 dBm)	63 mW (18 dBm) ²
RF data rate	250,000 b/s	250,000 b/s
Maximum data throughput	Up to 96,000 b/s	Up to 96,000 b/s
UART interface data rate	1200 b/s to 250,000 b/s	1200 b/s to 250,000 b/s
SPI data rate	Up to 5 Mb/s (burst)	Up to 5 Mb/s (burst)
Receiver sensitivity	-102 dBm, Boost mode -100 dBm, Normal mode	-101 dBm

Specification	XBee	XBee-PRO
Supply voltage	2.1 - 3.6 V	2.7 - 3.6 V
Transmit current (typical, VCC = 3.3 V)	45 mA (8 dBm, Boost mode) 33 mA (5 dBm, Normal mode)	120 mA (18 dBm)
Idle / receive current (typical, VCC = 3.3 V)	31 mA (Boost mode) 28 mA (Normal mode)	31 mA
Power-down current	<1 uA @ 25C	<1 uA @ 25C

Pin	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART data out
3	DIN/CONFIG	Input	UART data In
4	SPI_MISO	Output	Serial Peripheral Interface (SPI) Data Out
5	RESET	Input	Module reset (reset pulse must be at least 200 ns). This must be driven as an open drain/collector. The device drives this line low when a reset occurs. Never drive this line high.
6	PWM0/RSSI PWM	Output	PWM output 0 / RX signal strength indicator
7	PWM1	Output	PWM output 1
8	[Reserved]	-	Do not connect
9	DI8/SLEEP_RQ/DTR	Input	Pin sleep control line or digital input 8
10	GND	-	Ground
11	DIO4/SPI_MOSI	Both	Digital I/O 4 / SPI Data In
12	DIO7/CTS	Both	Digital I/O 7 / Clear-to-send flow control
13	ON/SLEEP	Output	Device sleep status indicator
14	V _{REF}	-	Feature not supported on this device. Used on other XBee devices for analog voltage reference.
15	DIO5/ASSOC	Both	Digital I/O 5 / Associated indicator
16	DIO6/RTS	Both	Digital I/O 6 / Request-to-send flow control
17	DIO3/AD3/SPI_SSEL	Both	Digital I/O 3 / Analog input 3 / SPI select
18	DIO2/AD2/SPI_CLK	Both	Digital I/O 2 / Analog input 2 / SPI clock
19	DIO1/AD1/SPI_ATTN	Both	Digital I/O 1 / Analog input 1 / SPI Attention
20	DIO0/AD0	Both	Digital I/O 0 / Analog input 0

Appendix B – Arduino Datasheet

Received from <http://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf>

Pin	Function	Type	Description
1	NC	NC	Not connected
2	IOREF	IOREF	Reference for digital logic V - connected to 5V
3	Reset	Reset	Reset
4	+3V3	Power	+3V3 Power Rail
5	+5V	Power	+5V Power Rail
6	GND	Power	Ground
7	GND	Power	Ground
8	VIN	Power	Voltage Input
9	A0	Analog/GPIO	Analog input 0 /GPIO
10	A1	Analog/GPIO	Analog input 1 /GPIO
11	A2	Analog/GPIO	Analog input 2 /GPIO
12	A3	Analog/GPIO	Analog input 3 /GPIO
13	A4/SDA	Analog input/I2C	Analog input 4/I2C Data line
14	A5/SCL	Analog input/I2C	Analog input 5/I2C Clock line

Pin	Function	Type	Description
1	D0	Digital/GPIO	Digital pin 0/GPIO
2	D1	Digital/GPIO	Digital pin 1/GPIO
3	D2	Digital/GPIO	Digital pin 2/GPIO
4	D3	Digital/GPIO	Digital pin 3/GPIO
5	D4	Digital/GPIO	Digital pin 4/GPIO
6	D5	Digital/GPIO	Digital pin 5/GPIO
7	D6	Digital/GPIO	Digital pin 6/GPIO
8	D7	Digital/GPIO	Digital pin 7/GPIO
9	D8	Digital/GPIO	Digital pin 8/GPIO
10	D9	Digital/GPIO	Digital pin 9/GPIO
11	SS	Digital	SPI Chip Select
12	MOSI	Digital	SPI1 Main Out Secondary In
13	MISO	Digital	SPI Main In Secondary Out
14	SCK	Digital	SPI serial clock output
15	GND	Power	Ground
16	AREF	Digital	Analog reference voltage
17	A4/SD4	Digital	Analog input 4/I2C Data line (duplicated)
18	A5/SD5	Digital	Analog input 5/I2C Clock line (duplicated)

Appendix C – Raspberry Pi 3B Datasheet

Received from <https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf>

Processor:	Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz
Memory:	1GB LPDDR2 SDRAM
Connectivity:	<ul style="list-style-type: none">■ 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE■ Gigabit Ethernet over USB 2.0 (maximum throughput 300Mbps)■ 4 × USB 2.0 ports
Access:	Extended 40-pin GPIO header
Video & sound:	<ul style="list-style-type: none">■ 1 × full size HDMI■ MIPI DSI display port■ MIPI CSI camera port■ 4 pole stereo output and composite video port
Multimedia:	H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics
SD card support:	Micro SD format for loading operating system and data storage
Input power:	<ul style="list-style-type: none">■ 5V/2.5A DC via micro USB connector■ 5V DC via GPIO header■ Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
Environment:	Operating temperature, 0–50 °C
Compliance:	For a full list of local and regional product approvals, please visit www.raspberrypi.org/products/raspberry-pi-3-model-b+
Production lifetime:	The Raspberry Pi 3 Model B+ will remain in production until at least January 2023.

Appendix D – Capacitive Sensor Datasheet

Received from

https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/SEN0193_Web.pdf

Specification

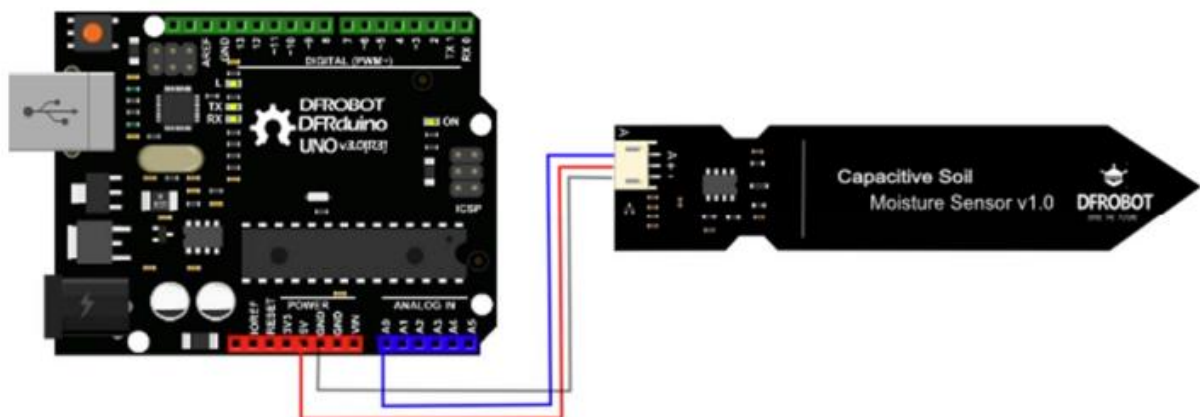
- Operating Voltage: 3.3 ~ 5.5 VDC
- Output Voltage: 0 ~ 3.0VDC
- Operating Current: 5mA
- Interface: PH2.0-3P
- Dimensions: 3.86 x 0.905 inches (L x W)
- Weight: 15g

Tutorial

Requirements

- **Hardware**
UNO x1
Capacitive Soil Moisture Sensor x1
Jumper Cable x3
- **Software**
Arduino IDE V1.6.5 [Click to Download Arduino IDE](#)

Connection Diagram



Appendix E – Python Code for Coordinator

XBee Read Test 1

```
import time
import serial
import RPi.GPIO as GPIO

from xbee import XBee, ZigBee
from digi.xbee.devices import XBeeDevice

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(23, GPIO.OUT)

def receive_data(data):
    print("Received data:{}".format(data))
    rx = data['rf_data'].decode('utf-8')

#ser = serial.Serial(port = '/dev/ttyS0', baudrate = 9600)
port = '/dev/ttyUSB0'
baudrate = 9600

def main():
    xbee = XBeeDevice(port, baudrate)
    #xbee.open()
    print('Waiting for data...\n')
    time.sleep(5)

    while True:
        try:
            def data_receive_callback(xbee_message):
                print(xbee_message.data.decode())
                xbee.add_data_received_callback(data_receive_callback)
            frame = xbee.read_data()
            data = frame.data
            print(data)
        except KeyboardInterrupt:
            break
    xbee.close()

if __name__ == '__main__':
    main()
```

XBee Read Test 2

```
import serial
import RPi.GPIO as GPIO
import os, time

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)

port = serial.Serial("/dev/ttyUSB0", baudrate = 9600, timeout = 1)
while True:
    rcv = port.readline().strip("b").decode().decode('utf-8')
    print(rcv)
```

XBee Read Test 3

```
import serial, time
import RPi.GPIO as GPIO
import os
from digi.xbee.devices import DigiMeshDevice, XBeeDevice, RemoteXBeeDevice,
RemoteDigiMeshDevice
from digi.xbee.models.address import XBee64BitAddress
from xbee import XBee, XBee
from digi.xbee.io import IOLine

port = "/dev/ttyUSB0"
FOLDER = "/home/pi/WSN"
baud = 9600

READ = []
router_addr = []
router_sample = []
router = []
no_routers = 0
with open('ADDR.txt', 'r') as f:

    READ = f.readlines()
    for i in READ:
        router_addr.append(i.strip())
    print("Router list:")
    print(router_addr)

def main():
    coordinator = DigiMeshDevice(port,baud)
    for line in router_addr:
        d = RemoteDigiMeshDevice(coordinator,
XBee64BitAddress.from_hex_string(line))
        router.append(d)
    no_routers = len(router)
    router_sample = [0 for i in range(no_routers)]
    try:
        while 1:
            if coordinator is not None and not coordinator.is_open():
                coordinator.open()
                print('Coordinator opened')
                coord_addr = coordinator.get_64bit_addr()
                coord_NI = coordinator.get_node_id()
                print('Connected to device ' + str(coord_addr) + ' Digimesh
coordinator ' + str(coord_NI))
```

```

        time.sleep(5)
    #while(1):
        for s in range(no_routers):
            d = RemoteDigiMeshDevice(coordinator,
XBee64BitAddress.from_hex_string(router_addr[s]))
            if d is not None:
                router_sample[s] = d.read_io_sample()
                io_line = IOLine.DIO3_AD3
                if(not isinstance(router_sample[s], int)):
                    if(router_sample[s].has_analog_value(io_line)):
print(d.read_io_sample().get_analog_value(io_line))
                print(f'Router {s} has DIO3 value: %s' %
router_sample[s].get_analog_value(io_line))

            finally:
                if coordinator is not None and coordinator.is_open():
                    print("Closing")
                    coordinator.close()

if __name__ == '__main__':
    main()

```

XBee Read Test 4

```

from digi.xbee.devices import DigiMeshDevice, RemoteDigiMeshDevice
from digi.xbee.io import IOLine

port = '/dev/ttyUSB0'
baudrate = 9600
ADDRESS_LIST = []
router_list = []
num_routers = 0
IOLINE_IN = IOLine.DIO3_AD3
IOLINE_VOLTAGE = IOLine.DIO1_AD1

with open('ADDR.txt','r') as f:
    READ = f.readlines()
    for i in READ:
        ADDRESS_LIST.append(i.strip())
    print('Router List: ')
    print(ADDRESS_LIST)

def main():
    coordinator = DigiMeshDevice(port,baudrate)

    while True:
        try:
            coordinator.open()
            def io_sample_callback(io_sample, remote_xbee, send_time):
                print("Sample received from %s - %s" %
(remote_xbee.get_64bit_addr(),io_sample.get_analog_value(IOLINE_IN)))
                print("Reference maximum: %s" %
io_sample.get_analog_value(IOLINE_VOLTAGE))
                print("Voltage level: %s" %
(io_sample.get_analog_value(IOLINE_VOLTAGE)/1023*100))
            coordinator.add_io_sample_received_callback(io_sample_callback)

```

```

        input()
    except KeyboardInterrupt:
        break
    coordinator.close()

    #for line in ADDRESS_LIST:
    #    remote_device = RemoteDigiMeshDevice(coordinator,
XBee64BitAddress.from_hex_string(line))
    #    router_list.append(remote_device)
    #num_routers = len(router_list)

if __name__ == '__main__':
    main()

```

XBee with Voltage level adjustment

```

from digi.xbee.devices import DigiMeshDevice, RemoteDigiMeshDevice
from digi.xbee.io import IOLine
port = '/dev/ttyUSB0'
baudrate = 9600
ADDRESS_LIST = []
router_list = []
num_routers = 0
IOLINE_IN = IOLine.DIO3_AD3
IOLINE_VOLTAGE = IOLine.DIO1_AD1

with open('ADDR.txt','r') as f:
    READ = f.readlines()
    for i in READ:
        ADDRESS_LIST.append(i.strip())
    print('Router List: ')
    print(ADDRESS_LIST)
def main():
    coordinator = DigiMeshDevice(port,baudrate)

    while True:
        try:
            coordinator.open()
            def io_sample_callback(io_sample, remote_xbee, send_time):
                Node_ID = remote_xbee.get_parameter('NI').decode()
                Sample = io_sample.get_analog_value(IOLINE_IN)
                voltage_level = Sample/io_sample.get_analog_value(IOLINE_VOLTAGE)/1023
                Sample_adjusted = Sample/voltage_level
                print("Sample received from %s - %s" % (Node_ID,Sample))
                print("Reference maximum: %s" % voltage_level)
                io_sample.get_analog_value(IOLINE_VOLTAGE)
                print("Voltage level: %s" % voltage_level)
                print("Sample received from %s adjusted for voltage: %s" % (Node_ID, Sample_adjusted))
            coordinator.add_io_sample_received_callback(io_sample_callback)
            input()
        except KeyboardInterrupt:
            break
    coordinator.close()

    #for line in ADDRESS_LIST:

```

```

#             remote_device = RemoteDigiMeshDevice(coordinator,
XBee64BitAddress.from_hex_string(line))
#         router_list.append(remote_device)
# num_routers = len(router_list)

if __name__ == '__main__':
    main()

```

XBee Final Script

```

import time
import dropbox

from dropbox.files import WriteMode
from dropbox.exceptions import ApiError, AuthError

from digi.xbee.devices import DigiMeshDevice, RemoteDigiMeshDevice
from digimesh.xbee.io import IOLine

port = '/dev/ttyUSB0'
baudrate = 9600
ADDRESS_LIST = []
router_list = []
num_routers = 0
IOLINE_IN = IOLine.DIO3_AD3
IOLINE_VOLTAGE = IOLine.DIO1_AD1

MOISTURE_100 = 178
MOISTURE_0 = 348

dbx = dropbox.Dropbox('sl.BMbZq9yVPp19AV0SJj8Qat5q53dZDCIPw-6PedKmWHyQCepJRI-
UgZpdDa8oBzxearaoQZsfsb-
7nKY_2sGBKbbBh2o6hiur0ntLvObSWMwnronjYABWax_pPhIhKyDL72VwK4P8PRP')

with open('ADDR.txt','r') as f:
    READ = f.readlines()
    for i in READ:
        ADDRESS_LIST.append(i.strip())
    print('Router List: ')
    print(ADDRESS_LIST)

def _map(x, in_min, in_max, out_min, out_max):
    return int((x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min)

def getMoistureLevel(reading):
    if reading >= MOISTURE_0:
        return 0
    if reading <= MOISTURE_100:
        return 100
    if reading < MOISTURE_0 and reading > MOISTURE_100:
        Moisture_percentage = _map(reading,MOISTURE_0,MOISTURE_100,0,100)
        return Moisture_percentage

def writetext(s,addr):
    with open(addr+'.txt','w') as f:
        f.write(s)

def uploadfile(addr):
    with open(addr+'.txt', 'rb') as f:

```

```

        dbx.files_upload(f.read(),                        '/' + addr + '.txt',
mode=writeMode('overwrite'))
        f.close()

def main():
    coordinator = DigiMeshDevice(port,baudrate)

    while True:
        try:
            coordinator.open()
            def io_sample_callback(io_sample, remote_xbee, send_time):
                Node_ID = remote_xbee.get_parameter('NI').decode()
                Sample = io_sample.get_analog_value(IOLINE_IN)
                voltage_level =
io_sample.get_analog_value(IOLINE_VOLTAGE)/1023
                Sample_adjusted = Sample/voltage_level
                print("Sample received from %s - %s" % (Node_ID,Sample))
                print("Reference          maximum:          %s" %
io_sample.get_analog_value(IOLINE_VOLTAGE))
                print("Voltage          level:          %s" %
(io_sample.get_analog_value(IOLINE_VOLTAGE)/1023*100))
                print("Sample received from %s adjusted for voltage: %s" %
(Node_ID, Sample_adjusted))
                percentage = getMoistureLevel(Sample_adjusted)
                print("Moisture reading %s percent" % (percentage))

                data_string = 'Moisture sensor reading: '+str(percentage)+'%
\nBattery voltage level: '+str(int(voltage_level*100))+'% \n Replace batteries
when reading reaches 70%'

                writetext(data_string,Node_ID)
                uploadfile(Node_ID)

            coordinator.add_io_sample_received_callback(io_sample_callback)
            input()
        except KeyboardInterrupt:
            break
    coordinator.close()

    #for line in ADDRESS_LIST:
    #    remote_device = RemoteDigiMeshDevice(coordinator,
XBee64BitAddress.from_hex_string(line))
    #    router_list.append(remote_device)
    #num_routers = len(router_list)

if __name__ == '__main__':
    main()

```


Appendix F – XBee Profiles

Coordinator Profile

▼ Networking & Security

Modify networking settings

i	CH Channel	D	
i	ID Network ID	3398	
i	MT Broadcast Multi-Transmits	3	
i	PL TX Power Level	Highest [4]	▼
i	PM Power Mode	Boost Mode Enabled [1]	▼
i	RR Unicast Retries	A	Retries
i	CA CCA Threshold	0	-dBm


▼ Diagnostic - MAC Statistics and Timeouts

MAC Statistics and Timeouts.

i	BC Bytes Transmitted	13E	
i	DB Last Packet RSSI	25	
i	GD Good Packets Received	4	
i	EA MAC ACK Failure Count	1	
i	EC CCA Failure Count	0	
i	TR Transmission Failure Count	0	
i	UA Unicasts Attempted Count	4	
i	%H MAC Unicast One Hop Time	19	
i	%8 MAC Broadcast One Hop Time	2C	




















▼ Network

Change DigiMesh Network Settings

i	CE Coordinator/End-Device Mode	Standard Router [0]	▼
i	BH Broadcast Hops	0	
i	DM DigiMesh Options	0	Bitfield 
i	NH Network Hops	7	Hops
i	MR Mesh Unicast Retries	1	Mesh Uni... Retries
i	NN Network Delay Slots	3	Network ...ay Slots
i	C8 Compatibility Options	128-bit AES ECB Encryption [4]	▼


▼ Addressing

Change Addressing Settings

i	SH Serial Number High	13A200		
i	SL Serial Number Low	41C13DAF		
i	DH Destination Address High	<input type="text" value="0"/>		 
i	DL Destination Address Low	<input type="text" value="FFFF"/>		 
i	TO Transmit Options	<input type="text" value="C0"/> Bitfield		 
i	NI Node Identifier	<input type="text" value="Coordinator"/>		 
i	NT Network Discovery Back-off	<input type="text" value="82"/> * 100 ms		 
i	NO Network Discovery Options	<input type="text" value="0"/> Bitfield		 
i	CI Cluster ID	<input type="text" value="11"/>		 





▼ Diagnostic - Addressing

Addressing Diagnostics and Timing.

i	N? Network Discovery Timeout	3C8A	
---	-------------------------------------	------	---













▼ Security

Change Security Parameters

i	EE Encryption Enable	<input type="text" value="Disabled [0]"/>	 
i	KY AES Encryption Key	<input type="text"/>	 

▼ Serial Interfacing















Change module interfacing options

i	BD Baud Rate	<input type="text" value="9600 [3]"/>	 
i	NB Parity	<input type="text" value="No Parity [0]"/>	 
i	RO Packetization Timeout	<input type="text" value="3"/> * character times	 
i	FT Flow Control Threshold	<input type="text" value="51"/> Bytes	 
i	AP API Enable	<input type="text" value="API Mode Without Escapes [1]"/>	 
i	AO API Options	<input type="text" value="API Rx Indicator - 0x90 [0]"/>	 

Sensor Node Profile










▼ Networking & Security

Modify networking settings

i	CH Channel	<input type="text" value="D"/>	 
i	ID Network ID	<input type="text" value="3398"/>	 
i	MT Broadcast Multi-Transmits	<input type="text" value="3"/>	 
i	PL TX Power Level	Highest [4]	 
i	PM Power Mode	Boost Mode Enabled [1]	 
i	RR Unicast Retries	<input type="text" value="A"/> Retries	 
i	CA CCA Threshold	<input type="text" value="0"/> -dBm	 
















▼ Diagnostic - MAC Statistics and Timeouts

MAC Statistics and Timeouts.

i	BC Bytes Transmitted	870	
i	DB Last Packet RSSI	0	
i	GD Good Packets Received	0	
i	EA MAC ACK Failure Count	0	
i	EC CCA Failure Count	0	
i	TR Transmission Failure Count	0	
i	UA Unicasts Attempted Count	0	
i	%H MAC Unicast One Hop Time	19	
i	%B MAC Broadca...ne Hop Time	2C	




















▼ Network

Change DigiMesh Network Settings

i	CE Coordinator/End-Device Mode	Standard Router [0]	 
i	BH Broadcast Hops	<input type="text" value="0"/>	 
i	DM DigiMesh Options	<input type="text" value="0"/> Bitfield 	 
i	NH Network Hops	<input type="text" value="7"/> Hops	 
i	MR Mesh Unicast Retries	<input type="text" value="1"/> Mesh Uni... Retries	 
i	NN Network Delay Slots	<input type="text" value="3"/> Network ...ay Slots	 
i	C8 Compatibility Options	128-bit AES ECB Encryption [4]	 


▼ Addressing

Change Addressing Settings

i	SH Serial Number High	13A200	
i	SL Serial Number Low	41C13E63	
i	DH Destination Address High	<input type="text" value="13A200"/>	 
i	DL Destination Address Low	<input type="text" value="41C13DAF"/>	 
i	TO Transmit Options	<input type="text" value="C0"/> Bitfield 	 
i	NI Node Identifier	<input type="text" value="Node1"/>	 
i	NT Network Discovery Back-off	<input type="text" value="82"/> * 100 ms 	 
i	NO Network Discovery Options	<input type="text" value="0"/> Bitfield 	 
i	CI Cluster ID	<input type="text" value="11"/>	 





▼ Diagnostic - Addressing

Addressing Diagnostics and Timing.

i	N? Network Discovery Timeout	3C8A	
---	-------------------------------------	------	---













▼ Security

Change Security Parameters

i	EE Encryption Enable	<input type="text" value="Disabled [0]"/>	 
i	KY AES Encryption Key	<input type="text"/>	 

▼ Serial Interfacing

Change module interfacing options

i	BD Baud Rate	<input type="text" value="9600 [3]"/>	 
i	NB Parity	<input type="text" value="No Parity [0]"/>	 
i	RO Packetization Timeout	<input type="text" value="3"/> * character times	 
i	FT Flow Control Threshold	<input type="text" value="51"/> Bytes	 
i	AP API Enable	<input type="text" value="API Mode Without Escapes [1]"/>	 
i	AO API Options	<input type="text" value="API Rx Indicator - 0x90 [0]"/>	 

▼ I/O Settings

Modify DIO and ADC options

i	D0 DIO0 Configuration	ADC [2]	↺ ↻ 🔧
i	D1 SPI_ATTN/AD1/...Configuration	ADC [2]	↺ ↻ 🔧
i	D2 SPI_SCLK/AD2/...Configuration	ADC [2]	↺ ↻ 🔧
i	D3 SPI_SSEL/AD3/...Configuration	ADC [2]	↺ ↻ 🔧
i	D4 SPI_MOSI/DIO4 Configuration	Disabled [0]	↺ ↻ 🔧
i	D5 DIO5 Configuration	Associated indicator [1]	↺ ↻ 🔧
i	D6 DIO6 Configuration	Disabled [0]	↺ ↻ 🔧
i	D7 DIO7 Configuration	CTS flow control [1]	↺ ↻ 🔧
i	D8 DIO8/SLEEP_REQUEST	Sleep Request [1]	↺ ↻ 🔧
i	D9 DIO9/ON_SLEEP	ON/SLEEP Output [1]	↺ ↻ 🔧
i	P0 DIO10/RSSI/PWM0	RSSI PWM0 Output [1]	↺ ↻ 🔧
i	P1 DIO11/PWM1	Disabled [0]	↺ ↻ 🔧
i	P2 SPI_MISO/DIO12 Configuration	Disabled [0]	↺ ↻ 🔧
i	PR Pull-up Resistor Enable	1FFF Bitfield	↺ ↻ 🔧
i	PD Pull-up/down Direction	1FFF Bitfield	↺ ↻ 🔧
i	M0 PWM0 Duty Cycle	0	↺ ↻ 🔧
i	M1 PWM1 Duty Cycle	0	↺ ↻ 🔧
i	LT Associate LED Blink Time	0 * 10 ms	↺ ↻ 🔧
i	RP RSSI PWM Timer	28 * 100 ms	↺ ↻ 🔧

▼ I/O Sampling

Configure I/O Sampling Parameters.

i	IC DIO Change Detect	0 Bitfield	↺ ↻ 🔧
i	IF Sleep Sample Rate	1	↺ ↻ 🔧
i	IR Sample Rate	1388 * 1 ms	↺ ↻ 🔧









▼ Sleep Commands

Configure Sleep Parameters

i	SM Sleep Mode	Async. Cyclic Sleep Pin Wake [5]	▼	↺	💡	
i	SO Sleep Options	0	Bitfield	📱	↺	💡
i	SN Number of Cy...een ON_SLEEP	1		↺	💡	
i	SP Sleep Time	1F4	* 10 ms	📱	↺	💡
i	ST Wake Time	4E20	* 1 ms	📱	↺	💡
i	WH Wake Host Delay	0	* 1 ms	📱	↺	💡








▼ AT Command Options

Change AT Command Mode Behavior

i	CC Command Sequence Character	<input type="text" value="2B"/>	Recommen...(ASCII)		
i	CT Command Mode Timeout	<input type="text" value="64"/>	* 100ms 		
i	GT Guard Times	<input type="text" value="3E8"/>	* 1ms 		

▼ Firmware Version/Information

Access Firmware Version/Information

i	VR Firmware Version	9002	
i	HV Hardware Version	2E4D	
i	DD Device Type Identifier	<input type="text" value="50000"/>	 
i	NP Maximum Pack...ayload Bytes	49	
i	CK Configuration CRC	49C4	
i	%V Supply Voltage	CEC	

Appendix G – Part List

Raspberry Pi Starter Kit – Received from <https://www.jaycar.co.nz/raspberry-pi-starter-kit/p/XC9010?pos=19&queryId=06d8126f418931fec24358459de3d0e6&sort=relevance>

Price: \$169 NZD

Includes:

Raspberry Pi 3B

Acrylic Case

Power Supply and USB Cable

Book (Programming the Raspberry Pi: Getting Started with Python)

Micro SD Card loaded with NOOBS software

Getting Started Guide

Duinotech Arduino Starter Kit – Received from <https://www.jaycar.co.nz/duinotech-arduino-starter-kit/p/XC3902>

Price: \$45.90

Includes:

1 x Duinotech UNO Arduino-Compatible Board

1 x USB Cable

1 x Breadboard

1 x Pack of Jumper Leads

SEN0193 Capacitive Soil Moisture Sensor – Received from https://www.digikey.co.nz/product-detail/en/dfrobot/SEN0193/1738-1184-ND/6588605?utm_adgroup=Evaluation%20Boards%20-%20Expansion%20Boards%2C%20Daughter%20Cards&utm_source=google&utm_medium=cpc&utm_campaign=Shopping_Product_Development%20Boards%2C%20Kits%2C%20Programmers&utm_term=&productid=6588605&gclid=Cj0KCQjwh_eFBhDZARIsALHjIKd-VwahXCUvgmX9TNUwrP1pluJDH4IN4bg3hT2TD_3UdH4TJL63k-0aAiWIEALw_wcB

Price: \$12.10 NZD each

DFR0015 XBee Shield for Arduino – Received from https://www.digikey.co.nz/product-detail/en/dfrobot/DFR0015/1738-1230-ND/7087127?utm_adgroup=Evaluation%20Boards%20-%20Expansion%20Boards%2C%20Daughter%20Cards&utm_source=google&utm_medium=cpc&utm_campaign=Shopping_Product_Development%20Boards%2C%20Kits%2C%20Programmers&utm_term=&productid=7087127&gclid=CjwKCAjwqvvyFBhB7EiwAER786axZNq0EAnty6Ef2Hn8eWl1v6gm1rSjCZ_7UZGbpdp79pB66urv-nRoCZucQAvD_BwE

Price: \$16.54NZD

Wi-Fi Mini ESP8266 Main Board – Received from [https://www.jaycar.co.nz/wifi-mini-esp8266-main-](https://www.jaycar.co.nz/wifi-mini-esp8266-main-board/p/XC3802?gclid=EAlalQobChMIwci9sMP68AIVQq6WCh3DrABiEAQYByABEgljmvD_BwE)

[board/p/XC3802?gclid=EAlalQobChMIwci9sMP68AIVQq6WCh3DrABiEAQYByABEgljmvD_BwE](https://www.jaycar.co.nz/wifi-mini-esp8266-main-board/p/XC3802?gclid=EAlalQobChMIwci9sMP68AIVQq6WCh3DrABiEAQYByABEgljmvD_BwE)

Price: \$28.90 NZD

Arduino Compatible Breadboard with 400 Tie Points – Received from

[https://www.jaycar.co.nz/arduino-compatible-breadboard-with-400-tie-](https://www.jaycar.co.nz/arduino-compatible-breadboard-with-400-tie-points/p/PB8820?pos=9&queryId=b0380ecc4332097a20e8f9e69f5a51c7&sort=relevance)

[points/p/PB8820?pos=9&queryId=b0380ecc4332097a20e8f9e69f5a51c7&sort=relevance](https://www.jaycar.co.nz/arduino-compatible-breadboard-with-400-tie-points/p/PB8820?pos=9&queryId=b0380ecc4332097a20e8f9e69f5a51c7&sort=relevance)

Price: \$9.50 each

XB24-AUI-001 XBee Transceiver Module 2.4GHz - Received from

<https://www.digikey.co.nz/product-detail/en/digi/XB24-AUI-001/XB24-AUI-001-ND/935967>

Price: \$29.10 NZD each

WRL-11812 XBee Explorer USB – Received from

[https://www.digikey.co.nz/en/products/detail/sparkfun-electronics/WRL-](https://www.digikey.co.nz/en/products/detail/sparkfun-electronics/WRL-11812/5762455?utm_medium=email&utm_source=oce&utm_campaign=4251_OCE22RT&utm_content=productdetail_NZ&utm_cid=2698999&so=74841347&mkt_tok=MDI4LVNYSy01MDcAAAGDM6c0Ajmd-ngpIYQkolZwz8wa5xb7c4LD4zYFAw5VeowSLI71ltfG9m-i_dI0LtHuxL-Z1isE1VNxV5Fsswzhj-MczWppsg6yChlExCU)

[11812/5762455?utm_medium=email&utm_source=oce&utm_campaign=4251_OCE22RT&utm](https://www.digikey.co.nz/en/products/detail/sparkfun-electronics/WRL-11812/5762455?utm_medium=email&utm_source=oce&utm_campaign=4251_OCE22RT&utm_content=productdetail_NZ&utm_cid=2698999&so=74841347&mkt_tok=MDI4LVNYSy01MDcAAAGDM6c0Ajmd-ngpIYQkolZwz8wa5xb7c4LD4zYFAw5VeowSLI71ltfG9m-i_dI0LtHuxL-Z1isE1VNxV5Fsswzhj-MczWppsg6yChlExCU)

[content=productdetail_NZ&utm_cid=2698999&so=74841347&mkt_tok=MDI4LVNYSy01MDcA](https://www.digikey.co.nz/en/products/detail/sparkfun-electronics/WRL-11812/5762455?utm_medium=email&utm_source=oce&utm_campaign=4251_OCE22RT&utm_content=productdetail_NZ&utm_cid=2698999&so=74841347&mkt_tok=MDI4LVNYSy01MDcAAAGDM6c0Ajmd-ngpIYQkolZwz8wa5xb7c4LD4zYFAw5VeowSLI71ltfG9m-i_dI0LtHuxL-Z1isE1VNxV5Fsswzhj-MczWppsg6yChlExCU)

[AAGDM6c0Ajmd-ngpIYQkolZwz8wa5xb7c4LD4zYFAw5VeowSLI71ltfG9m-i_dI0LtHuxL-](https://www.digikey.co.nz/en/products/detail/sparkfun-electronics/WRL-11812/5762455?utm_medium=email&utm_source=oce&utm_campaign=4251_OCE22RT&utm_content=productdetail_NZ&utm_cid=2698999&so=74841347&mkt_tok=MDI4LVNYSy01MDcAAAGDM6c0Ajmd-ngpIYQkolZwz8wa5xb7c4LD4zYFAw5VeowSLI71ltfG9m-i_dI0LtHuxL-Z1isE1VNxV5Fsswzhj-MczWppsg6yChlExCU)

[Z1isE1VNxV5Fsswzhj-MczWppsg6yChlExCU](https://www.digikey.co.nz/en/products/detail/sparkfun-electronics/WRL-11812/5762455?utm_medium=email&utm_source=oce&utm_campaign=4251_OCE22RT&utm_content=productdetail_NZ&utm_cid=2698999&so=74841347&mkt_tok=MDI4LVNYSy01MDcAAAGDM6c0Ajmd-ngpIYQkolZwz8wa5xb7c4LD4zYFAw5VeowSLI71ltfG9m-i_dI0LtHuxL-Z1isE1VNxV5Fsswzhj-MczWppsg6yChlExCU)

Price: \$50.26 NZD