INTERNET OF THINGS WATER QUALITY MONITORING SYSTEM

CHEW YUIN YEE

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Mechatronics Engineering

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

May 2020

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : 

Name : CHEW YUIN YEE

ID No. : 15UEB02897

Date : 15 May 2020

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"INTERNET OF THINGS WATER QUALITY MONITORING SYSTEM"** was prepared by **CHEW YUIN YEE** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Mechatronics Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : 

Supervisor : Dr. Kwan Ban Hoe

Date : 16 May 2020

Signature : 

Co-Supervisor : Ir. Danny Ng Wee Kiat

Date : 16 May 2020

ABSTRACT

In this report, it will be discussing about Internet of Things (IoT) water quality monitoring system. The demand for seafood is increasing with decreasing of wild fish in the ocean. At some point, nature can no longer produce enough seafood for human consumption. Therefore, aquaculture is a tool to fill up the gap for the demand for seafood. Up until today, many aquaculture monitoring systems have yet to be connected to the internet. Farmers needs to be on the site to measure the water parameters which is time and labor-intensive. Since, global demand for seafood is increasing, any potential threat will slow down the production of aqua farming or may cause risk to the consumer's health. The aim is to develop a real-time water quality monitoring system that is accessible anywhere in the world with the access of internet by utilizing Internet of Things technology. The objectives of this project are design and develop an embedded system architecture to perform real-time water quality monitoring, integrating IoT into real-time water quality monitoring system, and develop an Android GUI to display the water quality in a graphical format. The entire embedded system will be based on Arduino platform and Firebase cloud server to store the data. Arduino Nano was used to logged data from pH sensor, temperature sensors, and turbidity sensor and store it in both SD Card and Firebase cloud server. ESP8266 was used as data transmission gateway to communicate Arduino Nano to Firebase. Moreover, an Android application was developed specifically to allow user to monitor the fish farm from time to time and provide a graphical format to view the historical logged data. In addition, a custom mounting structure and housing was designed to mount all the electronics in the floating platform and power system architecture was also designed to run the entire system on rechargeable battery and solar power. The overall system is working and it is able to log data for 24 hours 7 days at aquaculture farm without without any human assistance. In conclusion, an Internet of Things water quality monitoring system has been successfully designed and developed. This system allows aquaculture farmer to monitor each of the fish ponds in real-time with several features such as real-time data logging, real-time notification and preview historical data in graphical format. Moreover, this system will be able to assist aquaculture farmer to detect water quality problems at the early stage and thus countermeasures can be planned ahead to prevent or reduce aquaculture lost.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

°C	degree celsius
A	ampere
NH_3	ammonia
ppm	parts per million
%	percentage
V	voltage
m	milli
k	kilo
G	giga
Hz	Hertz
mg/L	milligram per liter
s	second
bps	bit per second
c	centi
±	plus-minus
Ω	ohm

BNC	Bayonet Neill-Concelman
CVI	C for Virtual Instrumentation
DC	Direct Current
DO	Dissolved Oxygen
I/O	Input/Output
IDE	Integrated Development Environment
GUI	Graphic User Interface
NTU	Nephelometric Turbidity Unit
pH	potential of Hydrogen
RAS	Recirculating Aquaculture System
UART	Universal Asynchronous Receiver-Transmitter
MQTT	Message Queuing Telemetry Transport
PLC	Programmable Logic Control
IoT	Internet of Things
VPN	Virtual Private Network

PCB	Printed Circuit Board
RTC	Real-Time Clock
SD	Secure Digital
ADC	Analog to Digital Converter
PID	Proportional–Integral–Derivative
PLC	Programmable Logic Control

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Introduction to Internet of Things (IoT) Water Quality Monitoring System

Farming in the pond also known as aquaculture existed back in 1000 BCE in China (alimentarium, 2017). At the beginning of 21st century, it becomes one of the booming industry and based on Organization of the United Nation (FAO) in 2016, in term of global production volume, aquatic plant and farm fish surpassed the number of capture fisheries in the wild in 2013. This statement shows that water quality monitoring for aquaculture is getting more important in the near future.

Water quality is referring to the condition of the water including the amount of dissolved oxygen (DO), turbidity, ammonia level, pH level, and many other parameters. Moreover, by implementing a monitoring system that consists of sensors will be able to keep track of the water quality parameters depending on the types of sensors use. The difference between IoT water quality monitoring system and conventional water quality monitoring system is that IoT water quality monitoring system will be able to interact with cloud server where the user will be able to view the water parameter values via a cloud server. Moreover, data that uploaded to the cloud can be easily accessible anytime and anywhere in the world with internet access.

In recent years, many researchers have developed various type of IoT water quality monitoring system for aquaculture. The principle behind every system is almost identical where microcontroller/microprocessor is used to read the value(s) from the sensor(s) and upload the data to a cloud server. However, in-depth, there are several differences between the system architecture and the logic flow of the program which will be discussed in the literature review.

This study is to design and develop a system that is capable of performing water quality monitoring in real-time, upload the data to the cloud and display the data to the user. The hardware system will be based on a microcontroller, 3 sensors, and a Wi-Fi module. All the data will be uploaded to the cloud server and when the user is away from the aquafarm, he/she will be able to monitor the water quality of the aquafarm.

1.2 The Importance of IoT Water Quality Monitoring System

Aquaculture is a controlled process where various types of fishes or aquatic plants are kept or breed in an enclosed environment. This will allow the fish farmers to consistently monitor the environment around the fishes and produce high-quality seafood. Aquaculture existed all around the world and can be implemented in ocean, freshwater ponds, rivers or even on land in tanks. It is estimated to have 8.6 billion of world population by 2030 and the number is expected to increase to 9.8 billion in 2050 (UN DESA, 2017). With the increasing number of world population, the demand for seafood will be affected. From 2012 to 2017, global seafood trade grows approximately 4% per annum (Beyhan, 2019). This shows that the demand for seafood is getting more and more throughout the year.

In addition, fishing in the wild can no longer meet the increasing demand for seafood as illegal fishing and overfishing cause various type of fishes go extinct and decrease significantly in number. At some point, nature can no longer produce enough seafood for human consumption. Aquaculture is a tool to fill up the gap for the demand for seafood. Based on Figure 1.1 below, it shows the comparison between capture production versus aquaculture production from the period of 1950 to 2016.

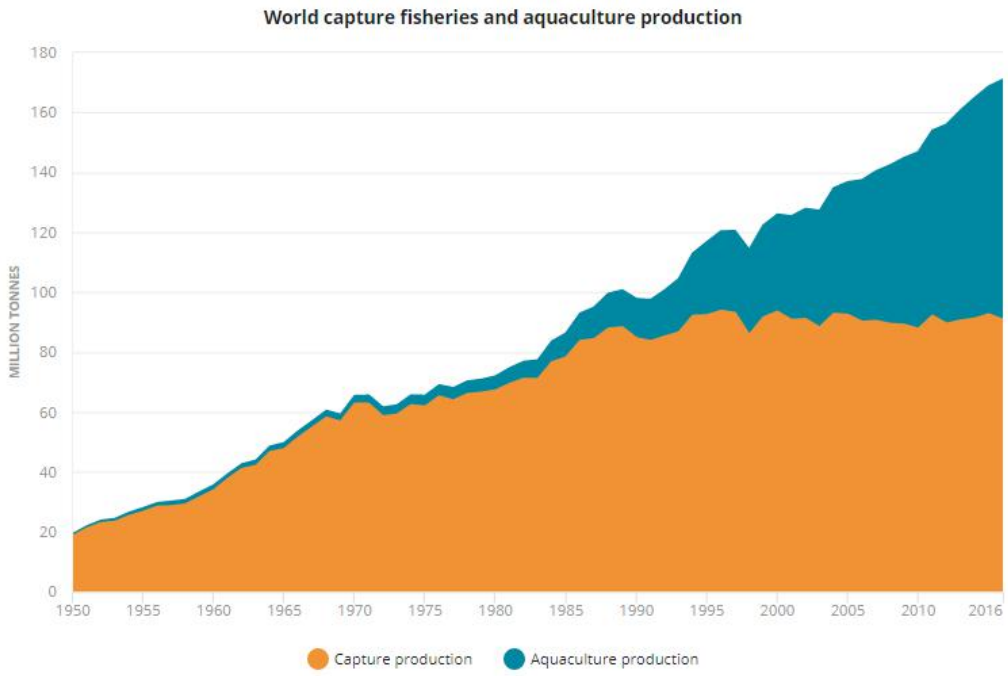


Figure 1.1: World capture fisheries and aquaculture production (FAO,2018)

From Figure 1.1, it shows that the production of fisheries increases significantly from 20 million tonnes in 1950 to approximately 170 million tonnes in 2016. It is clear that the demand for seafood has increased throughout the year. Capture production reaches its peak of approximately 80 million tonnes in 1990 and aquaculture production supply the additional fisheries in order to meet the world demand. Besides that, the graph also shows in 1990 the capture production reaches its peak and aquaculture production increased dramatically from 1990 to 2016. It is expected that aquaculture production will continue to grow in the near future. Moreover, sustainable aquaculture is very important so that it can supply enough seafood to the world. In addition, improving the efficiency of aqua farming technique with the help of advanced technology such as the Internet of Things, Artificial Intelligence, Big Data, and other technologies will be able to make aquaculture farming more sustainable in the future.

1.3 Problem Statement

Aquafarming can be implemented in indoor or outdoor. Indoor aquafarming uses a Recirculating Aquaculture System (RAS) which is a closed-loop system where fishes are grown at high density under an enclosed environment. This system is suitable where water surrounding the area is limited or the environment is not suitable for the species being cultured (Queensland Government, 2018). Since it is a closed-loop system, the water will be recirculated through the tank with a series of filters to remove waste products. If the filter fail or malfunction, the recirculated water will most likely to be contaminated and cause the fishes to die. On the other hand, outdoor aquafarming relay mostly on the external environment. If the nearby water source is contaminated, the fishes may suffocate and die due to the pollution in the water. In order to monitor the water quality, the farmer needs to be on the site to measure the water parameters which is time and labor-intensive.

It is important to make sure that the water quality and the environment the fishes live are in continuous monitoring and able to trigger an alarm when something went wrong especially for the farm that is large in size. This is because, keeping the water quality at the best will be able to reduce the risk of aquaculture disease problem. Since, global demand for seafood is increasing, any potential threat will slow down the production of aqua farming or may cause risk to the consumer's health. In the aquaculture industry, disease continues to affect the production of

seafood and bring a global impact where an estimated loss of 6 billion USD per annum (Stentiford, 2017).

Moreover, managing an aquaculture farm can be expensive and require labor to monitor and maintain the facilities. Up until today, many aquaculture monitoring systems have yet to be connected to the internet. This type of monitoring systems requires the farmer to stay at the aquafarm to monitor the water parameters. With IoT, the farmer can monitor the water quality anytime and anywhere with internet access. On top of that, by developing an Android Graphic User Interface (GUI) to show the processed the data, it allows the farmer to view the data collected in real-time and trace back all the historical data for better understanding about their aqua farm.

1.3 Aims and Objectives

In this project, the aim is to develop a real-time water quality monitoring system that is accessible anywhere in the world with the access of internet by utilizing Internet of Things technology. The objectives of the project are:

1. Design and develop an embedded system architecture to perform real-time water quality monitoring.
2. Integrating IoT into real-time water quality monitoring system.
3. Develop an Android GUI to display the water quality in a graphical format.

1.4 Scope and Limitation of the Study

The scope of this project is to design an embedded system that is able to perform real-time monitoring by integrating IoT. The first part of this project is to develop and test the embedded data monitoring system and the second part is to develop the GUI for the user to view the data.

The first limitation of this project is the cost of the sensor. Initially, 4 sensors (ammonia, pH, turbidity, and temperature) are plan to be integrated into the system. However, due to budget constraint, the ammonia sensor has to be eliminated from the system.

The second limitation is, Wi-Fi signal has to be strong around the ESP8266 module in order to transmit the data to cloud successfully or else the connection will be corrupted and the data will not be able to transmit.

1.5 Overview of Project

In this project, it consists of both hardware and software. The software used is mainly for coding and simulate the hardware. The hardware is used to deploy the prototype to the aquaculture farm to perform real-time data monitoring. Moreover, this project is categorized into three different parts which is research and report writing, building the prototype and data analysis.

In this report, it consists of five different chapters. The first chapter discusses the introduction of water quality monitoring system for aquaculture, the importance of IoT water quality monitoring system, problem statement, aim and objectives, scope and limitation of the study and overview of the project. The second chapter discusses heavily on the research done by other researchers that is related to IoT monitoring system and hardware used by other researchers are studied to provide a better decision making on the type of hardware to be used. Chapter 3 discusses about the approach to the project prototype which is the methodology. This chapter details the hardware and software used, the approach and timeline of the project. Next, chapter 4 will be discussing in-depth about the code flow of the prototype, the architecture of the prototype and result gathered by the prototype after deploy to the aquaculture farm. Finally, the last chapter summarize the overall report, state all the limitation of this project and propose future work and recommendation for this project.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Throughout the year, researchers have come out with several approaches of developing IoT based monitoring system ranging from integrating Arduino development board with ESP8266 Wi-Fi module (Daigavane and Gaikwad, 2017) or combination of Arduino development board with Raspberry Pi (Gondchawar and Kawitkar, 2016). Raspberry Pi has a built-in Wi-Fi capability which cannot be found in the Arduino development board such a Nano, Uno and Mega. However, Raspberry Pi is more power-intensive compared to Arduino board (SwitchDoc Labs, 2015) which is a disadvantage when the system is placed in the area that does not have access to continuous supply of electricity. Besides that, the total cost for both Arduino board and ESP8266 Wi-Fi module is much cheaper compared to Raspberry Pi alone. Figure 2.1 below shows the hierarchy for IoT application.

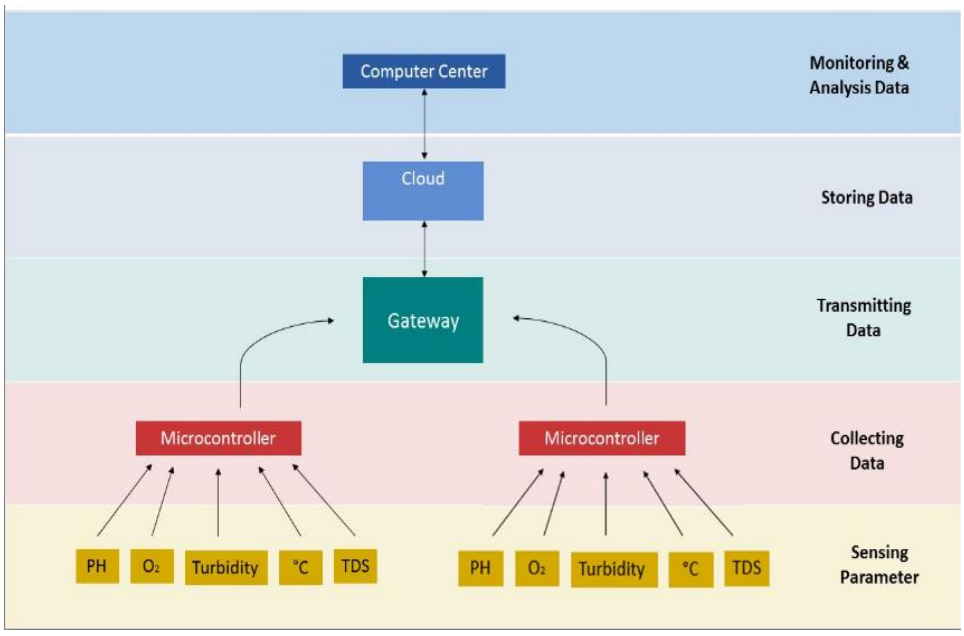


Figure 2.1: Hierarchy of IoT

The working principle for most of the aquaculture IoT system is somewhat related to Figure 2.1 hierarchy. The quality of water is measured by multiple sensors attached to the microcontroller and feed the data to the cloud server via a Wi-Fi

module. The cloud server will store the data and can be analyzed and displayed via different device to the user. There are many types of sensors used by different researchers for real-time water quality monitoring integrating with IoT system. The sensor used in the aqua farm depends entirely on the requirement of the farm. For example, Raju and Varma (2017) used 7 different types of sensors where most of them are ion sensors while Encinas, et al. (2017) used only 3 sensors to detect the physical environment of the water.

Even though most of the architecture for each system are almost similar to each other, however, the difference is the type of microcontroller used, sensors, Wi-Fi module, cloud service provider, the device to display the data and communication protocol.

IoT based water quality monitoring system is the system that uses wireless technology to store the data captured by the sensors into a cloud server and user able to monitor the system in real-time. Many types of sensors can be used in the system such as pH sensor, temperature sensor, turbidity sensor, ammonia sensor, DO sensor, carbon dioxide sensor and many more. In this project, only three sensors will be used which are pH sensor, temperature sensor, and turbidity sensor. Both microcontroller and microprocessor development board have been used by different researchers around the world to gather the data and send the data to the cloud. This project will use a microcontroller as a data logger and a data transmitter to the cloud server.

2.2 IoT Based Water Quality Monitoring System Architecture

Encinas, et al. (2017) state that oxygen, temperature and pH value of water are important to maintain at the adequate condition to make sure that the aquaculture system is in a healthy state. The focus of this group of researchers is mainly on developing IoT system that measures the most important variable in the aquaculture industry (DO, temperature, and pH) and solve the slow response time to take care of the water quality. The sensor communicates with a microcontroller via Universal Asynchronous Transmitter-Receiver (UART). Arduino microcontroller is used to control the multiplexer to read the analog signal from the sensor and send to Zigbee which is a Wi-Fi module. The data is stored in MySQL cloud service provider and display via mobile phone. In future work, it proposed to integrate artificial intelligence in the system which will bring several benefits such as process optimization, prevention of unwanted conditions and reduction of losses.

In the work of Knowledge Based Real Time Monitoring System for Aquaculture Using IoT published by Raju and Varma (2017), they are focusing on measuring the chemical parameter of the water quality such as salt, ammonia, unionized ammonia, nitrate and DO. Moreover, this system is able to provide a solution to the aqua farmer instead of just data representation in a graphical format. The system will alert the aqua farmer when the water quality parameter changes and then propose which medicine should be applied or necessary action should be taken to bring the water quality back to optimum condition. Apart from that, the system also uses solar energy as the main power supply and battery storage to store the electricity to avoid any unwanted circumstances that will interrupt the real-time operating system. This system can be improved by adding a Proportional–Integral–Derivative (PID) controller to adjust the water quality parameter automatically via sensors and actuators so that the amount neutralizing agent or other chemical can be added in a consistent and accurate manner.

The research conducted by Manyvone, Takitoge, and Ishibashi (2018) is focusing on low power water quality monitoring system for both aquaculture and agriculture IoT application. In IoT application, power consumption is one of the most important factors that need to take into consideration in the system design. In this research, the system uses beat sensors to measure different parameters of the water. On average, pH beats sensor, water level beats sensor and salinity beats sensor consumes only 70μW, 90μW and 500μW respectively. These low power sensors will be very useful for IoT application where power supply is not readily available. Moreover, these sensors effectively increase the life span of the power source which can prolong the data logging period without having to replace the power source frequently.

The paper titled Automatic monitoring and control of shrimp aquaculture and paddy field based on embedded system and IoT conducted by Sneha and Rakesh (2017), proposed a different approach of water quality monitoring system as compared to other researchers above. In this research, it uses Arduino microcontroller as a data logger and Raspberry Pi as a PID controller for actuator and Wi-Fi module. The microcontroller will log 5 different parameters which are pH, turbidity, temperature, soil moisture, and humidity. All the value log will be stored in the SD card of the Raspberry Pi if there is no internet available and upload to the cloud server when the internet is available. When the measured parameter value is

out of pre-defined range, the Raspberry Pi will activate the actuator to return the parameter value back to normal range automatically. Moreover, the real-time data monitoring also consists of user alert features which will alert the user via Telegram mobile application when there is any abnormality in the environment parameter.

The research on IoT-based Water Quality Monitoring System for Soft-Shell Crab Farming conducted by Niswar, et al. (2018) has a different type of monitoring approach compared to previous research. In this research, the monitoring system is movable and powered by a solar panel. The system architecture uses Raspberry Pi as a medium between the cloud and the sensors. It uses LabWindows and C for Virtual Instrumentation (CVI) for displaying and data analysis. Moreover, this system also uses Message Queuing Telemetry Transport (MQTT) protocol to communicate between devices, mobile, and sensors. The IoT system will alert the farmer that subscribe to the cloud server via e-mail if there is any problem with the monitored parameters. However, the limitation of the project is microprocessor consumes more power compared to a microcontroller. Since this project uses solar as the main energy supply, if the system does not have access to sunlight for a few days, it will lose its power faster if use microprocessor instead of a microcontroller.

Table 2.1: Advantage(s) and Disadvantage(s) Between Reviewed Water Quality Monitoring System Architecture.

	Advantage(s)	**Disadvantage(s)**
Encinas et al.	- Able to display data in mobile devices	- Sensor communicate via UART instead of 1-Wire interface - Require more I/O port - No real-time alert capability
Raju and Varma	- Real-time alert capability - Provide solution to aqua farmer when water quality parameter is out of range - Built-in solar power	- Do not have PID controller - Require manual adjustment when water parameter is out of range
Manyvone, Takitoge,	- Low power system	- Sensor is not available in

and Ishibashi	- Use beat sensors to measure water parameters	market
Sneha and Rakesh	- Have PID controller - Have real-time alert capability	- Use both microcontroller and microprocessor as data logger (not ideal for battery based system)
Niswar et al.	- Movable water quality monitoring system - Uses MQTT protocol which require low bandwidth for data transmission - Built-in solar power	- Microprocessor is used instead of microcontroller (not ideal for battery based system)

2.3 Water Quality Parameters

The health of the fish raised in an enclosed environment depends greatly on the water quality they live in. The water quality parameters need to be monitored closely in order to keep the fish healthy at all time. Moreover, water quality parameters such as temperature, DO, pH, turbidity, carbon dioxide, ammonia, nitrate, and many other parameters will affect the living condition of the fish based on species.

Next, the temperature of the living environment of the fish plays an important role and has a great effect on the fishes, biological filter activities, level of oxygen and metabolic rate of the fish. Fish can be classified into different species such as coolwater, coldwater and warmwater species. The temperature is crucial to determine the survivability of the fish and hence, the temperature needs to be maintained at a certain range in order to ensure the living condition of the fish is optimum. Table 2.2 shows the range of temperature requires for each species of fish.

Table 2.2: Temperature Range For Different Species of Fish (Swann,1997)

Species	**Temperature (°C)**
Coolwater	12.77 - 18.33
Coldwater	18.33 - 23.88
Warmwater	23.88 - 32.22

Coolwater fish lives in an optimum temperature of 12.77℃ to 18.33℃. Example of coolwater fish species is Walleye and Yellow Perch. Both of these species live in an optimum temperature ranging from 15.55℃ to 29.44℃ which makes them fall under the coolwater fish category. Next, for coldwater species, the example are Salmons and Trout fish. The optimum temperature for these species is ranging from 18.33℃ to 23.88℃. Lastly, warmwater fish such as Catfish and Talapia live in the temperature of 23.88℃ to 32.22℃. This temperature is important and the fish should be raised in the respective range of temperature in order to reduce disease and reduce the stress of the fish (Swann, 1997).

Another important parameter is DO in the water. Fishes raised at outdoor with a continuous supply of water externally do not have an issue with DO. However, fish raised in an enclosed environment with static water will experience lack of DO in the water. Fish requires oxygen to breathe, it will consume the oxygen dissolved in the water and convert it to carbon dioxide (Wonderopolis. 2019). After certain period of time, the amount of DO in the water will reduce causing the fish to suffocate due to lack of DO in the water. Normally, fish require approximately 4 to 5 ppm of DO and value below 4 ppm will cause the fish stress and suffocate (Leaffin, 2017). Moreover, water with lower DO tends to attract unwanted pests such as fly larvae, worms, beetle larvae and many more. In addition, based on Table 2.3 it shows that higher temperature will result in lower DO.

Table 2.3: Relationship Between Temperature and DO (Masser, et al., 2012).

Temperature of fresh water (℃)	**DO mg/L (ppm)**	**Temperature of fresh water (℃)**	**DO mg/L (ppm)**
10	10.92	24	8.25
12	10.43	26	7.99
14	9.98	28	7.75
16	9.56	30	7.53
18	9.18	32	7.32
20	8.84	34	7.13
22	8.53	36	6.95

For freshwater system, freshwater fish can survive in pH value ranging from 6 to 9.5. Lower pH will attract nitrifying bacteria and other toxic nitrogen waste which is difficult to be removed. This will lead to disease and other bacteria to grow in the water. The pH level of the water can be easily controlled by adding substances such as an alkaline buffer (Masser, et al., 1992).

Apart from that, the ammonia level in the water is also one of the major parameters that need to be monitored. Ammonia comes from the waste excreted by the fish and it will become deadly when it becomes high concentration. Besides that, unionized ammonia (NH_3) will damage the fish's tissue and every species of fish have a different level of toxicity. As long as the ammonia level is maintained below 0.02 ppm, the water is considered safe for the fish to live in. NH_3 has a direct relationship with both pH and temperature. Based on Table 2.4 , it shows that NH_3 is higher when pH and temperature are lower.

Table 2.4: Relationship Between Unionized Ammonia With Temperature and pH Value (Masser, et al. 1992).

pH	**Temperature (°C)**								
	16	18	20	22	24	26	28	30	32
5.0	99.3	99.2	99.2	99.1	99.1	99.0	98.9	98.9	98.9
5.5	99.7	97.6	97.4	97.3	97.1	96.9	96.7	96.3	96.3
6.0	93.2	92.8	92.3	92.0	91.4	90.8	90.3	96.5	89.1
6.5	81.2	80.2	79.2	78.1	77.0	75.8	74.6	89.7	72.1
7.0	57.7	56.2	54.6	53.0	51.4	49.7	48.2	73.4	45.0
7.5	30.1	28.9	27.5	26.3	25.0	23.8	22.7	46.6	20.6
8.0	12.0	11.4	10.7	10.1	9.6	9.0	8.5	8.0	7.6
8.5	4.1	3.9	3.7	3.4	3.2	3.0	2.9	2.7	2.5

Turbidity of water is referring to the amount of suspended particle in the liquid that is normally invisible to human's naked eye (LaMotte, 2017). The lower the turbidity, the clearer the water and vice versa. There are many types of solid that will make the cloudy for example clay, organic matter and fine inorganic, algae and other microorganisms (Perlman, 2016). In static water, most of the heavy particle will sink to the bottom and light particle will scatter around the water causing the

water to turn cloudy. Under those circumstances, it will cause less light able to penetrate to the bottom of the water. In addition, when the amount of particle increase, more heat will be trap by the particles and it will raise the water temperature and reduce DO level significantly. Besides that, the turbidity level will also increase mainly due to the erosion of soil and contamination of nearby water source. Clearwater measure at 20 mg/L while cloudy measure at 40 mg/L (Fondriest Environmental, 2014).

2.4 Hardware

2.4.1 Embedded Devices

In the work of Sowmya, et al. (2017), the author used Arduino Uno development board which has built-in ATmega328 chip as the main chip for the board. This development board is communicating with the XBee module as a Wi-Fi module to upload data to the cloud. Arduino board is used because it consists of 32 KB flash memory, 16 MHz clock speed, multiple input-output(I/O) port, USB interface and onboard voltage regulator which can regulate 3.3 volts that can power XBee module directly from the board. Apart from that, the Arduino board is favorable because it is open source and consist of multiple libraries and examples on the internet which make programming easy and reduce development time.

Next, the different embedded device is used by Gopavanitha and Nagaraju (2017) for real-time water quality monitoring. The author uses Raspberry Pi 3 as the main processor for the system. Moreover, the paper stated that Raspberry Pi comes with a range of driver, onboard Wi-Fi and Bluetooth device and several I/O port for interfacing with external devices. In addition, Raspberry Pi is able to measure both analog and digital output and able to work with different programming languages such as C++, C, Python, Java, and Ruby. This board can execute multiple tasks at once by running multiple scripts which microcontroller cannot performs. However, the main drawback is it requires longer startup time and more power-hungry compare as to a microcontroller.

Based on the research conducted by Liu, et al. (2018), the researchers use Arduino Pro Nano as a data logging device and LoRa device to transmit data to a network server. Lora device is preferred by the researcher because it can perform long-range data transmission and low power consumption which is suitable for

application that uses a battery as the main power source. LoRa device uses LoraWAN communication protocol which is specially design for LoRa devices. This communication protocol is different than other wireless communication protocol such as Wi-Fi and ZigBee which transmit data not far enough compared to LoRa and 3G/4G which is expensive to implement in the system. Thus, Lora device is suitable to use for the application that runs on battery and require long-range data transmission.

Next, Zaev, Babunski, and Tuneski (2016) used Programmable Logic Controller (PLC) to log the water quality parameter and consist of PLC communication module, external antenna, and telecommunication modem as the telecommunication system. The telecommunication system is used to connect PLC to the main server via Virtual Private Network (VPN) and the internet. The purpose of PLC in this research is to connect multiple sensors via PLC I/O module to PLC communication module where the data can be transmitted to the main server. The devices used in this research is the most expensive compared to other research reviewed earlier. The main advantage of this device is more robust and reliable as compared to the development board as it designed specially for industry standard.

Table 2.5: Advantage(s) and Disadvantage(s) Between Reviewed Embedded Device.

	Advantage(s)	**Disadvantage(s)**
Arduino Development Board with XBee module	- Easy to code - Library is widely available - Low cost	- Require extra PCB to mount XBee to Arduino
Raspberry Pi	- Built-in Wi-Fi - Support multiple language - Execute multiple task at once	- High power consumption - Slow boot up time
Arduino with Lora	- Low power and long range communication	- Require extra PCB to mount Lora to Arduino
PLC	- More robust and reliable - Industry standard	- Expensive - Require multiple module

2.4.2 Data Transmission Protocol

Currently, it consists of several data transmission protocol. High power data transmission protocol is Ethernet, Wi-Fi, ZigBee, ZWave and cellular (3G and 4G) while low power data transmission protocol is Bluetooth, thread, 802.15.4, LoRa, Sigfox, and NB-LTE-M. Some communication protocol only able to communicate between devices but not to the internet. In this subtopic, only Wi-Fi and ZigBee communication protocol will be discussed.

Based on the work of Mendez, Yunus, and Mukhopadhyay (2012), the researchers use WSN802G Wi-Fi module to log data from multiple sensors via a multiplexer (use to extend I/O pin) and later transmit the data using Wi-Fi protocol to the cloud server.

Next, the paper by Simbeye, Zhao and Yang (2014) used ZigBee module as the medium for data transmission. ZigBee protocol is able to support multiple communication methods such as peer to peer, point to point, point to multi-point and mesh network transparent data. Moreover, in the communication architecture, it consists of three-node in the network which is central coordinator (store information of the network), router (link network to the device) and the end device. Table 2.6 shows a comparison between Wi-Fi and ZigBee communication protocol.

Table 2.6: Wi-Fi and ZigBee Communication Protocol (EngineersGarage, 2012).

	Wi-Fi	**ZigBee**
IEEE Standard	802.11.x	802.15.4
Operating Frequency	2.4GHz, 5GHz	900-928MHz, 2.4GHz
Bandwidth	0.3MHz, 0.6MHz, 2MHz	1MHz
Data Transfer Speed	11Mbps, 54Mbps	250kbps
Bit Time	0.00185μs	4μs
Power Consumption	High	Lower than Wi-Fi
Application	Video, Web, Email	Monitoring and Control

Based on Table 2.6, both Wi-Fi and ZigBee communication protocols have its own advantage and disadvantage. Wi-Fi is able to operate at higher frequency but it consumes more power while ZigBee operates at a lower frequency and consume less power. The communication used merely depends on the application.

2.4.3 Temperature Sensor

SimBeye, Zhao, and Yang (2014) used DS18B20 thermometer as the temperature sensor for the system in their prototype. The range of this temperature sensor able to detect is -55℃ to 125℃, with the accuracy of ±0.5℃. It can be operated in between 3V to 5V which is ideal for a microcontroller that operates at 3.3V or 5V because of no extra voltage regulator needed in the circuit. Moreover, this sensor also uses a 1-Wire interface which will reduce the number of I/O pin used.

Next, a team of researcher uses waterproofed LE-438 3 in 1 sensor probe to measure the temperature of the water. This sensor consists of three different sensors integrated into one probe and a temperature sensor is one of it. Moreover, this sensor is capable to measure temperature ranging from 0℃ to 80℃ with an accuracy of ±0.5℃ (Jiang, et al., 2009).

Researcher such as Zhu, et al. (2009), make use of thermistor thermometer to measure the temperature of the water. This temperature sensor uses resistant to detect the temperature value and suitable to be used for both indoor and outdoor application. This sensor is lightweight and widely available in the market.

Another type of temperature sensor is LM35. Vanmore, et al. (2017) used this temperature sensor to detect the temperature in the surrounding environment. However, the main drawback of this sensor is, it is not waterproof and usually is used by integrating into a circuit board. This sensor is similar to LM335 which has an accuracy of 1℃ and operate at 400uA to 50mA (Tan, et al., 2006).

From the paper titled Design of Low-cost Autonomous Water Quality Monitoring System, the authors implement Atlas Scientific temperature sensor in the system. This sensor operates at 5 V and able to detect temperature ranging from -20℃ to 133℃ with the accuracy of ±0.1℃. Moreover, this sensor is designed to be fully submerged into water and come with BNC connector. The BNC will be able to connect the sensor to the circuit board. Besides that, it is also nonreactive to saltwater (Rao, et al., 2013).

Table 2.7: Advantage(s) and Disadvantage(s) of Reviewed Temperature Sensor.

	Advantage(s)	**Disadvantage(s)**
DS18B20 thermometer	- Use 1-Wire interface - Able to detect wide range of temperature - Waterproof	- Require microcontroller with ADC capability
LE-438 3 in 1 sensor probe	- Consists of multiple sensor in 1 probe - Waterproof	- Expensive
Thermistor thermometer	- Widely available in market - Small and lightweight	- Require microcontroller with ADC capability
LM35	- Operate at low current	- Not waterproof - Require signal conditioning circuit
Atlas Scientific temperature sensor	- Come with ADC module - Accurate to ±0.1℃	- Expensive - Require Atlas Scientific sensor module to function

2.4.4 pH Sensor

Fowler, et al. (1994) stated that the pH value of the water can be measured using an electronic component or via a chemical reaction. In the electrical component, it consists of an electrode that is able to react with the water and output voltage value based on the pH of the water. Next, if the pH is measured using chemical reaction method, the reagent will be added to the water which will alter the colour of the water corresponding to the pH value.

P450 series pH sensor used by Simbeye, Zhao, and Yang is able to measure pH value from -2 to 16. This sensor is able to measure negative pH value and have pH resolution of 0.01 pH. Negative pH value indicates that the molarity of the hydrogen ion in an acid solution is more than one. However, the main drawback is, this sensor is expensive.

Next, Rao, et al. (2013) choose the pH sensor from the company called Phidgets to measure the pH value in the water. This specific model name is 3550_0-ASP200-2-1 M-BNC which is able to measure pH from 0 - 14 and the operating

temperature ranging from 0℃ to 80℃. The sensor also comes with BNC connector which requires pH/ORP adapter to convert the pH value from the sensor before sending to the microcontroller or microprocessor.

Based on Spiten (2015) paper, the author used a pH probe from Atlas Scientific. This pH probe has a dimension of 15cm by 1.2cm which are long and wide respectively. The range of pH it can measure is ranging from 0 up to 14 and able to operate up to 100psi.

Table 2.8**:** Advantage(s) and Disadvantage(s) of Reviewed pH Sensor.

	Advantage(s)	**Disadvantage(s)**
P450	- Able to measure negative pH - pH resolution of 0.01	- Expensive
3550_0- ASP200-2-1 M-BNC	- Come with ADC module - Cheap - Waterproof	-
Atlas Scientific pH sensor	- Operate up to 100psi - Water proof - Come with ADC module	-

2.4.5 Turbidity Sensor

The turbidity sensor used in the work of Osman, et al. (2018) is SEN0189 analog/digital turbidity sensor. This sensor has an operating temperature of 5℃ to 90℃, 40mA maximum current and operates at 5V DC. In addition, this sensor able to operate in either analog mode or digital mode depending on the initial configuration of the sensor. This sensor is also widely used with Arduino development board. The draw back of this sensor is, it is unable to fully submerge into the water.

The system developed by Rasin and Abdullah (2014) used TSD-10 turbidity sensor to measure the cloudiness of the water. This sensor comes with phototransistor output, with the operating voltage of 3-5V DC and operate at -10℃ to 90℃. The operating principle of this sensor is when light passed through, depending on the suspended particles in the water, more particles will result in lesser light transmitted. This sensor is unable to be fully submerged into the water.

Based on the journal titled Design of sensor water turbidity based polymer optical fiber by Arifin, et al. (2017), the authors measure the turbidity of water by designing their own sensor using LED, photodetector, fiber optic, and binder sensor. The sensor is able to measure the turbidity of 0.12 NTU up to 20 NTU. This sensor measures turbidity by transmitting light from the LED via plastic optical fiber and receive at other end by a photodetector. Lower light receives by photodetector means the water turbidity is higher.

Table 2.9: Advantage(s) and Disadvtange(s) of Reviewed Turbidity Sensor.

	Advantage(s)	**Disadvantage(s)**
SEN0189	- Cheap - Widely available in market - Come with ADC module	- Unable to fully submerge into water
TSD-10	- Cheap - Widely available in market - Operate at wide range of temperature	- Unable to fully submerge into water
Turbidity Sensor Designed by Arifin and team	- Measure wide range of water turbidity - Waterproof	- Time consuming

2.4.6 Dissolved Oxygen Sensor

Folwer, et, al. (1994) stated that DO can be measured in two different methods which are by chemical reaction and using an electronic device. By chemically, it is known as Winkler's method (Deepa, 2015). This method consists of several steps and it will not be discussed here. An electronic device such as DO probe which consists of gold or platinum element and reagent solution. This solution is separated by membrane where only oxygen will be able to pass through and react with the gold or platinum element at the other side. The reaction will generate a certain voltage and it can be converted to a digital signal for more meaningful data.

In the work of Simbeye, Zhao and Yang (2014), DO in the water is measured using DO3000 sensor. This sensor is able to measure DO from 0mg/L to 20mg/L with a resolution of 0.1%. In addition, this sensor will be able to operate at the

temperature range from 0℃ to 60℃. The frame of this sensor is designed to be waterproof and to withstand different types of environment.

Galvanic DO sensor is used by Rao et, al. (2013) in the research of water quality monitoring system. This sensor consists of two electrodes which are anode and cathode. The anode which is positive electrode is made either from zinc, lead or iron while cathode which is negative electrode is made either from silver or platinum. The operating temperature of this sensor is below 50℃ and measures DO content from 0 mg/L up to 20 mg/L.

Table 3.0: Advantage(s) and Disadvantage(s) of Reviewed Dissolved Oxygen Sensor.

	Advantage(s)	**Disadvantage(s)**
DO3000	- Operate at wide range of temperature - Waterproof - Robust	- Expensive
Galvanic DO sensor	- Waterproof - Industry standard	- Operate only at temperature below 50℃

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Project Planning and Milestone

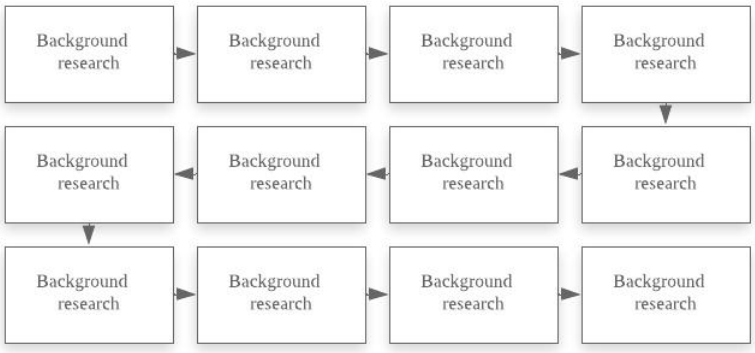


Figure 3.1: Overall flowchart of the project approach

Figure 3.1 shows the flow of the project approach to complete the final year project. Background research is done right after the final year project title is confirmed. After the research is done on the title background, the scope of the study is determined in order to identify the area to be a focus on so that the basic requirement of the final year project can be accomplished. Next, research and literature review of other researchers is carried out in order to get a better understanding of what is the current trend and the approached taken by other researchers that are related to the scope of the study.

At the proposed conceptual design stage, the type of hardware, programming language, cloud server platform, GUI to view the logged data and Integrated Development Environment (IDE) to compile the program were considered. After determining the type of hardware used, the circuit was designed using Proteus in order to simulate the actual device in the real world before constructing the circuit. Next, programming logic for the embedded system was written right after the circuit design was completed. The code was simulated using Proteus to check for any bug and the logic of the program. At the testing and debugging stage, the process mentioned earlier will be repeated until all the bug is solved and the program logic flow according to plan.

After simulation is completed, the next step is to design the Printed Circuit Board (PCB) of the embedded system using Proteus. The PCB was fabricated in the University's PCB etching lab and after the fabrication is done, the hardware component was solder on to the PCB board. The next stage is to upload the code to the hardware to test the code in the real world. In addition, testing and debugging process was carried out until the system worked according to the initial plan. The embedded system was deployed for trial run at a certain period of time after it was completed to make sure that the prototype does not have any problem.

Table 3.1 and Table 3.2 show the Gantt Chart of the project for FYP 1 and FYP 2. FYP 1 is mainly focused on the literature review, finding and report writing. Moreover, FYP 2 focuses mainly on conceptual design and product development.

Table 3.1 : Gantt Chart (FYP Part-1)

No.	Project Activities	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14
M1	Problem formulation and project planning														
M2	Determine the scope of study														
M3	Literature review														
M4	Propose conceptual design and component selection														
M5	Report writing and presentation preparation														

Table 3.2: Gantt Chart (FYP Part-2)

No.	Project Activities	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14
M1	Circuit design using software														
M2	Software development														
M3	PCB fabrication and Hardware testing														
M4	Android GUI development														
M5	Data analysis														
M6	Report writing														

3.2 System Architecture Block Diagram

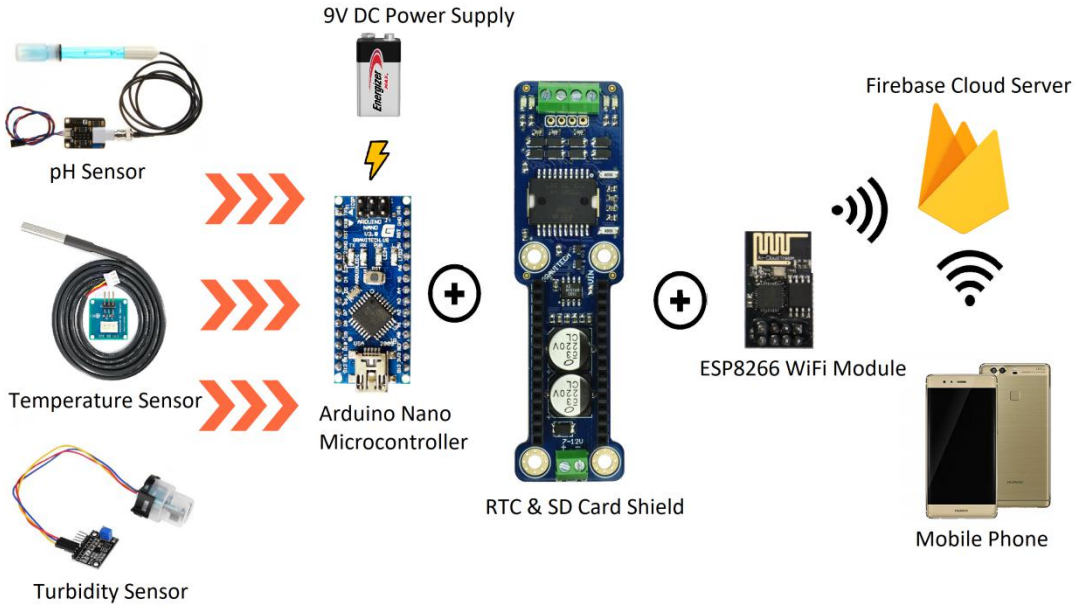


Figure 3.2: Block Diagram of Water Quality Monitoring System

From the block diagram in Figure 3.2, are there will be three sensors to measure the water parameters which are pH sensor, temperature sensor, and turbidity sensor. The value of the sensor is converted from analog to digital by Arduino Nano microcontroller for pH sensor and turbidity sensor. Moreover, the Arduino Nano microcontroller was attached on top of the RTCC and SD Card shield in order to store the data at accurate time and date and increase the data storage space whenever Wi-Fi is down. In addition, the ESP8266 Wi-Fi module is used to upload the data log by the microcontroller to the cloud server. The communication between Arduino Nano microcontroller and ESP8266 Wi-Fi module is UART. The cloud server platform used is Firebase which is one of the cloud server platform provided by Google. A mobile application will also be developed in order to view the data from Firebase.

Arduino Nano microcontroller is powered by a 9-volt battery as it has a built-in voltage regulator that will regulate the voltage to 5 volts. Moreover, since the ESP8266 Wi-Fi module only operates at 3.3 volts, it will be powered by the microcontroller via a pin that supplies 3.3 volts. In order to run the system for a long period of time, it is put into sleep mode whenever it is not logging any data. This will reduce the power consumption of the entire system.

3.3 Water Quality Monitoring System Program Flow Chart

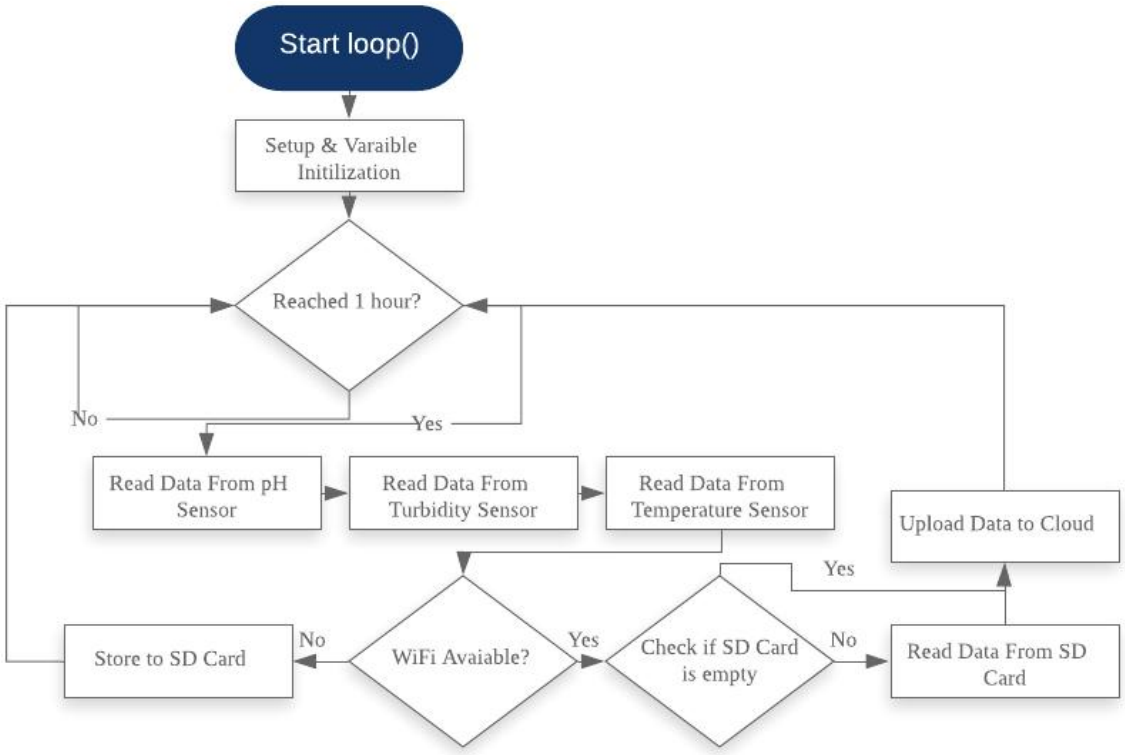


Figure 3.3: Microcontroller program flow chart

Figure 3.3 shows the logic flow of the aquaculture water quality monitoring system. When the microcontroller is powered up, it will start with the setting up system such as Wi-Fi, SD Card, Real-Time Clock (RTC), UART and I/O pins for the sensors. After that, the microcontroller will declare all the variables that it will use for data processing during the monitoring period. The monitoring approach is polling where the microcontroller keeps checking the RTC if it reaches 1 hour. Once the RTC reaches 1 hour, the microcontroller will read the data one by one from the sensors and store it in a temporary variable. Next, the microcontroller will check if Wi-Fi is available. If it is available, it will further check if the SD card has any data stored inside. If the SD card has value, it will read the value from the SD card and upload it to the cloud server. If the SD card is empty, only the current logged data will be uploaded to the cloud. If the Wi-Fi is not available, it will store the data in the SD card and continue polling from the beginning.

3.4 Water Quality Monitoring Android GUI Flow Chart

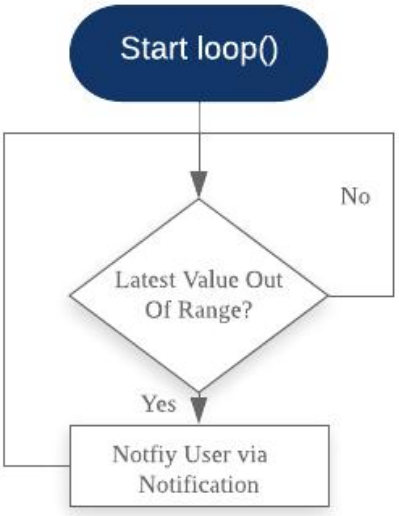


Figure 3.4: Flow Chart of Android Mobile Application (Background)

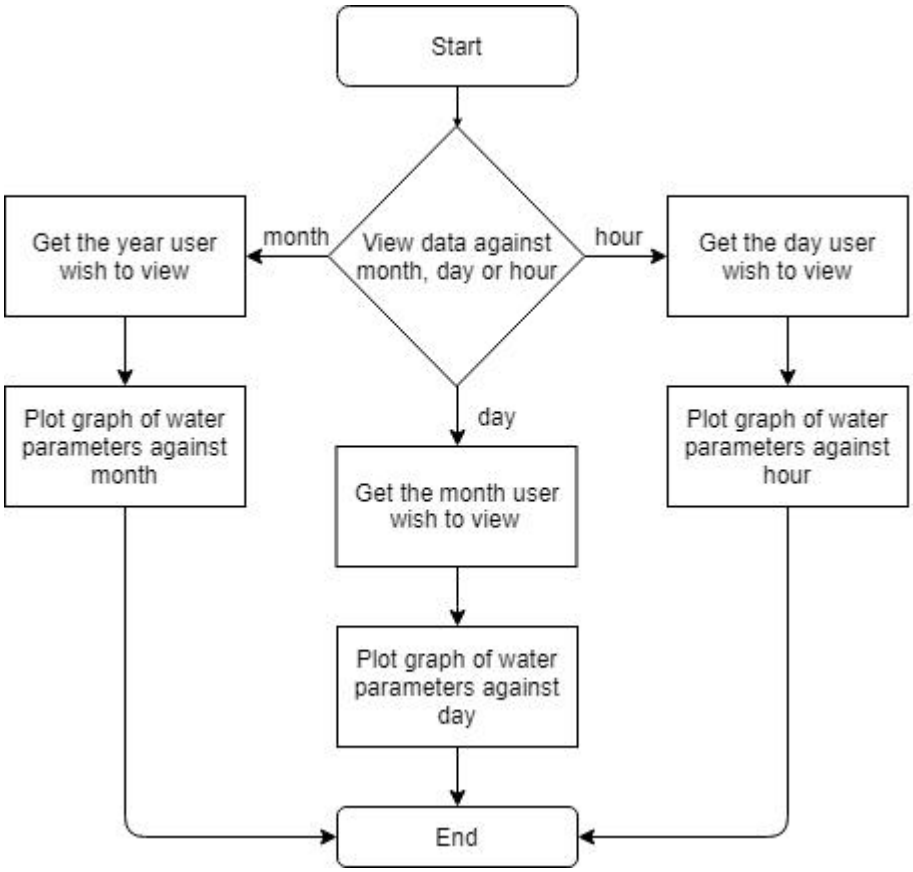


Figure 3.5: Flow Cart of Android Mobile Application

Figure 3.4 and Figure 3.5 show the flow chart of the android mobile application that runs in the background and GUI when the user clicks on the mobile application. In the background, the mobile application will check the data whenever there is new data uploaded to the cloud. When the data is out of range, it will notify the user via notification window on the mobile phone. This will notify the user in real-time whenever the water parameter value is out of range and the worker will be able to take immediate action in order to return the water parameters back to the optimum range.

Based on Figure 3.5, whenever the user wants to check the water parameter at the current time or historical data, he/she can click on the mobile application. The main page of the mobile application will allow the worker to choose which time frame of data he/she wants to view. The time frame can be in a month, day or hour. Once the user chooses the respectively time frame, the GUI will display a graph that shows the water parameter data against the respective time frame.

3.5 Hardware

3.5.1 pH Sensor

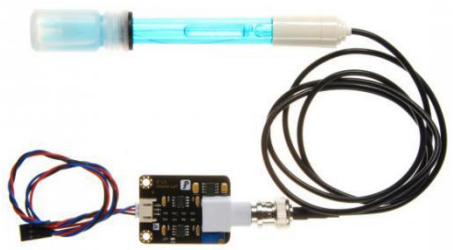


Figure 3.6: pH Electrode Probe

Figure 3.6 shows the pH sensor used in the system. This sensor is an aquarium hydroponic spare laboratory pH electrode probe with a BNC connector. Moreover, this sensor comes with an analog to digital converter (ADC) that will convert analog value into digital so that the microcontroller can process the data. Besides that, this sensor is capable of measuring pH value from 0 up to 14, at the temperature between 0°C to 60°C with the accuracy of ±0.1 pH. Apart from that, this sensor will be able to fully submerge into the water which makes it ideal for the system.

3.5.2 Temperature Sensor

Figure 3.7: DS18B20 Temperature Sensor

Currently, in the market, there are many types of temperature sensor that is capable of interfacing with microcontroller. Figure 3.7 shows the structure of the DS18B20 temperature sensor. In this project, this temperature sensor is used because it can submerge into the water and it is waterproof. This temperature sensor can measure temperature value ranging from -55℃ to 125℃ with the accuracy of ±0.5℃. The accuracy tolerance is acceptable in aquaculture water quality monitoring because ±0.5℃ changes from actual temperature will not bring much impact on the aquatic life.

3.5.3 Turbidity Sensor

Figure 3.8: Turbidity Sensor

Figure 3.8 shows the turbidity sensor used in the system. This sensor is chosen to detect the the haziness of the water that caused by tiny particles which is invisible to naked eye when it is in small amount. The turbidity sensor consist of light transmitter and receiver, when the water is very cloudy the amount of light transmitted to the receiver will reduce greatly which cause the measured value to decrease. The output signal is in analog, the signal need to be converted to digital. Moreover, the operating temperature is within 5℃ to 90℃ which is suitable to be implemented in aquaculture water quality monitoring system.

3.5.4 Arduino Nano

Figure 3.9: Arduino Nano

The microcontroller used in this system is Arduino Nano as shown in Figure 3.9. This microcontroller consists of 8 analog pin and 14 digital pin which is more than enough in this project. Moreover, this microcontroller also supports I2C, SPI, and UART communication which will be used to communicate with RTC, SD Card and ESP8266 Wi-Fi Module respectively. It also consists of an inbuilt boot loader which makes the uploading and debugging process easier. Apart from that, this microcontroller also compatible with different types of 'shields'. Shields is a device that is able to plug on top of the microcontroller and give extra features to the board such as sensors, SD Card, RTC and many more. Lastly, the size of this board is just 1.7 inch x 0.73 inch which makes the entire system design to be small and compact.

3.5.5 RTC and SD Card Shield

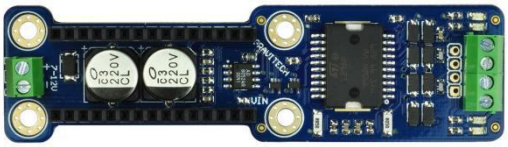


Figure 3.10: RTC and SD Card Shield for Arduino Nano

Figure 3.10 shows the shield used for Arduino Nano. This shield consists of SD Card and RTC. The SD card is used to store the data logged by Arduino Nano when there is no Wi-Fi available to prevent any data lost. The data will be uploaded to the cloud once Wi-Fi is available. Next, RTC is to keep track of the date and time when the data being logged. This shield is designed specifically for Arduino Nano where no soldering or any external wire is required. Apart from that, Arduino Nano can be powered in 2 different ways, which is from the USB 2.0 port or V_{IN} pin. However, when deploying the system to the aqua farm the system will not be connected to a

laptop/computer all the time so the USB 2.0 port cannot be used to powered. Apart from that, power via V_{IN} pin is just male to the female connector which is very loose. The shield provides a green connector to tighten the wire from the power source which makes the connection to connect securely.

3.5.6 ESP8266 Wi-Fi Module

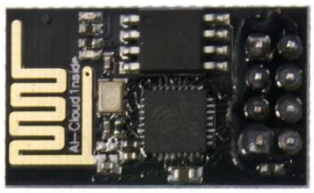


Figure 3.11: ESP8266 Wi-Fi Module (ESP-01s)

In the market, there is a variety of Wi-Fi Module that uses ESP8266 chipset by Espressif company. This Wi-Fi module is widely used in IoT application that uses a microcontroller that does not have Wi-Fi support. Figure 3.11 shows the ESP-01s model that uses ESP8266 chipset. This module is used to act as a medium to upload the data from Arduino Nano to the cloud server. It only consists of 2 GPIO pins and a pair UART pin. This is sufficient for the project since it only read the data from microcontroller via the UART pin and uploads the data via Wi-Fi.

3.6 Cloud Server

The cloud server platform that will be used in this system is Firebase. Firebase is a platform that is provided by Google. This platform is chosen because it consist of many features such as real-time database, cloud machine learning, authentication, function, and many other features. The main feature that provides by Firebase and not other cloud service provider is the real-time data. Firebase allows the device to be connected with each other in real-time which means that two devices can communicate in real-time. In addition, since Firebase is provided by Google, it can be easily linked with the android application. This makes the work simpler during the development of the system and mobile application. However, the data storage used by Firebase is in JSON format which is a non SQL format. This makes the migration of data to other cloud servers more difficult.

3.7 IDE and Debugging Tool

3.7.1 Arduino IDE

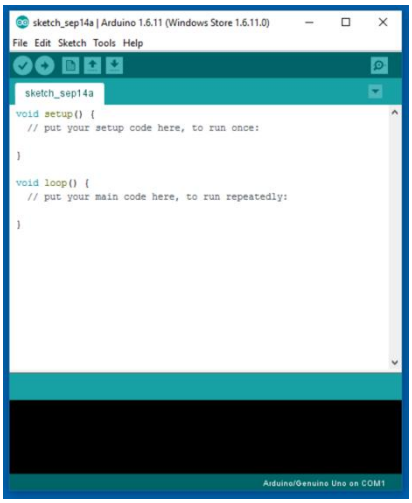


Figure 3.12: Snapshot of Arduino IDE

Figure 3.12 shows the snapshot of the Arduino IDE. This IDE is used to compile the code from C language to machine language before uploading the code into Arduino Nano and ESP8266. Since Arduino IDE is an open-source software and the Arduino development board is widely used, thus there many tutorials available online which make the development process faster and less time-consuming.

3.7.2 Proteus

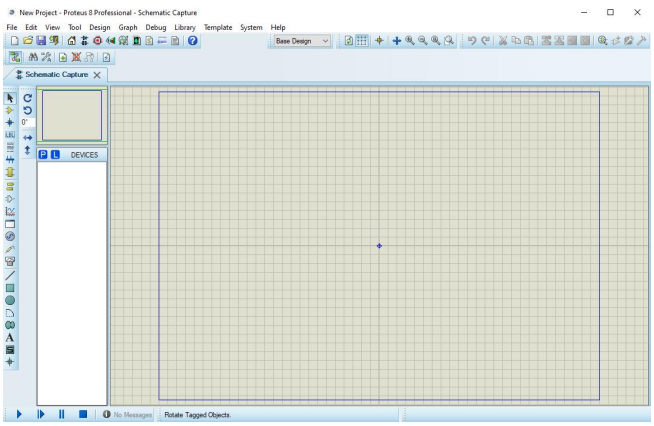


Figure 3.13: Snapshot of Proteus Software

Figure 3.13 shows the new project homepage in Proteus. Proteus is a very powerful simulator and consists of many debugging tools. In this project, Proteus is used to simulate the hardware in real life before constructing the hardware. This is useful because in Proteus the code can be debugged easily by applying breakpoint in the code, read the memory register, read the variable value and many more debugging tools. All of these tools are very useful to debug code efficiently before uploading and test the code at hardware.

3.7.3 Android Studio

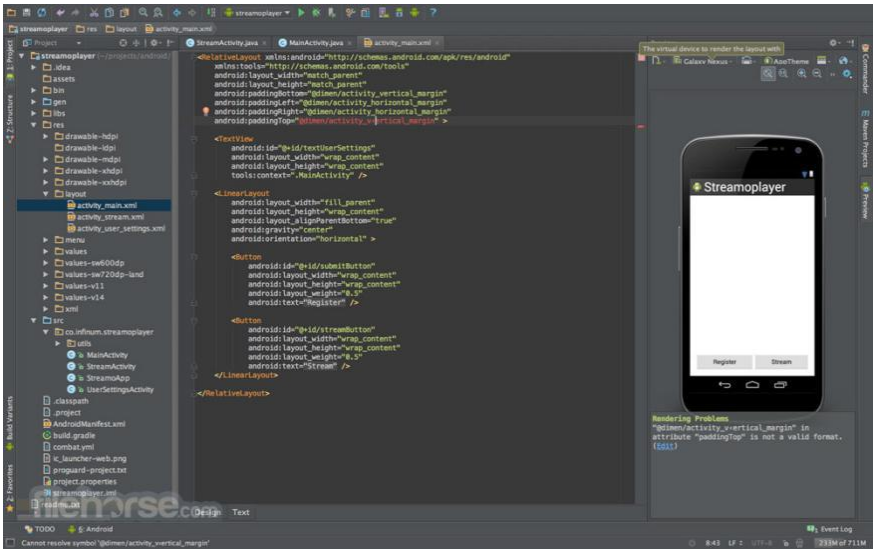


Figure 3.14: Snapshot of Android Studio IDE

Figure 3.14 shows the snapshot of the Android Studio IDE. This IDE is used to develop an android application for GUI and the back end script. The language used consists of Java for script and XML for GUI in order to fully develop a basic android application. As shown in Figure 3.14, there is a simulated cell phone in the IDE, the simulated cell phone is used for debugging purposes. Moreover, this IDE consists of many tools which one of them is Firebase. It can directly link the mobile application that is currently being developed into Firebase with just a click of a button. In addition, there are also many tutorials online to learn how to develop a simple mobile application using android studio IDE. Thus, the android studio IDE is the chosen platform to use as mobile application development.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Hardware Design

4.1.1 Circuit Diagram and Pin Connection

Figure 4.1 below shows the circuit diagram for the IoT water quality monitoring system. The circuit is drawn and illustrated by using Proteus PCB Design & Simulation software. Arduino Nano, RTCC and sensor modules are powered by 5V, while ESP-01 is powered by 3.3V. However, SD Card, ESP8266 and all the sensor modules libraries for Proteus are unable to obtain from the Internet. Thus, the software simulation is unable to be fully constructed in Proteus. Based on Figure 4.1, ESP 8266 is replaced by ESP-12F and all the sensors module is replaced by LED D1 to D4 as shown in Figure 4.1 for illustration.

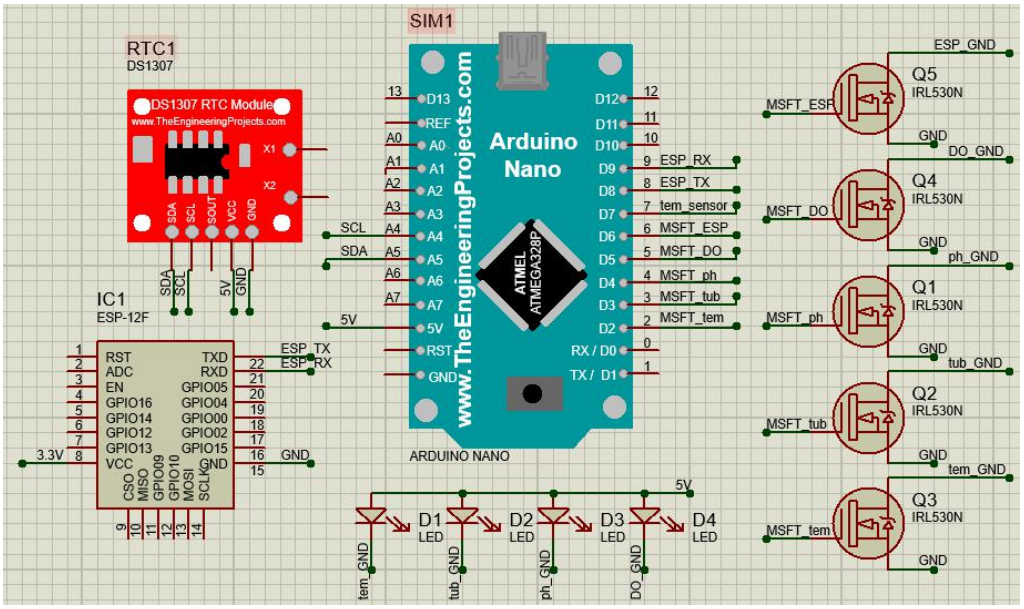


Figure 4.1: Circuit Diagram of IoT Water Quality Monitoring System

Based on Figure 4.1, the LEDs (sensor modules) for each sensor and ESP-01 are powered by IRL530N MOSFET. IRL530N MOSFET is used because, based on the datasheet of IRL530N MOSFET, this MOSFET is a logic-level gate MOSFET. In addition, only 2V is required at the gate to fully turn on the MOSFET. Based on

Appendices Figure A-1, the graph shows that with 5V gate voltage, it is able to deliver 30A which is more than enough for the sensor.

Moreover, this system is hard coded to log data in a specific interval instead of continuous logging, so in order to reduce the overall power consumption, IRL530N MOSFET is used as a switch to turn off the sensor modules and ESP8266 while it is not in use. Next, the 1000uF electrolytic capacitor is also placed parallel to the ESP8266 power source. This capacitor is used to prevent the current spike during turning on and ensure the power supply is smooth all the time as ESP8266 requires a smooth current in order to transfer data to the cloud successfully. Besides that, 4.7kΩ is placed as a pull-up resistor because the DS1850 temperature sensor has a high impedance and the interconnecting wire will generate noise which will cause the output sensor to be inaccurate. Table 4.1 below shows the connection of Arduino Nano and each of the components and modules used.

Table 4.1: Pins Connections Between Arduino Nano and Other Modules

Module	**Module Pins**	**Arduino Nano Pins**
Turbidity Module	ADC	Analog 7
pH Module	ADC	Analog 0
Temperature Module	Signal Input	Digital 7
Dissolved Oxygen Module	RX	Analog 1
	TX	Analog 2
SD Card Shield	SS	Digital 10
	MOSI	Digital 11
	MISO	Digital 12
	SCK	Digital 13
RTCC Shield	SCL	Analog 4
	SDA	Analog 5
ESP-01	TX	Digital 8
	RX	Digital 9

4.1.2 PCB Design

PCB is designed by using Eagle software. The wire diagram is constructed as shown in Figure 4.2 below. The wire diagram connection is based on Table 4.1 boards and modules.

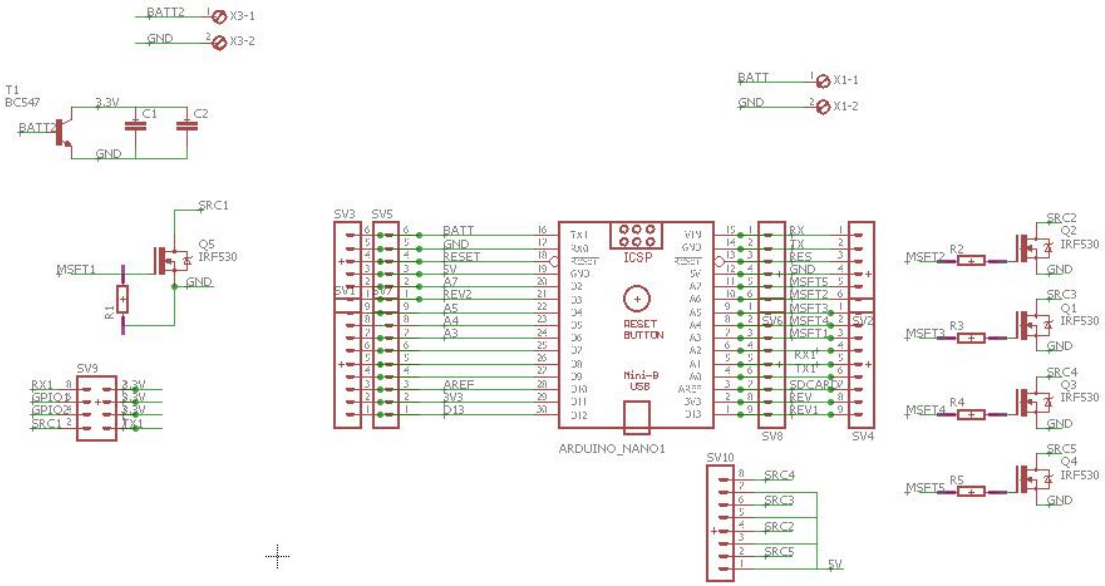


Figure 4.2: PCB Wire Diagram

Both Arduino Nano and ESP8266 will be mounted on the same PCB. Moreover, ESP8266 consumes a significant amount of current (approximately 300mA) which is much more than Arduino Nano and the sensor modules. Thus, ESP8266 is required to power by a separated battery in order to reduce the power consumption from the battery that powered Arduino Nano.

Besides that, Arduino Nano is powered by 3.7V 18650 battery via a MT3608 boost converter to increase the battery voltage. Furthermore, HT7833 3.3V voltage regulator is used to reduce 3.7V to 3.3V in order to power the ESP8266 module. Based on datasheet, this voltage regulator is capable of delivering 500mA at 3.3V which provide sufficient current for ESP8266 to function normally. Apart from that, this regulator also operates at a low dropout voltage which is suitable for battery-based application. All other features of the regulator can be obtained from Appendix A, Figure A-2.

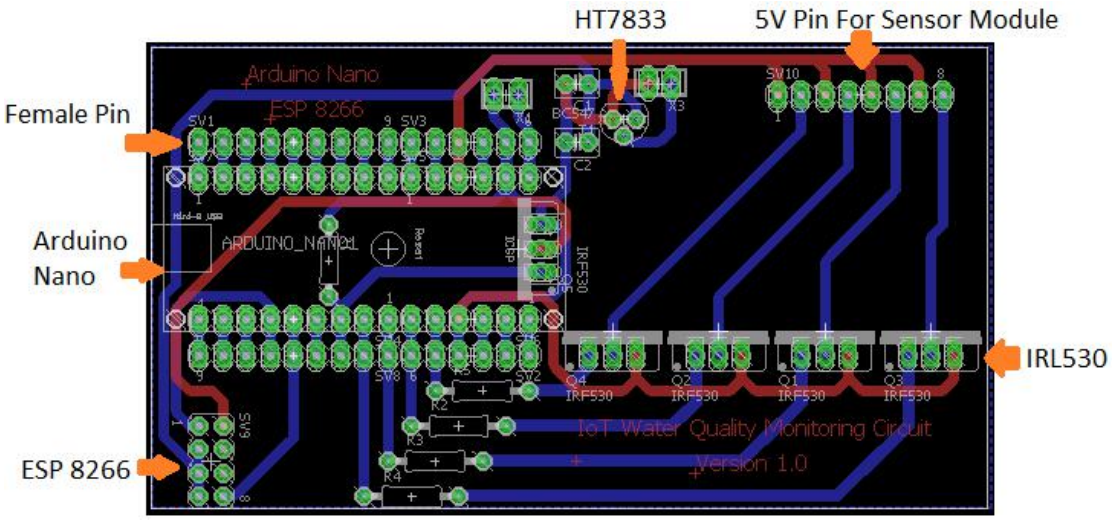


Figure 4.3: PCB Layout

Based on Figure 4.3, the red line represents the top layer and the blue line represents the bottom layer. This view shows all the connections and the arrangement layout for each component. Next, Figure 4.4 below shows the top layer and the bottom layer of the PCB design. The total area of the PCB fabricated is 5cm by 9cm which makes PCB to be able to fit in perfectly into a 20cm by 12cm junction box together with other components such as sensor modules and batteries. Furthermore, Figure 4.5 shows the final result of PCB with all the components soldered inside and Arduino Nano shield laying beside the PCB.

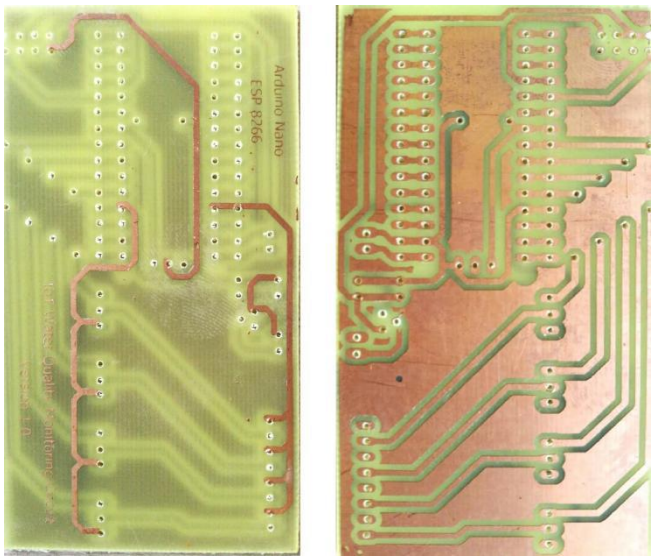


Figure 4.4: PCB Top View (left) & Bottom View (right)

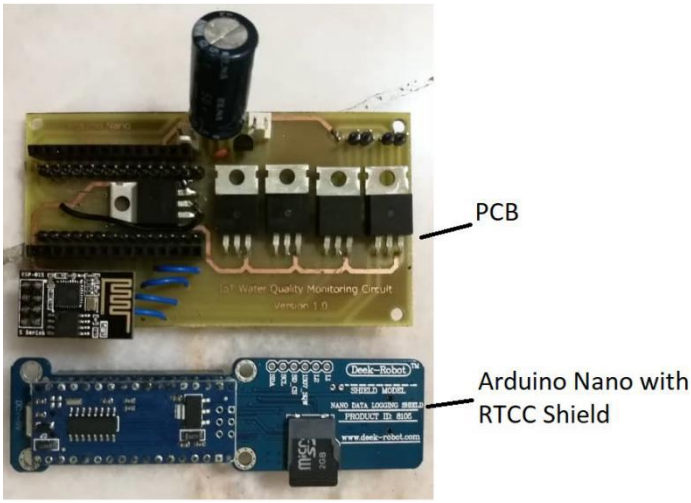


Figure 4.5: Soldered PCB (top) and Arduino Nano Shield (bottom)

4.1.3 Power System Architecture

The circuit will be powered by two independent single 18650 battery and the battery is charged by 20W solar panel. Furthermore, the board that is used to charge the battery is the TP4056 charging module. This board is important to prevent the battery from overcharge and over-discharge which can damage the battery and reduce the life span of the battery. Apart from that, this charging module also able to deliver up to 1A charging current. Since 18650 battery only output 3.7V, a boost converter is added to increase the voltage to 5V before delivering to Arduino Nano and the sensor modules. The battery used to power ESP8266 will be converted down to 3.3V by HT7833 3.3V voltage regulator as ESP8266 operates at 3.3V. Figure 4.6 below shows the complete power system architecture for this project.

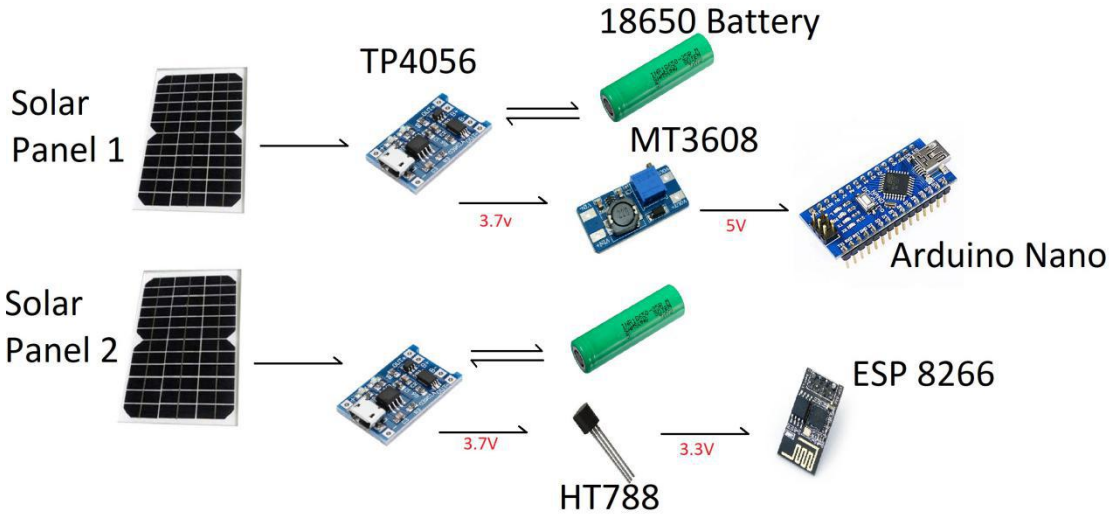


Figure 4.6: Power System Architecture

4.2 Software Development

4.2.1 Arduino Nano

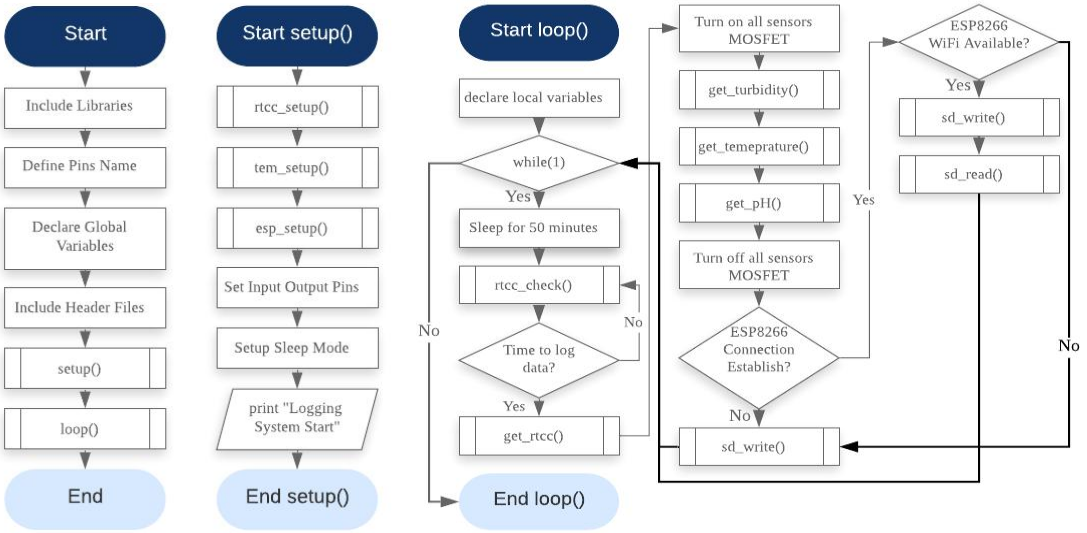


Figure 4.7: Arduino Nano Main Program Flow Charts

Figure 4.7 shows the main program of Arduino Nano where all the data logging algorithm is executed. The code starts with including all necessary libraries such as SD Card, RTCC and other libraries (refer to code in the Figure C-1-1 in Appendix C). After including all the libraries, all pins are defined followed by declaring global variables and include all header files such as RTCC.h, temperature.h, and other header files. The microcontroller will call the first function which is setup(). The setup() function is to set up all modules interface with the microcontroller, settings up the configuration for the microcontroller and defines input-output pins. After setting up the microcontroller, the main code will run in the loop() function. In the loop function, it will define the local variables to be used in the function and go into an infinite loop which is “while(1)”. By doing so, the variable does not need to be re-declare every time the loop finish. Next, in the infinite loop, the microcontroller is set to sleep for 50 minutes and wake up and check rtcc time by calling rtcc_check() function. By using ATMEga328p, it can only sleep for 8 seconds, so, in order to sleep for 3000 seconds (50 minutes) a “for loop” is used to loop the sleep function for 375 times. After the microcontroller awake, it will continue communicating with the RTCC module to check if 1 hour have reached. After 1 hour has reached, the microcontroller will proceed to get the current date and time and store it into an array

and turn on all the MOSFETs for the sensor modules. Next, the array will be passed to every sensor's functions (get_turbidity(), get_temperature() and get_ph()) to get the value for the respective water parameter. After logging, the MOSFET will be turned off and the data will be stored in SD Card before sending it to ESP8266 for transmitting the data to cloud. The data is stored in SD Card before sending is to ensure all the unsuccessful transmitted data is transmitted before the new set of data being transmitted to the cloud. Moreover, Arduino Nano will communicate to ESP8266 via UART and ask if connection establish and WiFi is available. If one of the connections fails, the data will be stored into SD Card. Finally, after completion of all logging and transmitting process, the microcontroller will go to sleep for another 50 minutes before repeating the same process. The overall code used 22383 bytes of program space and 1345 bytes of dynamic memory of the microcontroller.

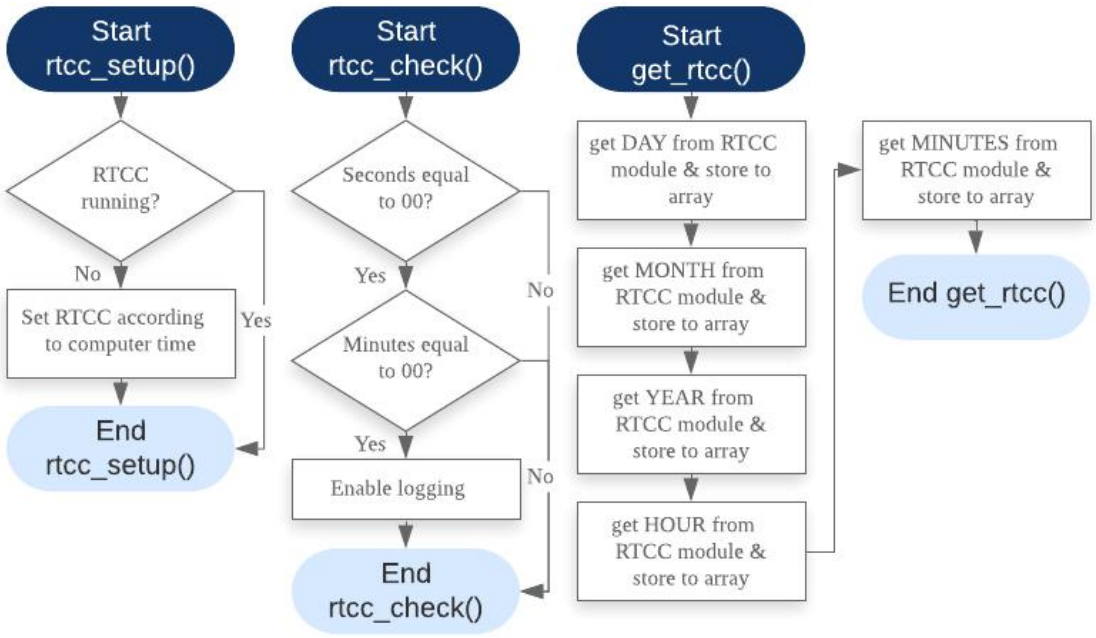


Figure 4.8: RTCC Flow Charts

Figure 4.8 above shows 3 flow charts under RTCC header files. The first flow chart from left is rtcc_setup() function. This function is to set up the RTCC if the clock is not running during upload the code to the microcontroller. Next, rtcc_check() function is to check if 1 hour have passed. RTCC module will only enable Arduino Nano to log when seconds and minutes are at 00 and 00 at the same time. Lastly, get_rtcc() function is to store date and time value to an array. The function will store day follow by month, year, hour and minute before ending the loop.

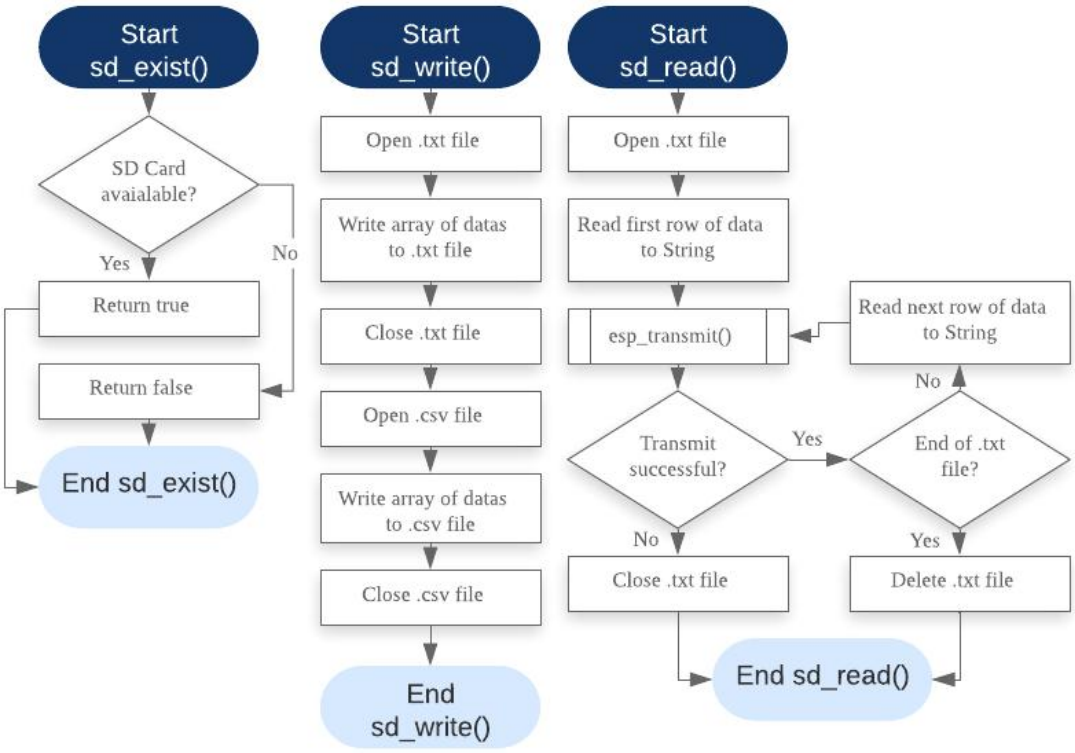


Figure 4.9: SD Card Flow Charts

Next, Figure 4.9 shows the flow charts for the SD Card header file. SD Card flow charts consist of three flow charts which are sd_exist(), sd_write and sd_read() respectively. The sd_exist function is used to check if the SD Card exists and will return true or false based on the result. The next function sd_write is to perform store the array of data logged in the SD Card where .txt and .csv files will be created. Two files are created because .txt is the file that store data that fails to transmit to the cloud where .csv is the file that uses for data analysis. The sd_read() function is to read from the SD Card and transmit the data to ESP8266. In this function, Arduino Nano will read the .txt file and transmit data row by row to ESP8266. If all data is sent, .txt file will be deleted. However, if the data transmission fails halfway, the transmission will stop and all the data will remain in .txt file. By doing so, this will ensure no data is lost during transmission.

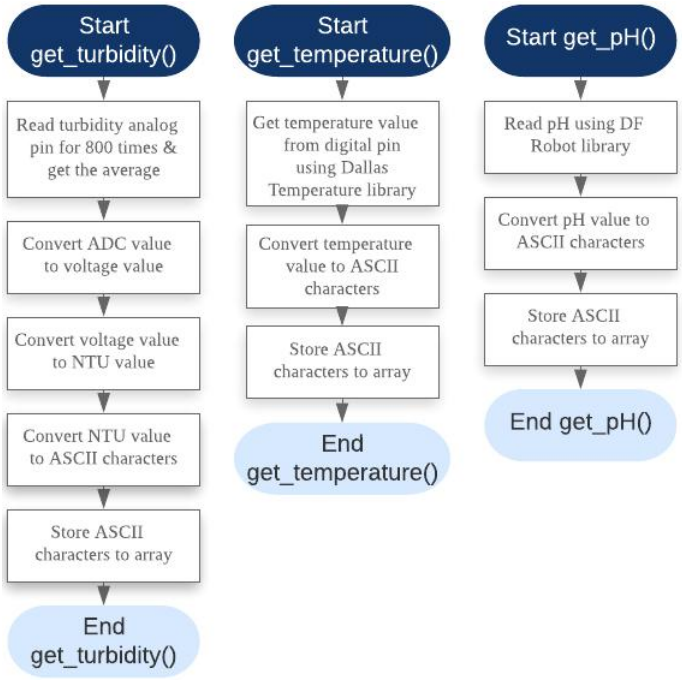


Figure 4.10: Sensor Flow Charts

Figure 4.10 above shows the flow charts for each sensor header files. From left, it shows the flow chart for the turbidity sensor, where the analog input for the turbidity sensor is logged 255 times and value is averaged to get the actual ADC value. Next, the ADC value is converted to voltage and the voltage will be converted to NTU. The conversion will be discussed in the later section of this report. Moreover, the NTU value is converted to ASCII character before storing to array. This is because UART is communicating in ASCII format and to ensure the communication is readable by the programmer during debugging, the value is converted to ASCII. Next, the temperature sensor value is captured by utilizing the Dallas Temperature library where no calculation is required to convert the raw data. Besides that, for pH sensor, the value is captured by using DF Robot pH library where the pH value can be obtained directly by using the library. Both temperature and pH values are also converted to ASCII before storing it to the array.

4.2.2 ESP8266

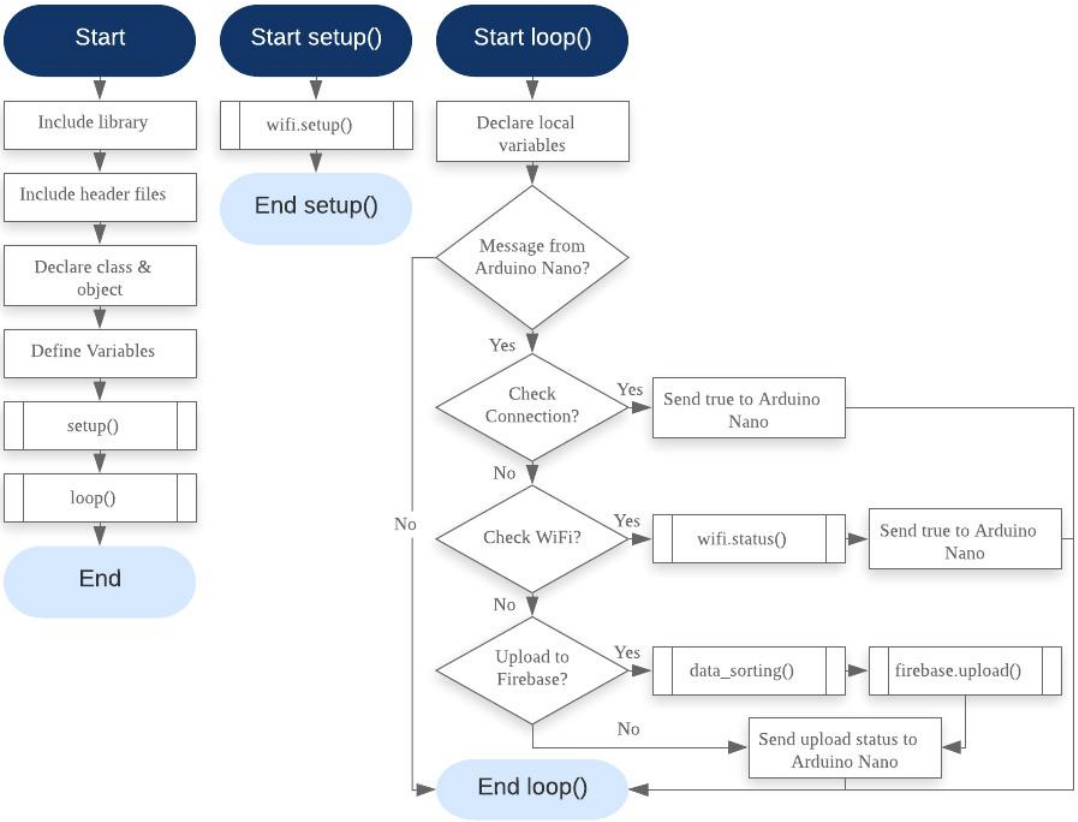


Figure 4.11: ESP8266 Main Program Flow Charts

Figure 4.11 above shows two flow charts for ESP8266 main program. The left flow chart is identical to the Arduino Nano flow chart where the configuration is done followed by calling two functions that are setup() and loop(). Next, the main loop for ESP8266 is executed to keep the waiting message from Arduino Nano. Upon receiving the message from Arduino Nano, ESP8266 will decode the message and go to the respective loop such as check connection, check WiFi or upload data to Firebase. Moreover, before uploading the data to Firebase, the data is sorted to break the array of ASCII string into individual categories such as date, time, temperature, turbidity, and pH. This algorithm is executed so that during the transmission of data to the cloud, the data will be upload to the correct branch at the cloud server. The overall program used about 467900 bytes of program space and 30176 bytes of dynamic memory.

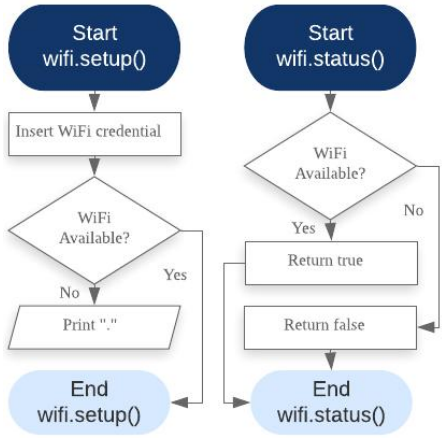


Figure 4.12: WiFi Flow Charts

Figure 4.12 above shows two short flow charts of the WiFi header file on ESP8266. The first flow chart (left) represents wifi.setup() where it connects ESP8266 to WiFi and checks if the WiFi is connected before proceeding. The second flow chart (right) shows the wifi.status() flow chart where it will check for WiFi connection and return true or false if this function is called.

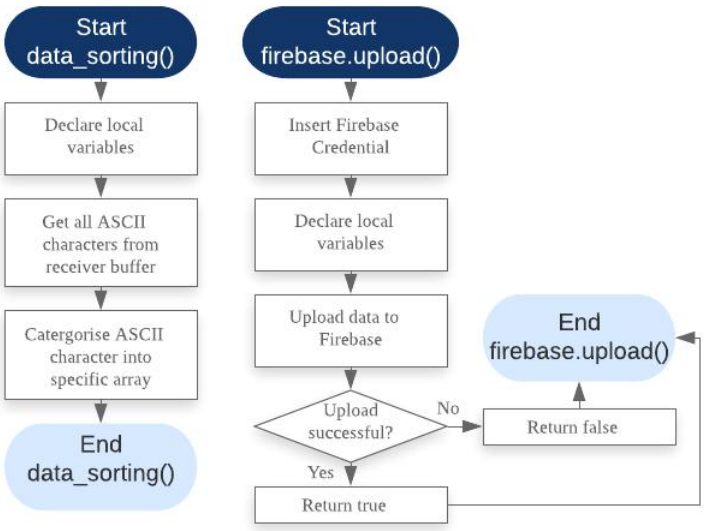


Figure 4.13: Data Sorting and Firebase Flow Charts

The two other functions in ESP8266 are data_sorting() and firebase.upload(). The data_sorting() function is to read the receiver buffer of ESP8266 and store it into an array of characters. The array of characters will later be categorized into different arrays according to date, time, turbidity, temperature, and pH. Moreover, for the firebase.upload() function, it is used to upload all the data to the Firebase cloud

server. Firstly, ESP8266 is required to set up the credential of the specific Firebase it wants to upload the data to. Next, EPS8266 will upload the data and return true if the upload is successful or else it will return false if upload is unsuccessful.

4.2.3 Firebase Functions

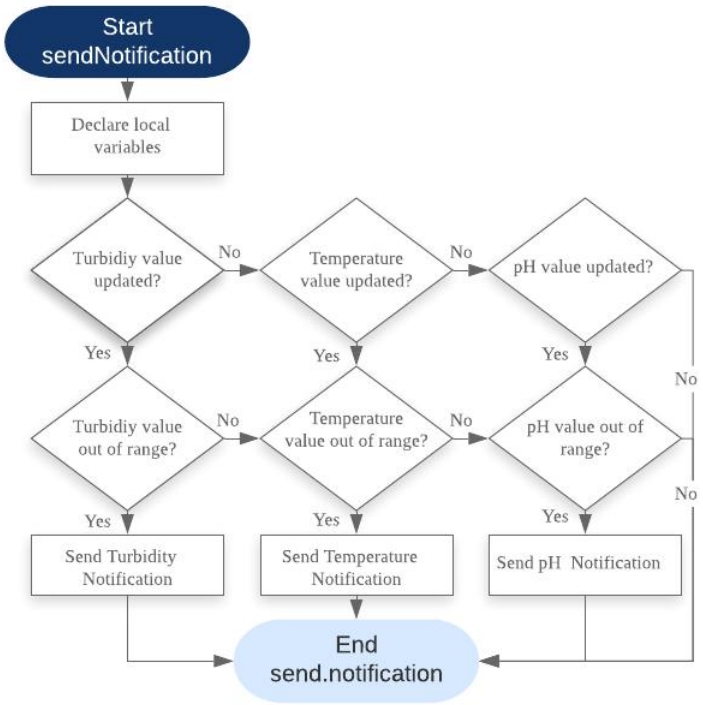


Figure 4.14: Firebase Function Flow Chart

Figure 4.14 above shows the flow chart of the code that runs at the Firebase server. The code is uploaded to Firebase Function and it will trigger once there is an update at the Firebase Realtime Database. The code starts with initializing local variables followed by checking each of the uploaded sensor's value. After any one of the sensor's value is above and below the initially defined range, Firebase Function will trigger Firebase Cloud Messaging to send a notification to the android mobile application.

4.2.4 Android Mobile Application

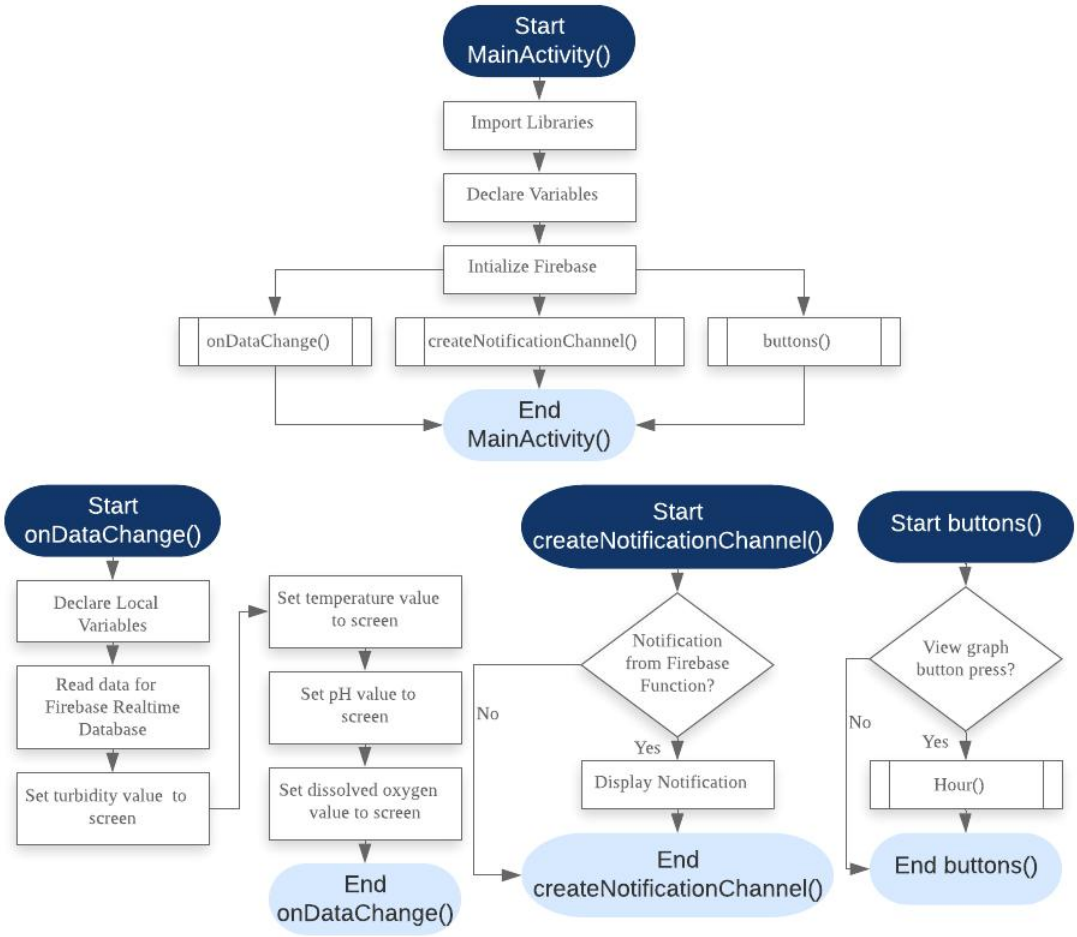


Figure 4.15: Android Mobile Application Main Page Flow Charts

The android mobile application has three pages, Figure 4.15 above shows the flow chart for the first page which is the main page. MainActivity() is the first function called by the microprocessor in the mobile phone. As usual, the first function import libraries, declare variables and initialize Firebase credentials. After done with configuration, the microprocessor will run three functions in parallel which are onDataChange(), createNotificationChannel() and buttons(). OnDataChange() function is to display the latest logged date, time and sensor's value on the screen, where else for createNotificationChannel() function is used to trigger the notification on the mobile device when there is any incoming notification from Firebase Cloud Messaging. The function buttons() will wait for the "view graph button" displayed on the screen to be pressed by user and after being pressed, it will call Hour() function to navigate the screen to second page.

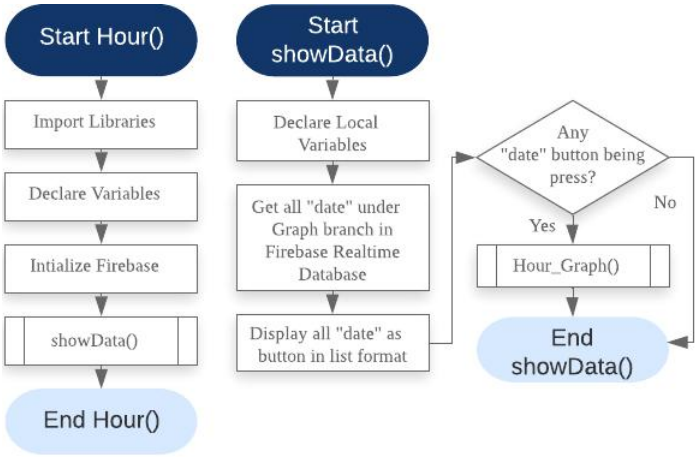


Figure 4.16: Android Mobile Application Hour Page Flow Charts

Figure 4.16 above shows the second page flow chart of the android mobile application. The function Hour() started by import libraries, declaring variables and initialize Firebase and finally call showData() function. The showData() function is used to retrieve all the logged date from the cloud server and display it in the list view. Users will be able to select the date in the list view to view the graph they wanted on a specific date. Once the user clicks on the date, it will call Hour_Graph() function and navigate to the third page of the mobile application.

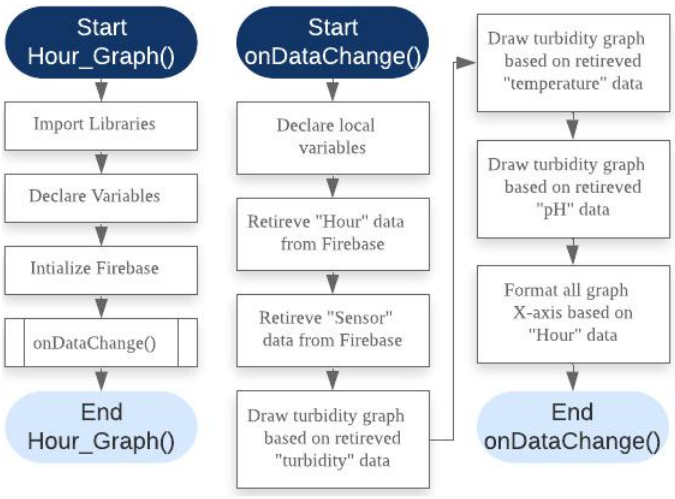


Figure 4.17: Android Mobile Application Graph Page Flow Charts

The last flow charts shown in Figure 4.17 above is the third page flow charts for the mobile application. The Hour_Graph() function starts by importing libraries, declaring variables and initializing Firebase before call onDataChange() function.

Next, onDataChange() function executes step by step sequence shown at the right flow chart to build the graph. The function starts by declaring variables, followed by retrieving the time from Firebase and its respective sensor's value. Next, each of the sensor's graph is drawn by the retrieved sensor's value on the Y-axis and retrieved time in the X-axis.

4.2.5 Sensor Calibration

In this project, there are only two sensors that require calibration which are the turbidity sensor and pH sensor. The turbidity sensor is calibrated by adjusting the surfaced mount potentiometer as shown in Figure 4.18 below.

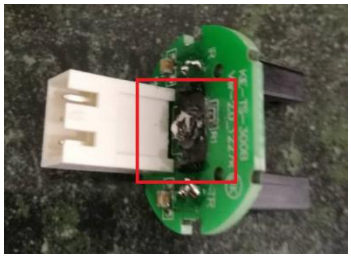


Figure 4.18: Potentiometer on Turbidity Sensor

Based on the red box shown in Figure 4.18 above, it shows the potentiometer soldered into the sensor. The potentiometer adjusts the resistor to limit the voltage delivered from the sensor to the microcontroller . In addition, by turning the resistor clockwise or anti-clockwise it will reduce or increase the resistance respectively. Moreover, the turbidity sensor output voltage is read by connecting the turbidity sensor to the Arduino Nano. Next, the Arduino Nano will read the voltage from analog input to perform ADC conversion. The converted ADC value will then be converted to voltage based on the following equation:

$$V_{IN} = \frac{V_{REF} \times ADC}{1023} \tag{4.1}$$

where,

V_{IN} = Input Voltage, V

V_{REF} = Reference Voltage, V

ADC = Converted Input Voltage

From Equation 4.1, the V_{IN} is the voltage from the turbidity sensor and can be calculated by multiplying V_{REF} by 1024 or 1023 (10-bit microcontroller) and divided by ADC value captured by the microcontroller. The calculated V_{IN} can be placed into the following equation:

$$NTU = -1120.4(V_{IN})^2 + 5742.3(V_{IN}) - 4352.9 \tag{4.2}$$

where,

V_{IN} = Input Voltage, V

NTU = Nephelometric Turbidity Unit, NTU

Equation 4.2 is derived by the turbidity sensor supplier (DF Robot) of the turbidity sensor by using the following graph shown in Figure 4.19 below.

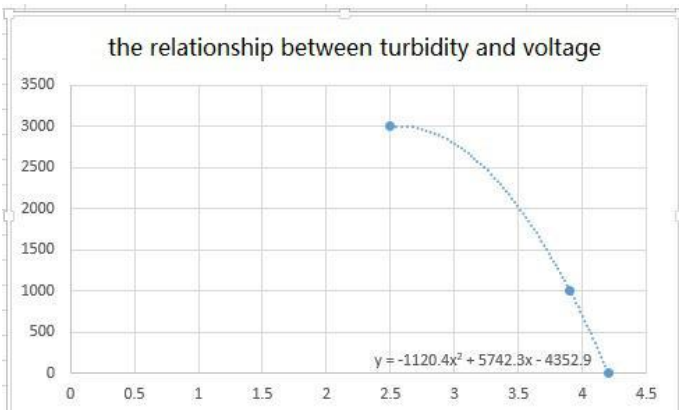


Figure 4.19: Graph of Turbidity versus Voltage

Hence, by looking at the graph in Figure 4.19, the output voltage from the turbidity sensor is adjusted until approximately 4.25V at the clear water and the calculated value for NTU should be 0. This shows the calibration method used for the turbidity sensor in this project.

Next, the pH sensor has two methods to perform calibration. The first method is to use the DF Robot library to perform calibration and the second method is using 2-point calibration. By using the DF Robot library, it provides auto-calibration and temperature compensation to the output pH value. Furthermore, standard 2-point calibration requires to form a linear equation using the ADC value measured by two

different pH values. Equation 4.3 below is used to form the linear equation by substituting the two values get from two different pH value which is pH 4 and pH 7.

$$y - y_1 = -m(x - x_1) \tag{4.3}$$

$$m = \left| \frac{y_2 - y_1}{x_2 - x_1} \right| \tag{4.4}$$

where,

x_1, y_1 = X_1 and Y_1 coordinates

x_2, y_2 = X_2 and Y_2 coordinates

m = gradient

From Equation 4.3 and 4.4, x and y represent coordinates and m represent gradient of the graph. Based on the ADC value measured for pH 4 and pH 7 respectively, the result is 470 and 353 respectively. By converting both raw value to voltage using Equation 4.1 with V_{REF} as 5V, the result of conversion is 2.29V and 1.73V for pH 4 and pH 7 respectively. Next, by substituting y_2 as 4, y_1, as 7, x_2 as 2.29 and x_1 as 1.73 to Equation 4.3 the resulting gradient is 5.36. After getting the gradient, pH 7 and 1.73V is substitute into Equation 4.3 along with 5.36 as gradient which result in the following equation.

$$\begin{aligned} y - 7 &= -5.36(x - 1.73) \\ y &= -5.36x + 16.27 \\ pH &= -5.36(V_{IN}) + 16.27 \end{aligned} \tag{4.5}$$

where,

V_{IN} = Input Voltage, V

pH = pH value, pH

Hence, by looking at the resultant linear Equation 4.5, pH value can be calculated by substituting V_{IN} which is the input voltage from the pH module.

4.3 Project Prototype

4.3.1 Electronics Mounting

In order to put all the electronic components in the boat, a 20cm × 12cm junction box was purchased. The junction box is used because it is waterproof and can protect all the electronics from water damage due to rainfall and water splashed around the prototype. Two battery holder has an area of 4.5cm × 7cm and the PCB has the area of 5cm × 9cm. Both of these fit perfectly in the junction box leave 10.5cm × 12 cm of area for the sensor modules. Figure 4.20 below shows the final result of the junction box after all the components are fitted in.

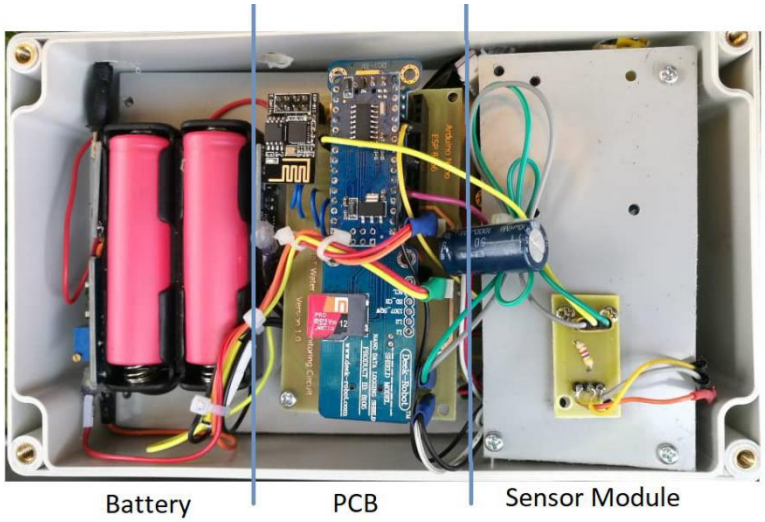


Figure 4.20: Complete System In Junction Box

Based on Figure 4.20, the first column, second column, and third column consist of battery components, PCB components, and sensor modules respectively. On the first column in the junction box, it consists of 2 batteries for ESP8266 (left) and Arduino Nano (right) in the PCB compartment. TP4056 is mounted on each side of the battery holder and a single MT3608 boost converter for Arduino Nano is also mounted on the right side of battery holder . Next, the second column consists of the PCB with Arduino Nano and ESP8266 mounted on top of it. Finally, all the sensor modules are placed on the third column of the junction box. The yellow PCB shown in Figure 4.20 is temperature sensor module and the other 3 sensors modules are placed under the grey coloured board.

Next, in order for the sensors wire to pass through the junction box, six different size of holes are drilled on the side of the junction box, depending on the wire size. Four holes are drilled for pH sensor, turbidity sensor, temperature sensor, and dissolved oxygen sensor while the other two holes are drilled for cable that charge the battery from the solar panel. In order to prevent water from entering the junction box via the drilled holes, the holes are sealed with duct tape both inside and outside of the box.

4.3.2 Solar Panel and Junction Box Mounting

Figure 4.21 below shows the design of the solar panel and junction box mounting. The prototype is designed using Solidworks software.

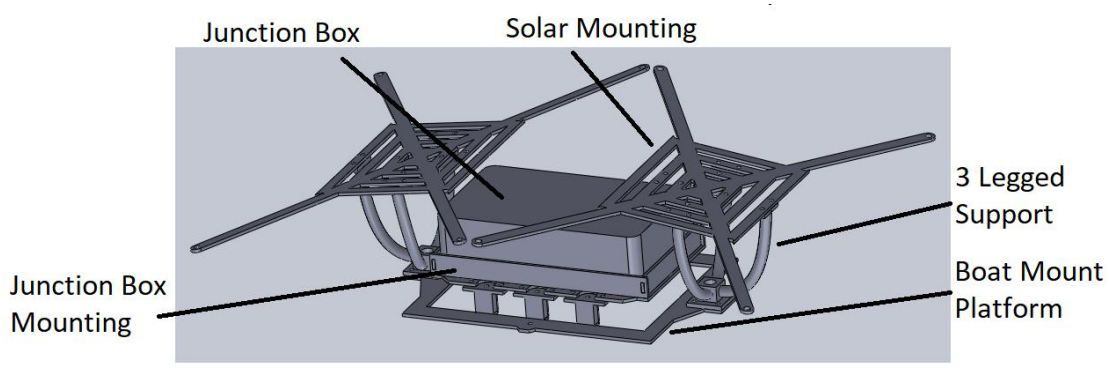


Figure 4.21: Solar Panel and Junction Box Mounting Design

The mounting design is based on the purchased boat measurement. All dimension is designed based on the size of the purchased boat so that it fits perfectly on top of the boat. Based on Figure 4.21, the junction box is designed to be mounted at the center of the boat with both solar panels shading the mounting box to prevent sunlight shining on the junction box which will cause heat building up inside the box. Moreover, the solar mounting is designed to be slanted approximately 8 degrees so that rainwater will not puddle on top of the solar panel. All of the mountings are made out of clear acrylic and cut by a C02 laser cutter from the mechanical workshop. However, the 3 legged support shown in Figure 4.22 below is made out of PLA material printed by 3D Printer in 3D Printer laboratory. In order to secure the solar panel better, 10mm aluminum shaft is also added to the 3 legged support as shown in Figure 4.22.

Figure 4.22: Three Legged Support for Solar Panel

4.3.3 Sensor Housing

The sensor is placed at the front of the boat. All of the sensors are supported by sensor housing to prevent the sensors from hanging and moving around the boat. The sensor housing is made from two materials which are scrap metal sheets gathered from the mechanical workshop and plastic netting. The metal sheet was cut, bend and holes are drilled so that the shaped metal sheet can join by nuts and bolts. The final product of the sensor housing can be seen in Figure 4.23 below.

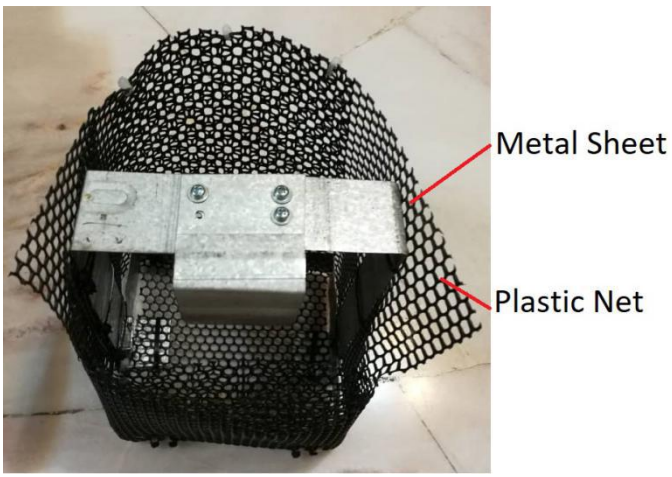


Figure 4.23: Sensor Housing

4.3.4 Final Prototype

After all of the parts are finalized and built, the assembly begins by placing the sensor mounting at the front end of the boat followed by inserting the mounting on top of the boat. Next, the junction box is placed on top of the mounting and all the sensor are placed inside the sensor housing. The final assembly is mounting the solar panel on the acrylic and the prototype is ready for field test. The final prototype of

this project is shown in Figure 4.24 and Figure 4.25 below for front view and side view respectively.

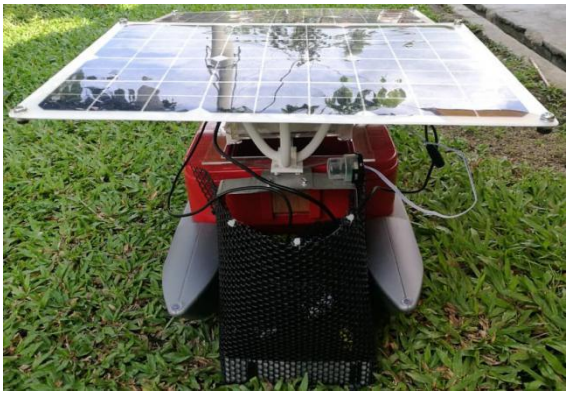


Figure 4.24: Final Prototype Front View

Figure 4.25: Final Prototype Side View

4.4 Features and Performance of Prototype

4.4.1 Data Storage

In this system, all the data logged will be stored in both the SD Card and the cloud server. The data stored in the SD Card will be in the Comma Separated Value (.csv) format. This format is chosen because Arduino Microcontroller is unable to read and write excel formats such as .xlsx. In addition, .csv format will be able to open by using excel for further data analysis such as plotting the graph and perform mathematical calculations. However, the drawback of this method is whenever the user wanted to take the data for analysis, the user will need to go the prototype and retrieve the SD Card in order to view all the data collected. The SD Card chosen has a storage capacity of 2 gigabytes, where, the data logged for every 10 minutes it will occupy approximately 5 kilobyte for one day. Thus, it will take about 400,000 days of data to fully utilize the space of the SD Card.

Next, Firebase provided by Google is the cloud server used to store data for this project. Firebase does not store the data in the Structured Query Language (SQL) format. Firebase is a Non-SQL database where all the data is stored in JavaScript Object Notation (JSON) format. Figure 4.26 below shows part of the data stored in the Firebase Realtime Database. The data is stored in a nested loop under Farm 1 as the master branch. One master branch represents only one system is active. For future expansion, the second master branch can be created to represent the second system. Under the master branch, it has two child branch which are Graph, and Notification. Graph branch stores all the database logged by the water quality monitoring system, whereas notification branch stores the latest data logged by the system.

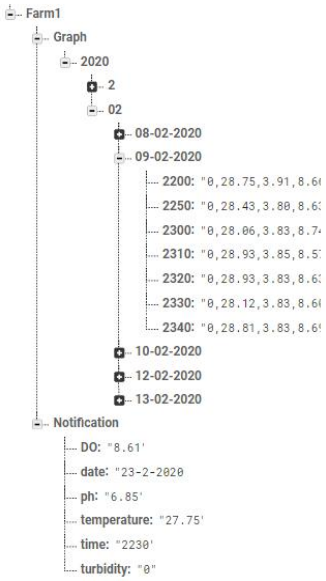


Figure 4.26: Firebase Realtime Database

Firebase is chosen because the data storage is free up to 1 gigabyte of space. It is able to link with Android devices directly, and it is able to run JavaScript on the server in order to process the data and perform real-time notification. Moreover, one day data took about 5 kilobytes of space, in order to fully utilize the space of Firebase Database it requires 2,00,000 days of data. Based on Figure 4.26, the nested loop is created in such a way because it is easily expandable. For example if there is another system introduced to the database, it can be easily separated into another nested file.

4.4.2 Data Monitoring via Computer and Mobile Phone

The data can be monitored by three different methods, the first and second method are via computer and the third method is via mobile phone. In order for the user to view the data via a computer, the user can manually get the data from the SD Card on the farm to view all the logged data. However, this method defeats the purpose of this project and it is inconvenient for the user to go to the farm to get the data. So the second method to monitor the water quality can be done by login to Firebase to view in JSON format as shown in Figure 4.26 earlier. The data will be updated in real-time based on the log interval set earlier.

Next, via mobile phone, the user will be able to experience GUI for a more friendly interface instead of rows and columns of data in Excel or JSON format in Firebase. Moreover, via mobile phone the user only able to view the last log data and the historical data in graphical format. Figure 4.27 below shows the main page, date selection page, and graph page of the mobile device.

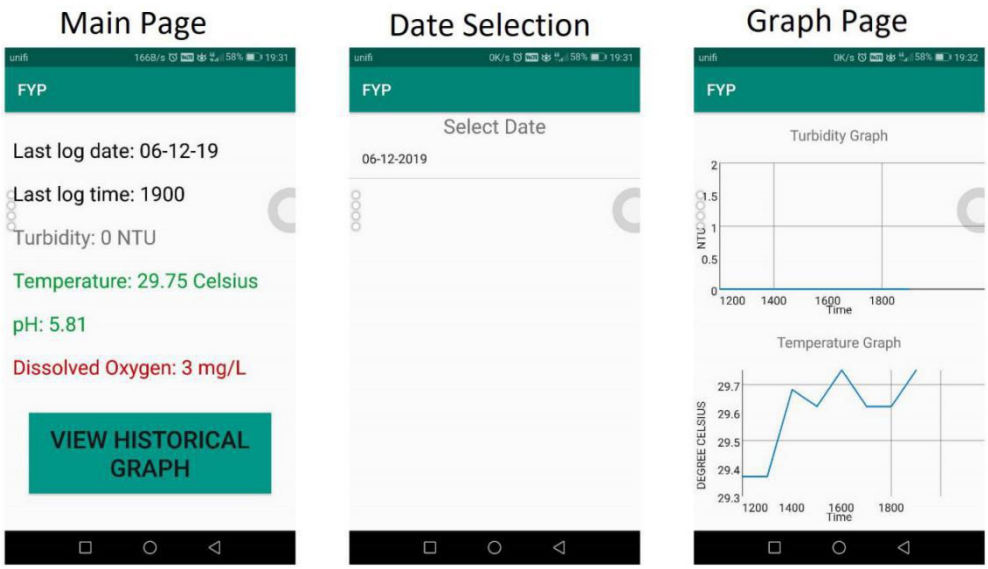


Figure 4.27: Android Mobile Application Layout

Based on Figure 4.27 above, the main page shows the latest information about the water quality parameters such as turbidity, temperature, pH and dissolved oxygen logged at a specific date and time. Moreover, by pressing the view historical graph button it will navigate to the second page which is the date selection page. At this page, the user will be able to choose the respective date they would like to view

the data. By clicking on the date, the mobile application will navigate to the graph page and display the logged value for each sensor.

4.4.3 Real-Time Notification

The real-time notification for this prototype is via a mobile application. When the system logged the water parameters to the Firebase Realtime Database, Firebase Function will trigger a JavaScript program to screen through all the latest water parameters. When the water parameters are out of range, Firebase Function will trigger a push notification to the user mobile devices and alert the user about the specific water parameter that is out of range. Figure 4.28 below shows the push notification for a water temperature notification that appears in user mobile application once it rises above 35 degree Celsius.

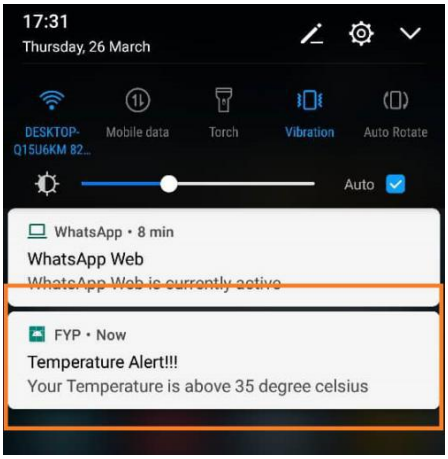


Figure 4.28: Push Notification

In addition, when the user is looking at the mobile application and the data is logged at the same time the latest water parameter value will turn red if the water parameter is out of range. This will alert the user when they look at the latest logged data. Figure 4.29 shows the result of water temperatures that are out of range.

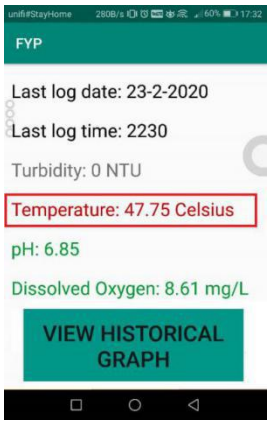


Figure 4.29: Out Of Range Display

4.5 Problem Encountered and Improvement During and After Field Test

4.5.1 First Location Field Test

Figure 4.30: Field Test at Taman Tasik Danau Kota

Figure 4.30 above shows the first field test with the prototype at Taman Tasik Danau Kota located at Kuala Lumpur. The test was started at 11 am and ended at 2 pm before the sky turned dark. There are several problems encountered during the first field test such as software and the prototype.

The main problem with the code is the conversion algorithm for pH sensor was wrong and unable to trace back the raw ADC data as the ADC data is converted to American Standard Code Information Interchange (ASCII) character before store into the SD Card. The pH library used conversion algorithm provided by DF Robot since the sensor module used in this project is provided by the same company. During calibration, the sensor was able to work correctly, however during the field

test the output value from the sensor returns the wrong value. This problem is countered by getting rid of the pH library and remove the conversion of value from integer to characters before storing it into the SD Card. This method is applied to another sensor such as the turbidity sensor. Moreover, the raw data will be transmitted to the Firebase cloud server and all the data will be processed in the server to convert the raw data to the actual measurement value. By doing so, if there are is any problem with the conversion all of the errors can still be traced back by looking at the raw data.

Figure 4.31 shows the changes made to the Firebase Realtime Database where a new branch is added which is “RawData”. This branch is to store all the raw value logged by each and every sensor and will later be processed by Firebase Function in the cloud server. The converted data will later be stored into “Graph” branch and “Notification ” branch. Since the mobile application will only pull data from “Graph” branch and “Notification” branch so no addition changes will be made to the cloud server.

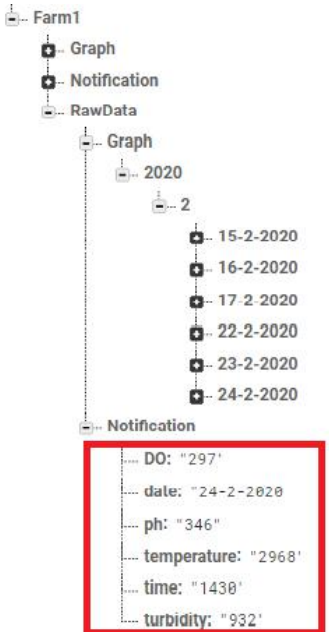


Figure 4.31: “RawData” Branch

Next, there are several problems with the prototype. The main problem is the weight of the sensor housing. By using the metal sheet as the main structure of the housing, it contributes a lot of weight to the front of the boat causing the boat to sink in the front as shown in Figure 4.32.

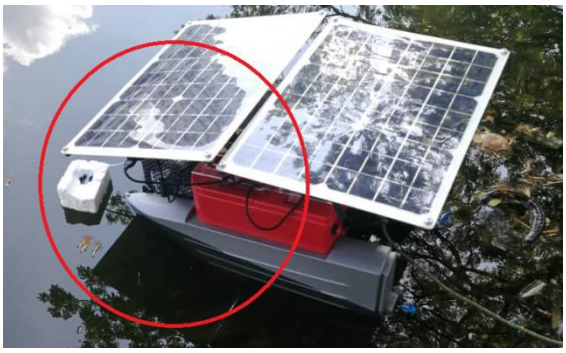


Figure 4.32: Wrong Weight Distribution in Prototype

This problem is encountered by changing the sensor housing design and make the sensor housing to be independent of the body of the boat. This solution will allow all the sensors to float independently with the boat. Hence, it will not cause additional weight to the front of the boat. Figure 4.33 below shows the new sensor housing design made by acrylic and plastic bottles.

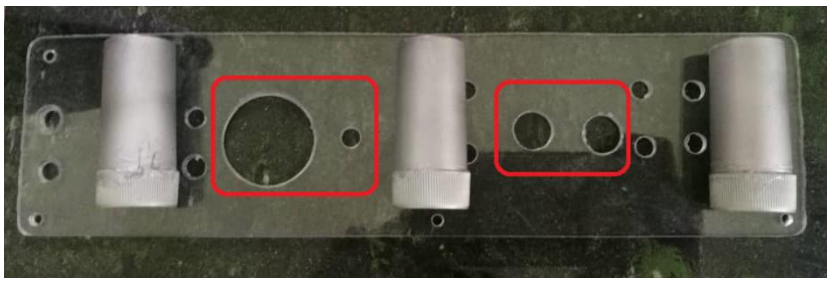


Figure 4.33: New Sensor Housing Design

The new sensor housing design consists of 4 holes to hold the sensors in place. The red box on the left shows the hole for the turbidity sensor and temperature sensor while the red box on the right shows the hole for both pH sensor and dissolved oxygen sensor. The rest of the holes are for cable ties to hold the plastic bottle in place to act as a floater for this sensor housing.

In addition, the left floater of the boat also leaking causing water to enter into the floater after a few hours placing the boat into the lake. The leakage will cause the boat to sink in one direction which will eventually cause the boat to sink after a long run. After several leakage testing by placing the boat in a big tub, there is no bubble coming out from the floater. Thus, it is unable to determine the leakage area. To solve this problem, an additional floater is added to the center of the boat as shown in Figure 4.34.

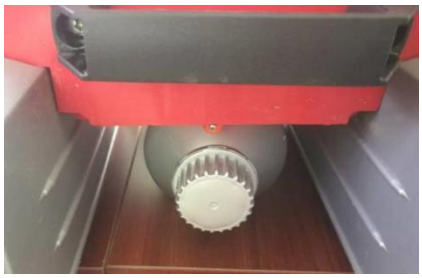


Figure 4.34: Additional Boat Floater

4.5.2 Second Location Field Test

After solving the coding and prototype problem, the prototype is sent for the second field test. The second field test was conducted at Broga, Negeri Sembilan aquaculture fish farm. The farm consists of 3 different ponds which categorize as clean, moderate clean and dirty ponds. All of the water comes from the upstream river flowing nearby where a pump is used to pump the water into the first pond (clean pond) and the water will be reused for the second pond (moderate clean) and finally the last pond (dirty pond). This prototype is placed on the first pond as shown in Figure 4.35 below.

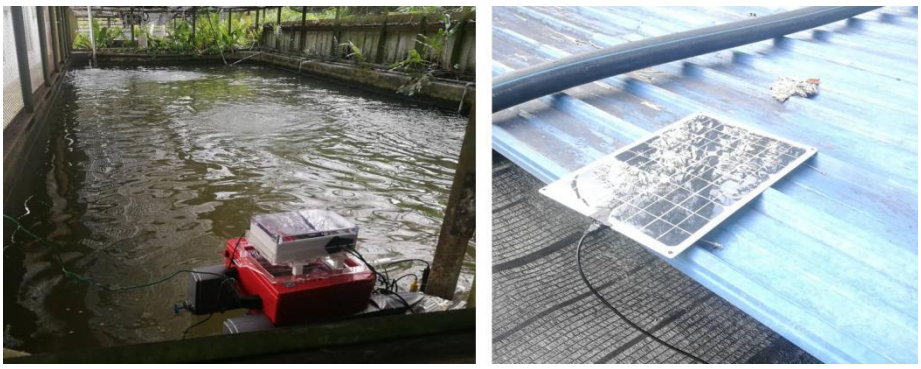


Figure 4.35: Field Test at Broga, Negeri Sembilan

Based on Figure 4.35 above, the left image shows the prototype being deployed to the pond and the right image shows the solar panel is placed on top of the roof. The solar is placed on top of the roof because the pond is covered by netting which greatly reduces the sunlight intensity in the pond so. In order to get a good amount of sunlight, the solar panel is removed from the boat and placed on top of the roof. Moreover, there is only one entrance in this pond so the boat can only placed nearby the gate of the fish pond. There are also several problems encountered throughout the period of data logging in the second field test location. During the

first week of data logging, the sensor are clogged with dirt as shown in Figure 4.36 below.

Figure 4.36: Sensors Covered With Dirt

During the first week, the water in the pond was quite dirty where the bottom of the pond cannot be seen. Moreover, the cleanliness of the pond depends on the cleanliness of the river nearby. Besides that, the sensor was placed in static water this causes the dirt to pile accumulate at the sensor. To solve this problem, the sensor was placed nearby the outlet of the water pump where the water velocity is high enough to clean and prevent the sensor from accumulating dirt.

Next, the pH sensor reading still outputs the wrong raw value even without performing any conversion after gathered the data. Based on Figure 4.37 below, it shows the screenshot of a video where the pH module requires about 1 minute to 2 minutes to stabilize the reading after it turns on by the MOSFET.

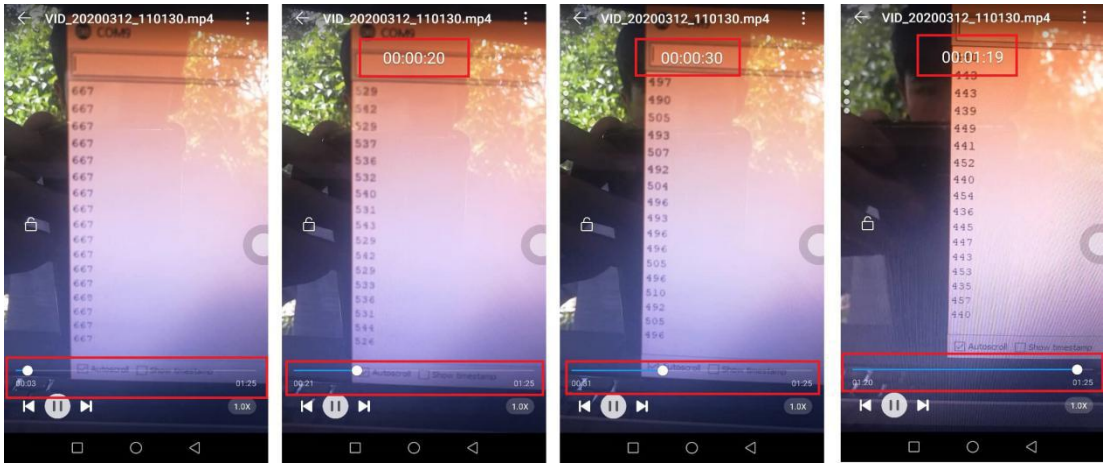


Figure 4.37: pH Sensor Raw Value

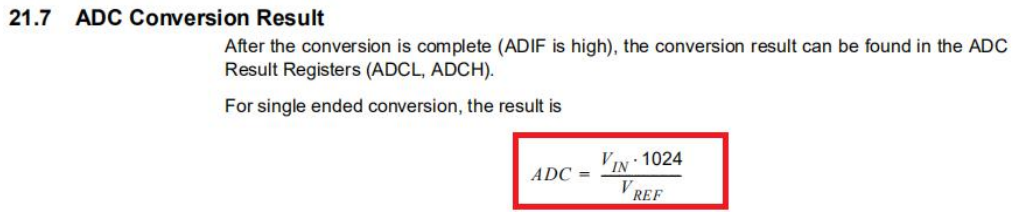
From Figure 4.37 above, the first image from the left shows the time at 3 seconds, the raw value from the pH sensor is about 667. Moving towards 20 seconds

at the second image from left, the raw value drops to approximately 540, followed by the third image at 30 seconds the raw value fluctuates about 490. Finally, the value stabilizes around 440 at 1 minute and 19 seconds. This proves that the sensor requires more than 1 minute in order to capture the actual pH value of the water. This problem is solved by turning on the sensor module to run for 24 hours to prevent a similar problems occur in the future.

4.5.3 Additional Improvements

There are several additional improvements were done in this project such as improve the sensor accuracy, adding the "log now" button, let the user choose the logging interval, add a minimum and maximum line in graph and make the graph more interactive to the user.

The sensor accuracy is improved by getting the voltage value at A_{REF} of Arduino Nano. As mentioned earlier, Arduino Nano is powered by battery via a boost converter, once the battery drains the value of boost converter will fluctuate by 0.1V to 0.2V. Moreover, based on actual measurement, the input voltage of Arduino Nano is set to 5V but the voltage at the A_{REF} pin of Arduino Nano is less than 5V. This will result in measurement error during converting the raw value to the actual measurement value. This is because, based on page 261 in the datasheet for ATmega328p which is the microcontroller used for Arduino Nano board, it states that the ADC (raw value) is calculated by multiplying V_{IN} (voltage from the sensor) with 1024 (10-bit microcontroller) and divide by V_{REF} (voltage at A_{REF} pin). Figure 4.38 below shows a snapshot of the ATmega328p microcontroller datasheet.



21.7 ADC Conversion Result

After the conversion is complete (ADIF is high), the conversion result can be found in the ADC Result Registers (ADCL, ADCH).

For single ended conversion, the result is

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

Figure 4.38: ATmega328p Microcontroller Datasheet Snapshot

By referring to the datasheet, in order to get a more accurate voltage from the sensor, the voltage of V_{REF} must be known. V_{REF} voltage can be obtained by measuring the voltage at the A_{REF} pin of Arduino Nano. A_{REF} can be measured by

activating bandgap voltage reference (1.1V) in the microcontroller and use the Equation 4.1 earlier to get the voltage at V_{REF}. The calculation below shows the sample calculation to get V_{REF} by taking the converted ADC value as 235 as an example.

$$V_{REF} = \frac{1.1 \times 1024}{ADC} = \frac{1.1 \times 1024}{235} = 4.8V$$

Next, by referring to Table 4.2 below, it shows the result of V_{REF} measured internally and using multimeter by manipulating the power supply voltage. 18650 battery fully charge at 4.2 V and stop discharging at 2.5 V which prevented by TP4056 module. By altering the voltage from 4.2V to 2.6V, it shows that the output of boost converter only changed 0.03V. Moreover, V_{REF} measured internally and using a multimeter is identical to each other at every test.

Table 4.2: Table of Boost Converter and V_{REF} Measurement.

Power Supply Voltage, V	**Boost Converter Output, V**	**V_{REF} measured internally, V**	**V_{REF} measured by multimeter, V**
4.20	5.00	4.79	4.79
4.00	5.00	4.79	4.79
3.70	5.00	4.79	4.80
3.50	5.00	4.79	4.79
3.40	4.99	4.79	4.79
3.00	4.99	4.79	4.79
2.80	4.98	4.78	4.78
2.60	4.97	4.78	4.78

The next improvement for this project is to increase the logging interval flexibility for the user by adding two additional features to the mobile application such as the user able to retrieve the water quality parameters at any time and the user will be able to change logging intervals. Figure 4.39 below shows the additional button which allows the user to use the additional features in the mobile application.

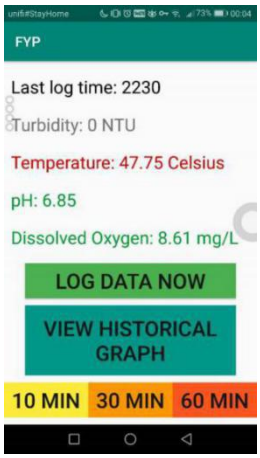


Figure 4.39: Additional Features In Mobile Application

Moreover, in Firebase Realtime Database, an additional branch needs to be created in order to deliver the message to Arduino Nano via a cloud server. Figure 4.40 below shows the additional "Others" branch added to the database.

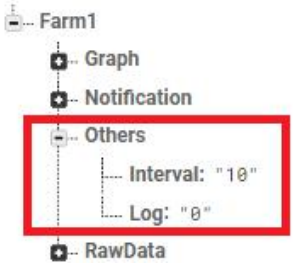


Figure 4.40: Additional "Others" Branch

Based on Figure 4.40, "Interval" represents the logging interval for the water quality monitoring system and "Log" represents whether the user wants to view the water quality parameters now. Whenever the user changes the logging interval, it will only take effect at exactly 12 am where the Arduino Nano will read the "Interval" node and change the logging interval according to the value under the "Interval" node (10 represent 10 minutes). Changing the interval at 12 am will avoid inconsistency in the X-axis of the graph if the user changes the interval at a different time other than 12 am. Next, the "Log" node value will change to 1 when the user pressed the "LOG DATA NOW" button as shown in Figure 4.39 earlier. Arduino Nano will continuous reading the value at "Log" node to perform logging once "Log" node value is turned into 1. However, the drawback of these features is Arduino Nano cannot put into sleep mode in order to ensure the logging system able

to respond in real-time. By increasing the capacity of the battery will solve this power consumption problem.

Finally, the last improvement is to improve the graph user experience. Figure 4.41 below shows an example of temperature and pH graph where the maximum and minimum lines are added to the graph for better visualization. Moreover, when the user pressed on the graph at every specific point, it will show the exact value of the point where the user pressed.

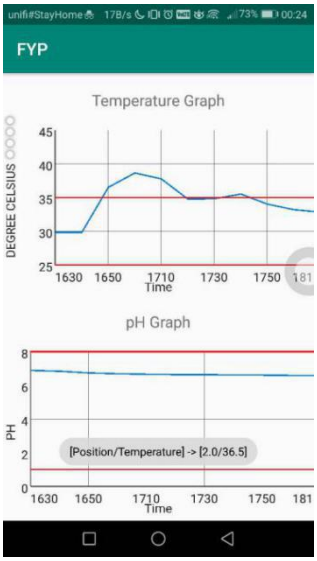


Figure 4.41: Improved Graph Features

4.6 Data Analysis

Throughout the field test from 4th March 2020 to 18th March 2020, there is a total of three water parameters being monitored which was turbidity, temperature, and pH. The interval was set to 10 minutes and the system was placed to monitor for 24 hours. All of the graphs can be found in Appendix B of this report.

Moreover, there is a total of five field trips made to the test site where the first field trip was to set up the water quality monitoring system at the fish pond, the second field trip was to gather data, the third field trip was to clean the prototype and gather data, the fourth field trip was to reprogram and re-calibrate pH sensor and finally, the last field trip was to bring the prototype back from the farm. The first field trip is made on 3rd March 2020 followed by 8th March 2020, 12th March 2020, 13th March 2020 and 19th March 2020.

Based on the graphs shown in Appendix B, some of the data may be missing

because, during the data collection, cleaning and reprogramming work was done on spot hence, the water quality parameters are not being logged for few minutes.

4.6.1 Turbidity

Figure B-1-1, Figure B-2-1, Figure B-3-1 until Figure B-15-1 in Appendix B show all the graphs logged by the turbidity sensor. Based on the graph, throughout the period of data logging, the water turbidity fluctuate around 2400 NTU to 3000 NTU. These numbers shows that there are quite a number of particles in the water and the number of particles varies from time to time as the water pump into the pond is coming directly from the upstream river. The cloudiness or haziness of the pond water is depending on the cleanliness of the river nearby. Figure 4.42 below shows the water colour on 3rd March 2020 the left and 13th March 2020 on the right.

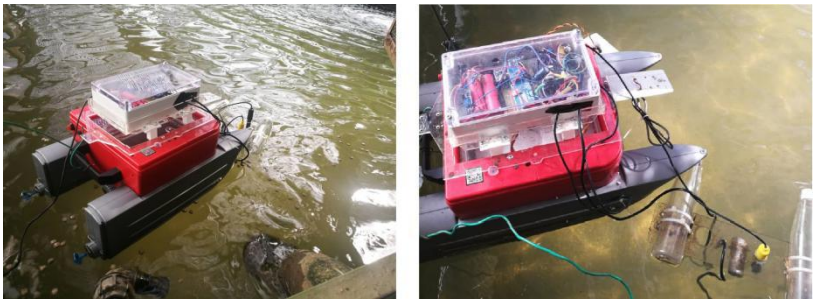


Figure 4.42: Water Colour On Different Day

The first data logged which is around 2800 NTU where the picture on the left shown in Figure 4.42 is taken while the water turbidity shown in the picture on the right is about 2350 NTU (based on graph Figure B-10-1). Moreover, Figure 4.43 below shows the colour of the pond taken during the last field trip on 19th March 2020.

Figure 4.43: Last Field Trip Photo

Based on Figure B-14-1 and Figure B-15-1 in Appendix B, the turbidity only fluctuated around 2900 NTU to 3000 NTU. This shows that on the 17th and 18th of March the water turned muddy and the overall particle has increased and stayed close to maximum.

Next, Figure B-9-1 in Appendix B shows an almost constant graph after 4 pm, that problem occurs because some water has entered into the turbidity sensor housing. The water might have short-circuited the circuit while entering which causes a turbidity sensor unable to log data.

4.6.2 Temperature

Figure B-1-2, Figure B-2-2, Figure B-3-2 until Figure B-15-2 in Appendix B show all the graphs for the pond temperature from the 4th of March 2020 to the 18th of March 2020. By looking at all of the graphs, the pond temperature produces a sine wave and it is able to be predicted easily compared to turbidity. From most of the graph, every 12 am in the midnight, the pond dissipates heat and the water temperature reduces until it is 8 am in the morning where the sun starts to warm up the river water causing the temperature of the pond water to rise. The temperature will rise usually from 8 am to 4 pm in the evening and reaches its peak. From 4 pm to 12 am the pond temperature either remains constant or reduces slightly. Moreover, the temperature of the pond only changed about 1 to 2 degrees Celsius.

Next, based on Figure B-8-2 to Figure B-9-2 in Appendix B, there is a problem with the data logged where the value remain constant at 25.8 degree Celsius. Apart from that, the graph is shown in Figure B-10-2 (from 12 am to 2:30 pm), which shows the water temperature is constant at about 14 degrees Celsius. The logged data is completely wrong and the problem started on 11th March 2020 around 7 am (Figure B-8-2). The temperature problem is only been figured out on 12th March 2020 after gathered all the data from the system and solve on 13th March 2020. This is because, the data was being analyzed at home not the test field. Thus, causing a few hours delay to correct the temperature sensor. There is no major issue with the temperature sensor during identifying the problem with the sensor. The only suspected problem might be a loose wire or bad soldering. After plugging in and out of the temperature sensor wire, the sensor able to work without any problem until the end of the field test.

4.6.3 pH

Figures B-1-3, Figures B-2-3, Figures B-3-3 until Figure B-15-3 in Appendix B show the pH value logged by the system. All of the data logged by the pH sensor before 2:30 pm on 13th March 2020 (Figure B-10-3) and after 12 am of 14th March 2020 (Figure B-11-3) were wrong due to several reasons. Before 2:30 pm on 13th March 2020, the problem state relates to turning on and off the MOSFET has not been discovered yet. This is because, during calibration, all of the sensors MOSFET are turned on and then perform calibration and the calibration process usually takes more than 2 minutes where the pH value has been signalized as discussed earlier. The problem had only been discovered during the on field calibration. Thus, all data before 2:30 pm on 13th March 2020 was wrong. In addition, the pH log fluctuated around pH 0 to pH 2 which is too acidic for the fish to survive.

After solving the MOSFET problem, the pH meter able to log successfully from 5:30 pm to 12 am from 13th March to 14th March. Figure 4.44 below shows the close up graph of Figure B-10-3 in Appendix B.

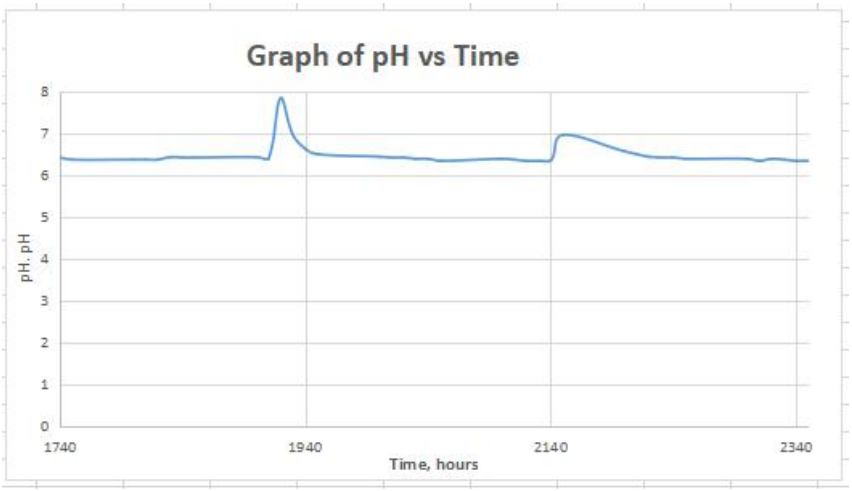


Figure 4.44: Graph of pH versus Time

Figure 4.44 above shows the graph of pH versus Time from 5:40 pm to 11:50 pm on 13th March 2020. The graph value was logged right after correcting the MOSFET problem and the graph shows that the pond pH is about pH 6 and fluctuates about pH 7 around 7:30 pm and 10 pm.

However, after 12 am from 13th March to 14th March, the pH drops back to

pH 0 to pH 2 and stays the same until the 18th of March 2020. These phenomena happen because of several reasons, where the sensor might be clogged, the sensor spoilt as the sensor is not designed to submerge in water for a long period of time or there is a connection problem between the pH sensor module and the PCB.

CHAPTER 5

CONCLUSION

5.1 Conclusion

This project has successfully developed an Internet of Things water quality monitoring system that is capable of monitor 3 types of water parameters which are turbidity, temperature, and pH in real-time. Moreover, this system also powered by solar and able to perform monitoring for 24 hours without any human assistance. Arduino Nano and ESP8266 are the mainboards that use in this project to perform data logging and data transmitting to the cloud.

The cloud function used in this project is Firebase hosted by Google and it is chosen because it is capable of storing data, processing data and send a notification to mobile devices. Next, the Android mobile application developed in this project is able to view all historical data, change logging interval and request logging in real-time from the microcontroller via the cloud server.

Over the course of the field tests, there are several problems encountered ranging from software to hardware. All of the problems had been successfully solved and discussed in Chapter 4 of this report. Moreover, there are also several improvements done to this prototype which also discussed in Chapter 4 in this report.

To sum up, all the objectives of this project have been met successfully which are design and develop an embedded system architecture to perform real-time water quality monitoring, integrating IoT into real-time water quality monitoring systems and finally, develop an Android GUI to display water parameters in graphical format.

5.2 Limitation of The Prototype

In this project, there are several limitations in the IoT Water Quality Monitoring Prototype. The first limitation is the overall power consumption used by Arduino Nano and ESP8266 can only sustain for maximum period of 48 hours without sunlight. Moreover, increasing the battery capacity require a larger junction box because the junction box used in the prototype unable to hold more than four batteries.

Next, ESP8266 used HTTP protocol to communicate with Firebase. This protocol require a stronger internet connection and consume more power during data transmission. The farm require a stronger WiFi router that is able to cover a wider area if this system is placed in a larger farm and the prototype is far away from the router.

Furthermore, this system is only able to perform water quality monitoring in real-time. It is not able to preform any future prediction for the water parameters logged such as temperature, pH and turbidity. Thus, aquaculture farmer is unable predict what will happen to the water quality in the coming days.

The final limitation of this prototype is, the prototype is only able to monitor three types of water parameters which is not sufficient for the aquaculture farmer to get a better understanding of water quality. Other sensors such as ammonia sensor, dissolved oxygen sensor, carbon dioxide sensor and conductivity sensor are equally important to monitor the water quality.

5.3 Recommendation For Future Work

There are several recommendations can be done for future work to improve the overall IoT water quality monitoring of this project. Firstly, the Arduino Nano development board can be replaced by a 16-bit low power microcontroller provided by Microchip in order to reduce the overall power consumption of the system and increase ADC accuracy as ATMega328p is a 10-bit microcontroller.

Secondly, LoRa device can also be used to replace ESP8266 as data transmission as LoRa device consume significantly less power than ESP8266 and it is able to transmit at longer range. Moreover, with bigger aquaculture farm, ESP8266 require stronger WiFi signal in order to transmit the data. Thus, LoRa device is a better choice for a larger aquaculture farm. Moreover, instead of changing to the LoRa device, ESP8266 can also use the MQTT protocol to transmit data as it is able to operate in narrowband and consume less power then HTPP protocol. However, the drawback of both of these methods is, it requires to set up a ground station to receive the data from the logging system and transmit the data to the cloud via WiFi at the ground station.

Thirdly, as the logging system collects more and more data, Artificial Intelligence can also be implemented in the cloud server in order to provide a prediction based on the water quality parameters logged into the cloud. This will

allow the aquaculture farmer to get an earlier alert before the water quality is out of range.

Lastly, more sensors can also be added to the system to monitor more parameters and more decisions can be made based on the monitored result. Sensors such as an ammonia sensor should be added to the aquaculture pond as water ammonia levels are equally important as pH to the fish.

REFERENCES

alimentarium, 2017. *The history of aquaculture.* [online] Available at: https://www.alimentarium.org/en/knowledge/history-aquaculture [Accessed 18 Aug 2019].

Arifin, A., Irwan, I., Abdullah, B. and Tahir, D., 2017. Design of sensor water turbidity based on polymer optical fiber. *Proceedings - 2017 International Seminar on Sensor, Instrumentation, Measurement and Metrology: Innovation for the Advancement and Competitiveness of the Nation, ISSIMM 2017*, 2017-January, pp.146–149.

Beyhan, d.J., 2019. *World Seafood Map 2019: Value Growth in the Global Seafood Trade Continues*. [online] Available at: <https://research.rabobank.com/far/en/sectors/animal-protein/world-seafood-trade-map.html> [Accessed 18 Aug 2019].

Daigavane, V. and Gaikwad, M. (2017). Water Quality Monitoring System Based on IOT. *Advances in Wireless and Mobile Communications.* [online] Available at: <https://www.ripublication.com/awmc17/awmcv10n5_24.pdf> [Accessed 12 Aug 2019].

Deepa, S., 2015. *Measuring Dissolved Oxygen (3 Methods)*. [online] Available at: <http://www.biologydiscussion.com/cell-biology/measuring-dissolved-oxygen-3-methods/7605> [Accessed 18 Aug2019].

Encinas, C., Ruiz, E., Cortez, J. and Espinoza, A., 2017. Design and implementation of a distributed IoT system for the monitoring of water quality in aquaculture. *Wireless Telecommunications Symposium*, pp.1–7.

EngineersGarage, 2012. *ZigBee v/s Wi-Fi*. [online] Available at: <https://www.engine ersgarage.com/contribution/zigbee-vs-wi-fi> [Accessed 18 Aug 2019].

Fondriest Environmental, 2014. *Turbidity, Total Suspended Solids & Water Clarity*. [online] Available at: <https://www.fondriest.com/environmental-measurements/parameters/water-quality/turbidity-total-suspended-solids-water-clarity/#Turbid2> [Accessed 18 Aug 2019].

Food and Agriculture Organization (FAO), 2018. SOFIA 2018 - State of Fisheries and Aquaculture in the world 2018. [online] Available at: <http://www.fao.org/state-of-fisheries-aquaculture> [Accessed 18 Aug 2019].

Fowler, P., Baird, D., Bucklin, R., Yerlan, S., Watson, C. and Chapman, F., 1994. Microcontrollers in Recirculating Aquaculture Systems. *ESS-326 Florida Energy Extension Service*, pp. 1-7.

Gondchawar, N. and Kawitkar, P.R.S., 2016. IoT based Smart Agriculture. *International Journal of Advanced Research in Computer and Communication Engineering*, [online] 5(6), pp.838–842. Available at: <http://www.kresttechnology.com/krest-academic-projects/krest-major-projects/ECE/BTech Major ECE EMBED DED2016-17/Btech ECE Embedded Major BP 2016-17/3. Automated Irrigation

System In Agriculture.pdf%0Ahttp://www.ijtre.com/images/scripts/2016040325.pdf>.

Gopavanitha, K. and Nagaraju, S., 2018. A low cost system for real time water quality monitoring and controlling using IoT. *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing, ICECDS 2017*, pp.3227–3229.

Jiang, P., Xia, H., He, Z. and Wang, Z., 2009. Design of a Water Environment Monitoring System Based on Wireless Sensor Networks. *Sensors*, 9, pp. 6412-6434.

Leaffin, 2017. *What is Role of Dissolved Oxygen in Aquaponics?* [online] Available at: <https://www.leaffin.com/dissolved-oxygen-aquaponics/> [Accessed 18 Aug 2019].

Liu, Y.T., Lin, B.Y., Yue, X.F., Cai, Z.X., Yang, Z.X., Liu, W.H., Huang, S.Y., Lu, J.L., Peng, J.W. and Chen, J.Y., 2018. A solar powered long range real-time water quality monitoring system by LoRaWAN. *2018 27th Wireless and Optical Communication Conference, WOCC 2018*, pp.1–2.

Manyvone, D., Takitoge, R. and Ishibashi, K., 2019. Wireless and low-power water quality monitoring beat sensors for agri and acqua-culture IoT applications. *ECTI-CON 2018 - 15th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, pp.122–125.

Masser, M.P., Rakocy, J. and Losordo, T.M., 1992. Recirculating Aquaculture Tank Production Systems: Management of Recirc ulating Systems. *Southern Regional Aqu aculture Center*, pp. 1-6.

Mendez, G. R., Yunus, M. A. M. and Mukhopadhyay, S. C., 2012. A Wi-Fi based Smart Wireless Sensor Network for Monitoring an Agricultural Environment.

Niswar, M., Wainalang, S., Ilham, A.A., Zainuddin, Z., Fujaya, Y., Muslimin, Z., Paundu, A.W., Kashihara, S. and Fall, D., 2019. IoT-based water quality monitoring system for soft-shell crab farming. *Proceedings - 2018 IEEE International Conference on Internet of Things and Intelligence System, IOTAIS 2018*, pp.6–9.

Osman, S.O., Mohamed, M.Z., Suliman, A.M. and Mohammed, A.A., 2018. Design and Implementation of a Low-Cost Real-Time In-Situ Drinking Water Quality Monitoring System Using Arduino. *2018 International Conference on Computer, Control, Electrical, and Electronics Engineering, ICCCEEE 2018*, pp.1–7.

Queensland Government, 2018. *Recirculating aquaculture systems*. [online] Availabl e at: <https://www.business.qld.gov.au/industries/farms-fishing-forestry/fisheries/aquaculture/site-selection-production/production-systems/recirculating-systems> [Accessed 18 Aug 2019].

Rasin, Z. and Abdullah, M.R., 2009. Water Quality Monitoring System Using Zigbee Based Wireless Sensor Network. *International Journal of Engineering & Technology*, 9(10), pp.14–18.

Raju, K.R.S.R. and Varma, G.H.K., 2017. Knowledge based real time monitoring system for aquaculture Using IoT. *Proceedings - 7th IEEE International Advanced Computing Conference, IACC 2017*, pp.318–321.
Rao, A. S., Marshall, S., Gubbi, J., Palaniswami, M., Sinnott, R., and Pettigrove, V., 2013. Design of Low-cost Autonomous Water Quality Monitoring System. *Internati onal Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 1-6.

Simbeye, D. S., Zhao, J., and Yang, S., 2014. Design and deployment of wireless sensor networks for aquaculture monitoring and control based on virtual instruments. *Computers and Electronics in Agriculture*, 102, pp. 31-42.

Sneha, P.S. and Rakesh, V.S., 2018. Automatic monitoring and control of shrimp aquaculture and paddy field based on embedded system and IoT. *Proceedings of the International Conference on Inventive Computing and Informatics, ICICI 2017*, (Icici), pp.1085–1089.

Sowmya, C.H., Naidu, C.D., Somineni, R.P. and Ramesh Reddy, D., 2017. Implementation of wireless sensor network for real time overhead tank water quality monitoring. *Proceedings - 7th IEEE International Advanced Computing Conference, IACC 2017*, pp.546–551.

Spiten, C., 2015. Study of potential for Micro-ROVs for Underwater Monitoring Applications and Market. *Master Thesis at The Norwegian University of Life Sciences Department of Mathematical Sciences and Technology.*

Stentiford, G., 2017. *Solving the $6 billion per year global aquaculture disease pro blem*. [online] Available at: <https://marinescience.blog.gov.uk/2017/02/02/solving-the-6-billion-per-year-global-aquaculture-disease-problem/> [Accessed 18 Aug 201 9].

Swann, L., 1997. A Fish Farmer's Guide to Understanding Water Quality. *Aquaculture Extension*.

SwitchDoc Labs, 2015. *Raspberry Pi and Arduino Power Consumption - INA3221*. [online] Available at: <https://www.switchdoc.com/2015/03/ina3221-raspberry-pi-and-arduino-power-consumption/> [Accessed 18 Aug 2019].

Tan, X., Kim, D., Usher, N., Laboy, D., Jackson, J., Kapetanovic, A., Rapai, J., Sabadus, B. and Zhou, X., 2006. An Autonomous Robotic Fish for Mobile Sensing. *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*.

UN DESA (United Nations Department of Economic and Social Affairs), 2017. *World population projected to reach 9.8 billion in 2050, and 11.2 billion in 2100.* [online] Available at: <https://www.un.org/development/desa/en/news/population/w orld-population-prospects-2017.html.> [Accessed 18 Aug 2019].

Vanmore, P. S. V., Khot, M. M., Shirasagi, M. L., Salavi, M. R., 2017. Microcontroller Based automatic Aquaponics System. *International Research Journal of Engineering and Technology (IRJET)*, 4(2), pp. 1495-1499. Wonderopolis, 2019. *How Do Fish Breathe Underwater?* [online] Available at: <https://www.wonderopolis.org/wonder/how-do-fish-breathe-underwater/> [Accessed 18 Aug. 2019].

Zaev, E., Babunski, D. and Tuneski, A., 2016. SCADA system for real-time measuring and evaluation of river water quality. *2016 5th Mediterranean Conference on Embedded Computing, MECO 2016 - Including ECyPS 2016, BIOENG.MED 2016, MECO: Student Challenge 2016*, pp.83–86.

Zhu, X., Li, D., He, D., Wang, J. M, D. and Li, F., 2009. A remote wireless system f or water quality online monitoring in intensive fish culture. *Computers and Electronics in Agriculture*, 71, pp. 3-9.

APPENDICIES

APPENDIX A: Datasheets

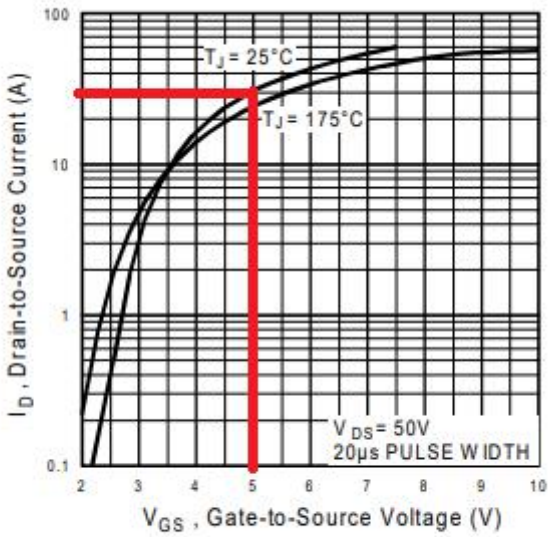


Fig 3. Typical Transfer Characteristics

Figure A-1: IRL530N V_{GS} Versus I_{DS} Characteristic

HOLTEK

HT78XX Series
500mA TinyPower™ LDO

Features

- Output voltage ranges: Fixed range of 1.8V, 2.5V, 2.7V, 3.0V, 3.3V, 5.0V type.
- Highly accuracy: ±2%
- Low voltage drop: 360mV (typ.), V_{OUT}=5.0V at 500mA
- Guaranteed output current: 500mA
- Low quiescent current: 5µA (typ.)
- Current limiting
- Over-temperature shutdown
- SOT-89, TO-92 Packages

Applications

- Battery powered systems
- Personal Digital Assistants
- Peripheral cards
- PCMCIA cards
- Personal Communication Equipment

Figure A-2: HT7833 3.3V Voltage Regulator Features

APPENDIX B: Graphs

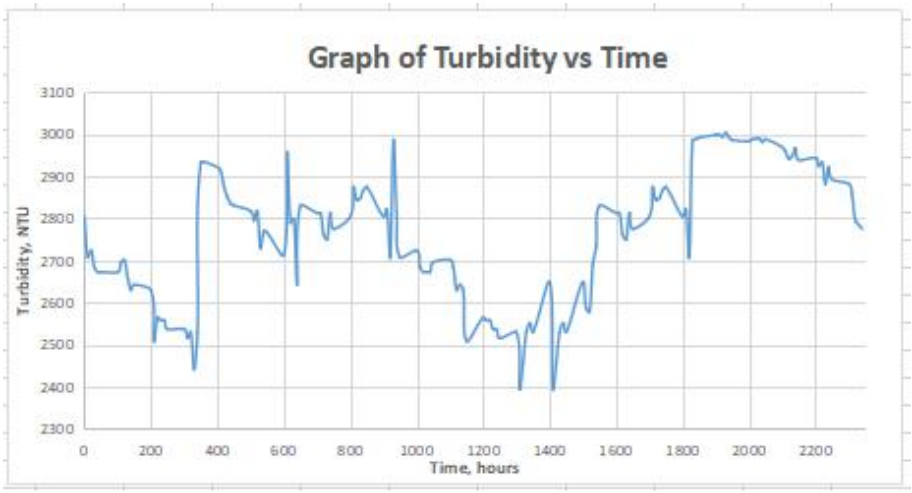


Figure B-1-1: Turbidity Graph at 04-03-2020

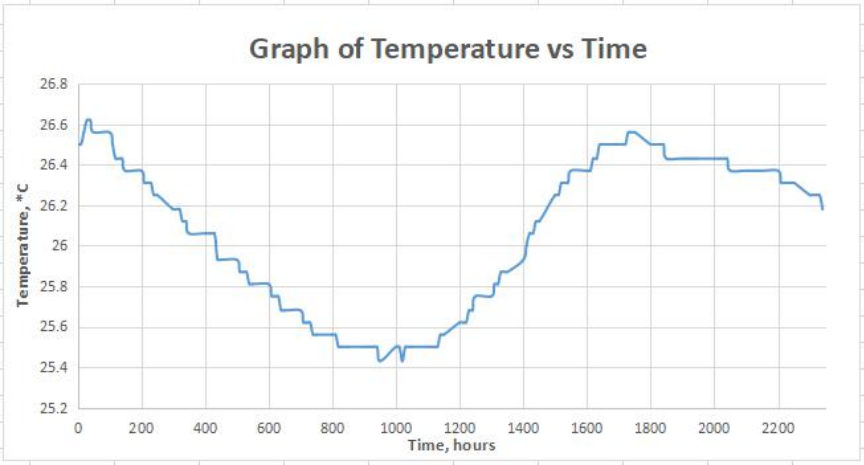


Figure B-1-2: Temperature Graph at 04-03-2020

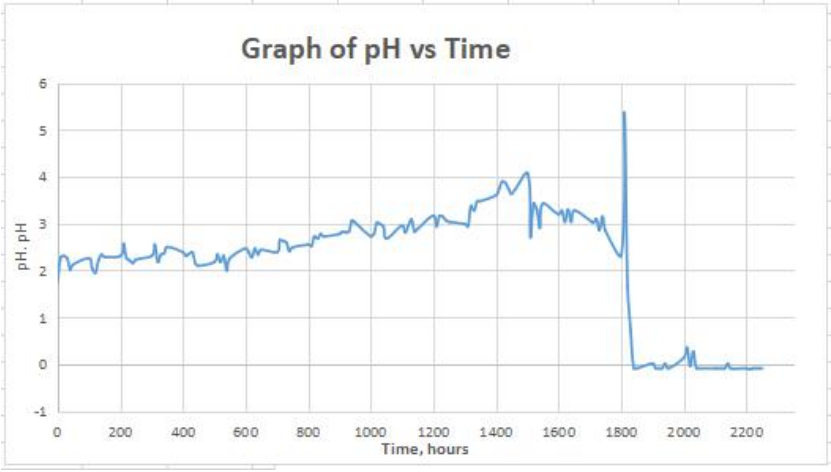


Figure B-1-3: pH Graph at 04-03-2020

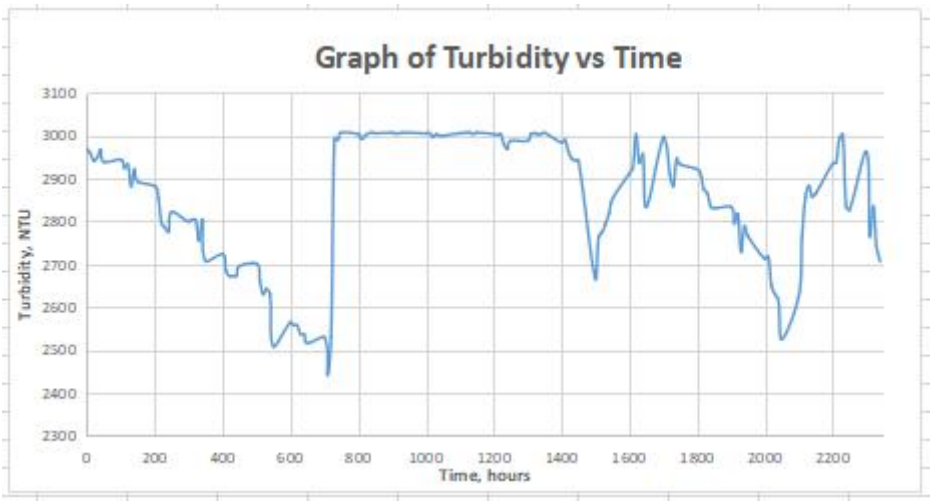


Figure B-2-1: Turbidity Graph at 05-03-2020

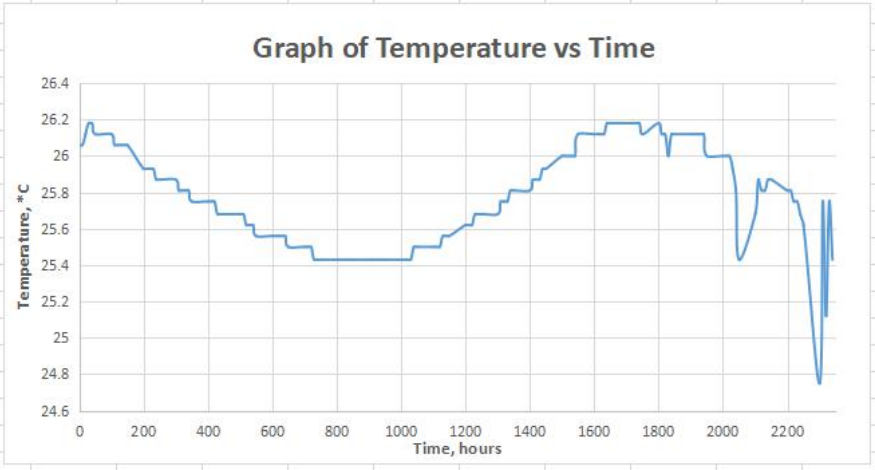


Figure B-2-2: Temperature Graph at 05-03-2020

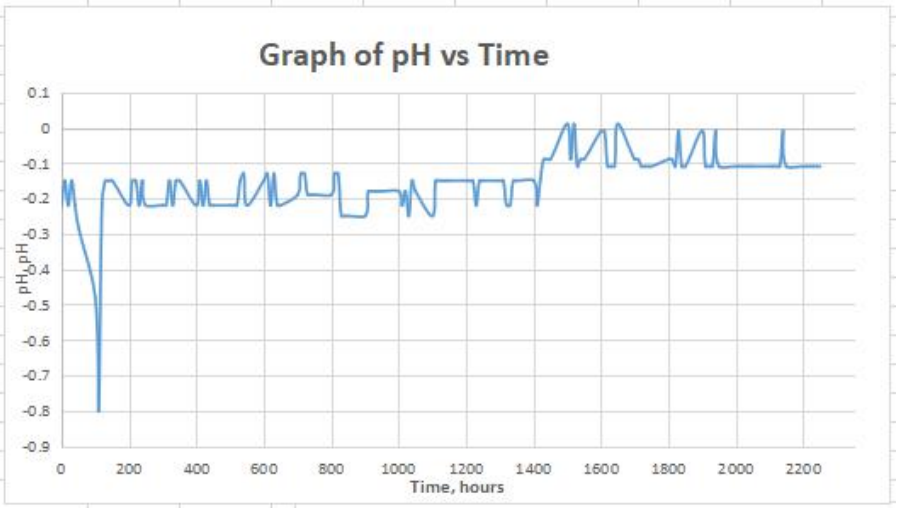


Figure B-2-3: pH Graph at 05-03-2020

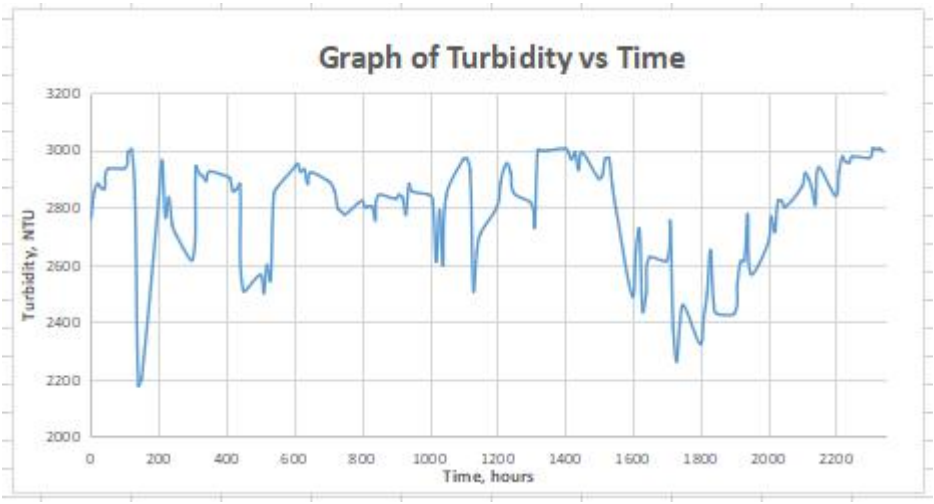


Figure B-3-1: Turbidity Graph at 06-03-2020

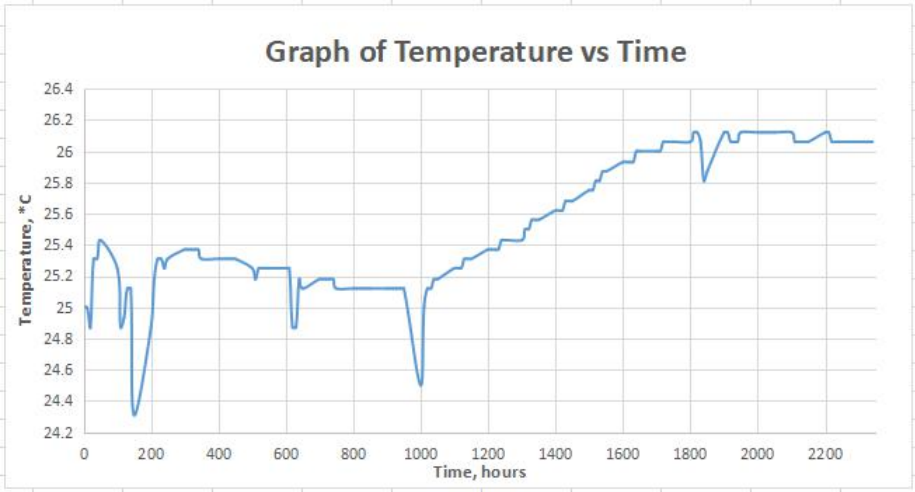


Figure B-3-2: Temperature Graph at 06-03-2020

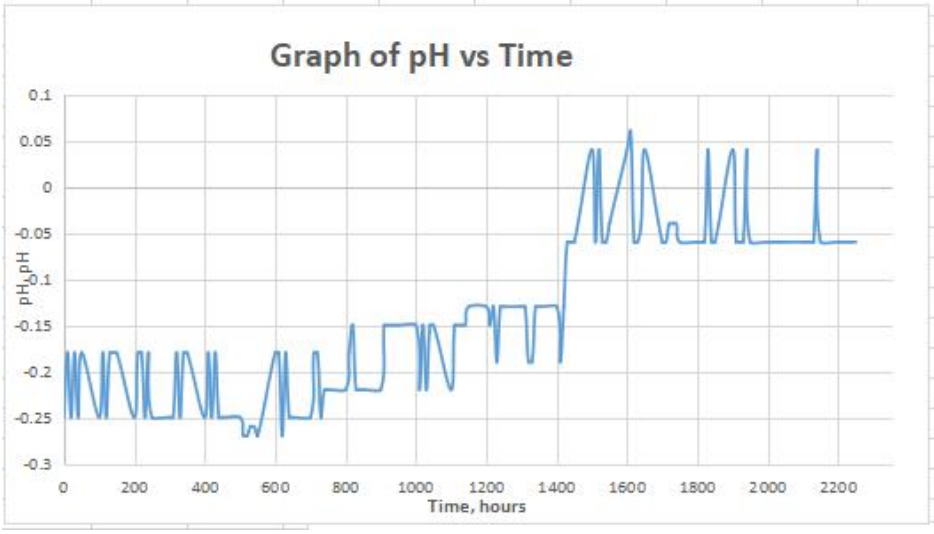


Figure B-3-3: pH Graph at 06-03-2020

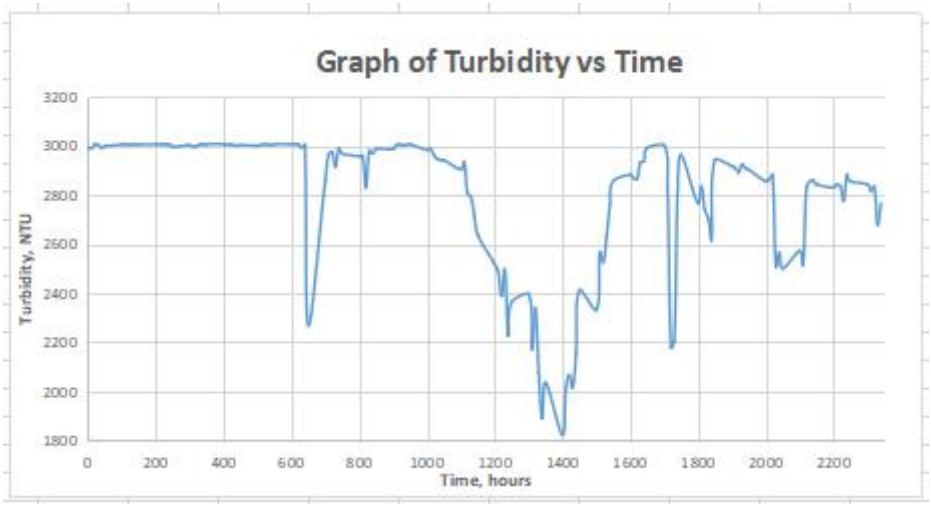


Figure B-4-1: Turbidity Graph at 07-03-2020

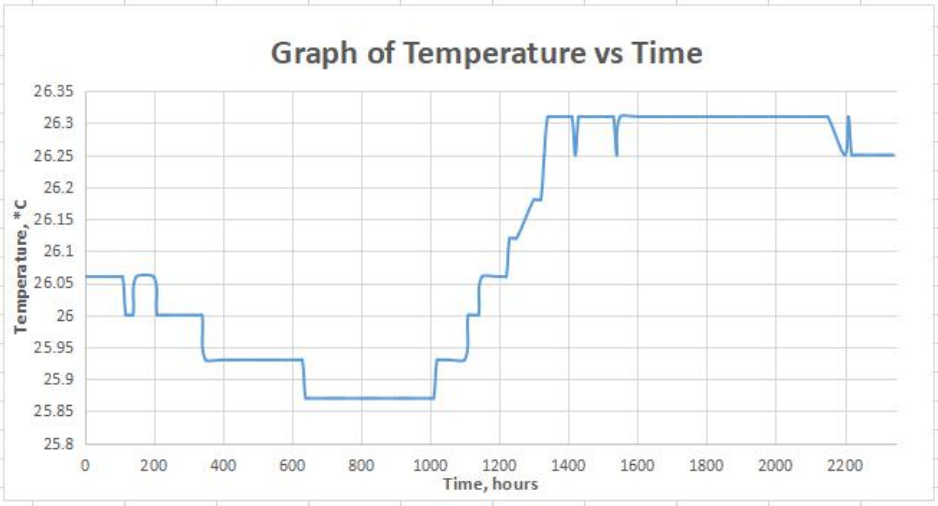


Figure B-4-2: Temperature Graph at 07-03-2020

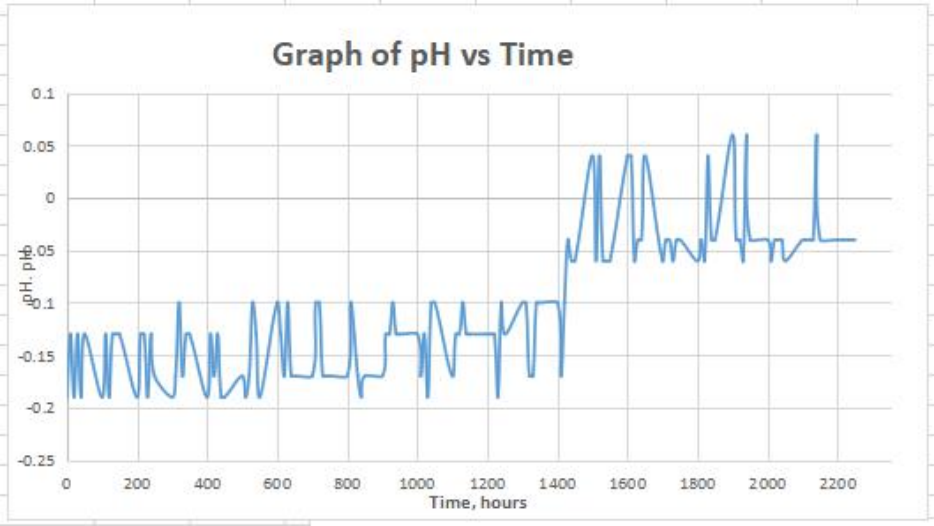


Figure B-4-3: pH Graph at 07-03-2020

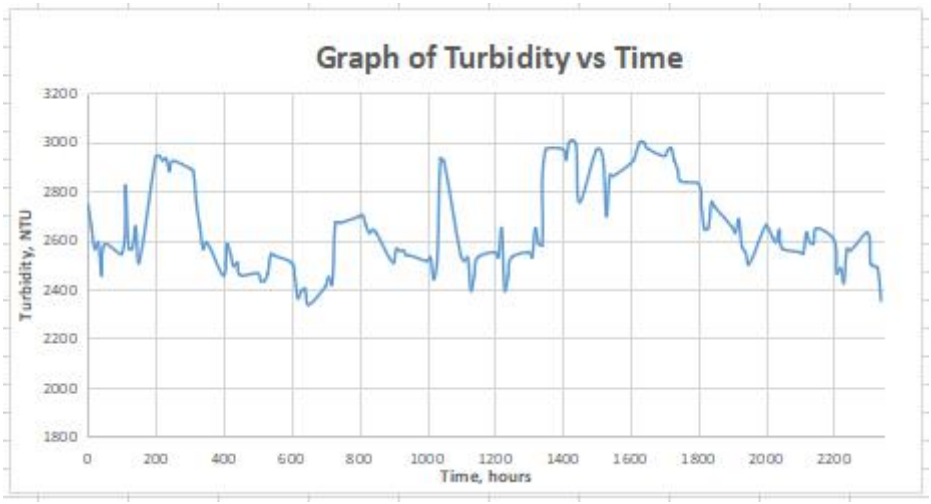


Figure B-5-1: Turbidity Graph at 08-03-2020

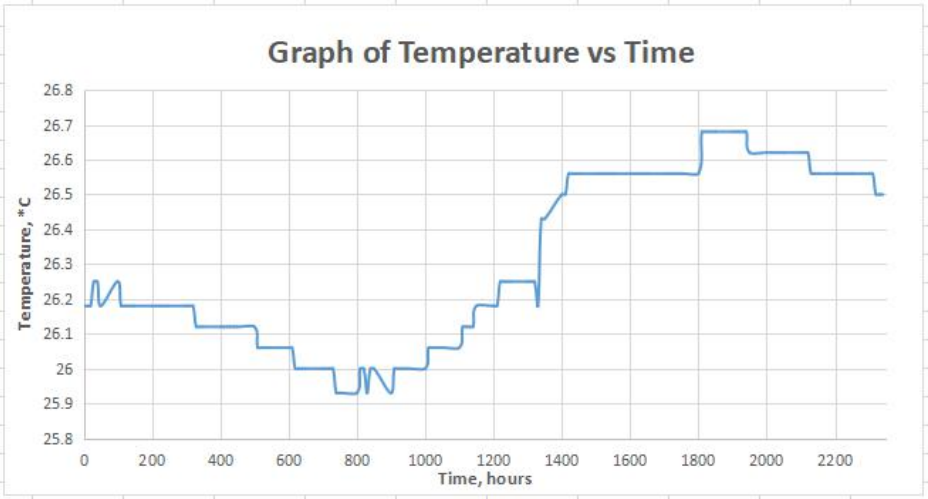


Figure B-5-2: Temperature Graph at 08-03-2020

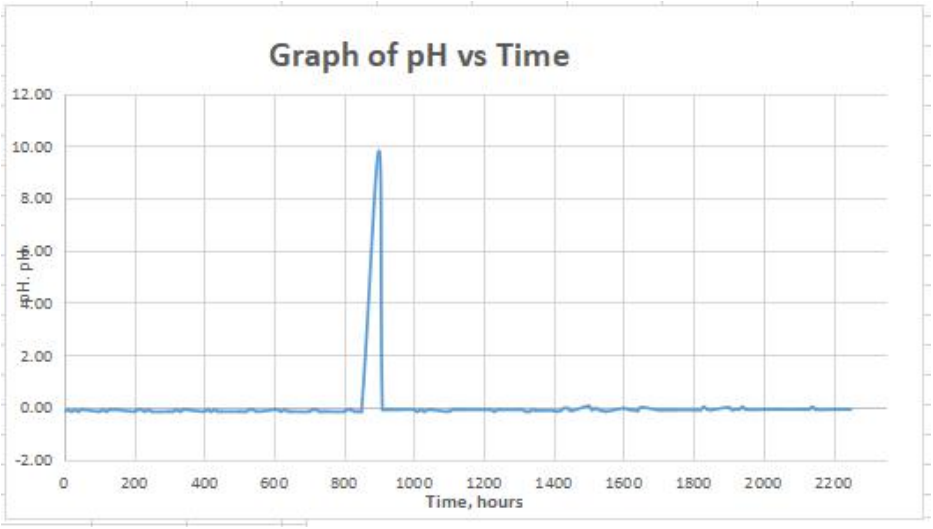


Figure B-5-3: pH Graph at 08-03-2020

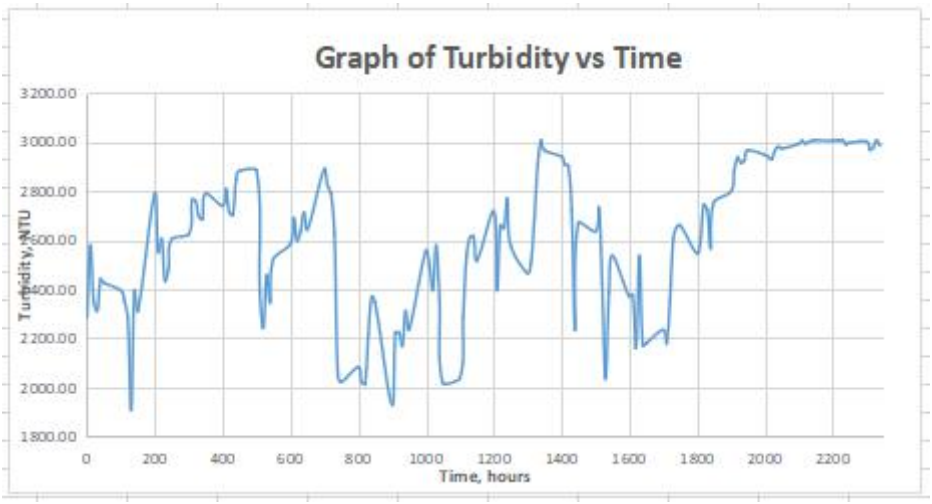


Figure B-6-1: Turbidity Graph at 09-03-2020

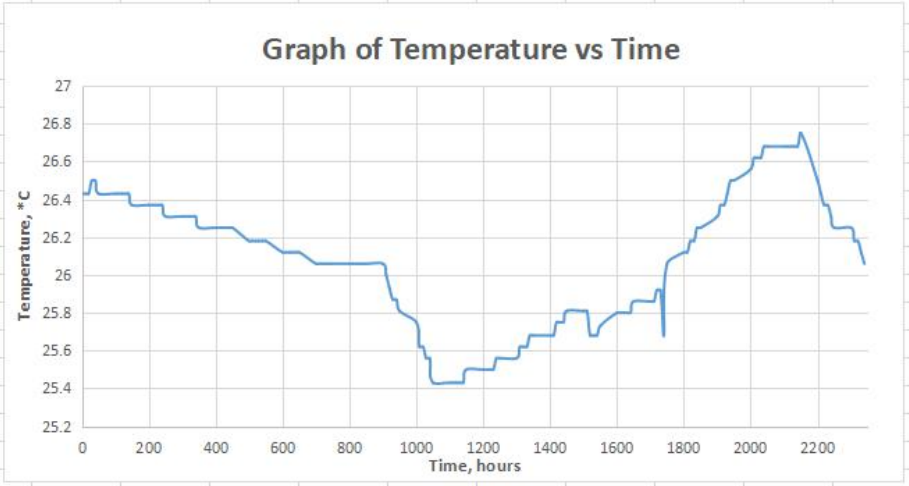


Figure B-6-2: Temperature Graph at 09-03-2020

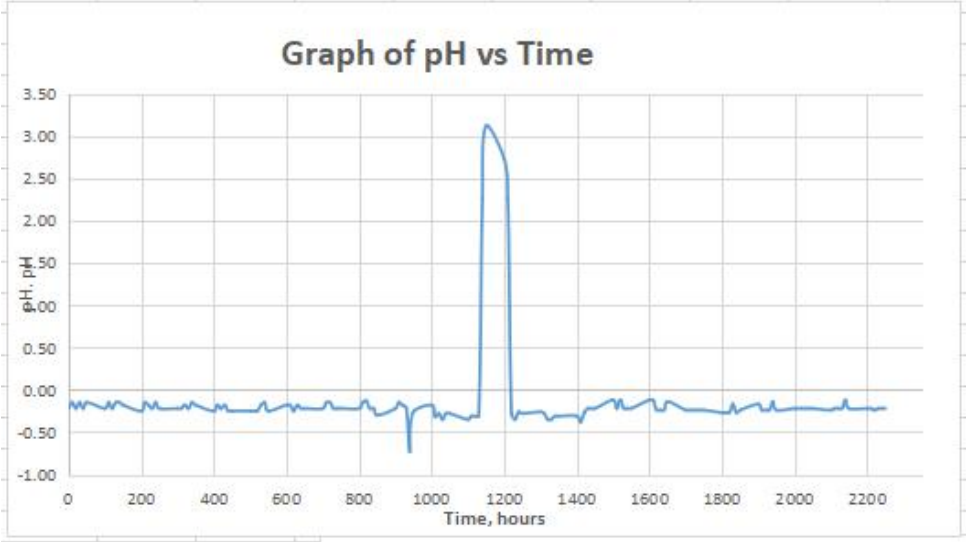


Figure B-6-3: pH Graph at 09-03-2020

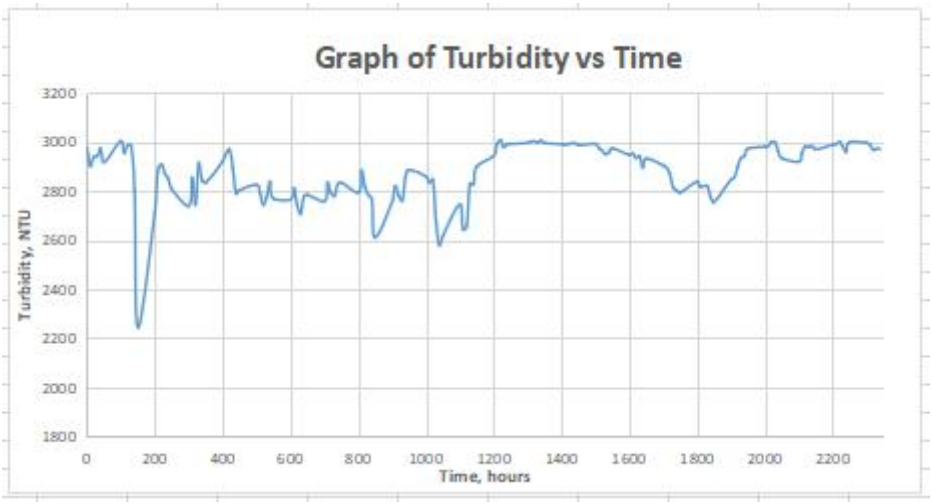


Figure B-7-1: Turbidity Graph at 10-03-2020

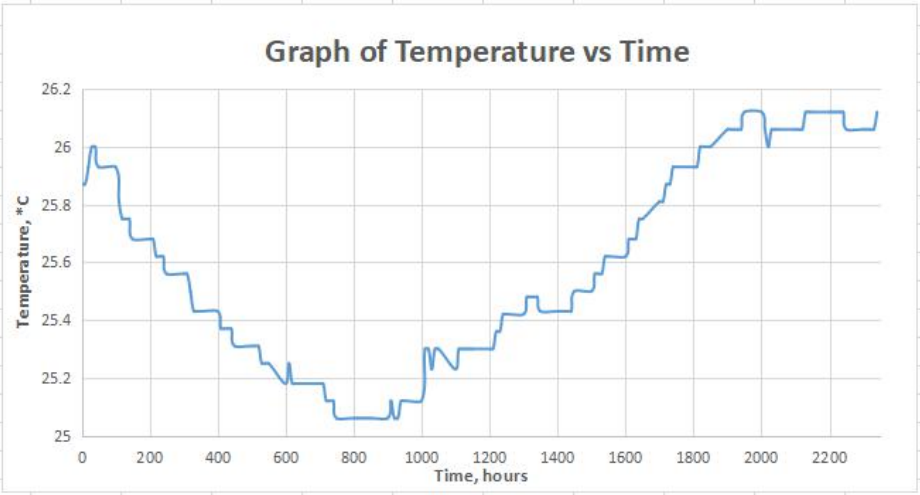


Figure B-7-2: Temperature Graph at 10-03-2020

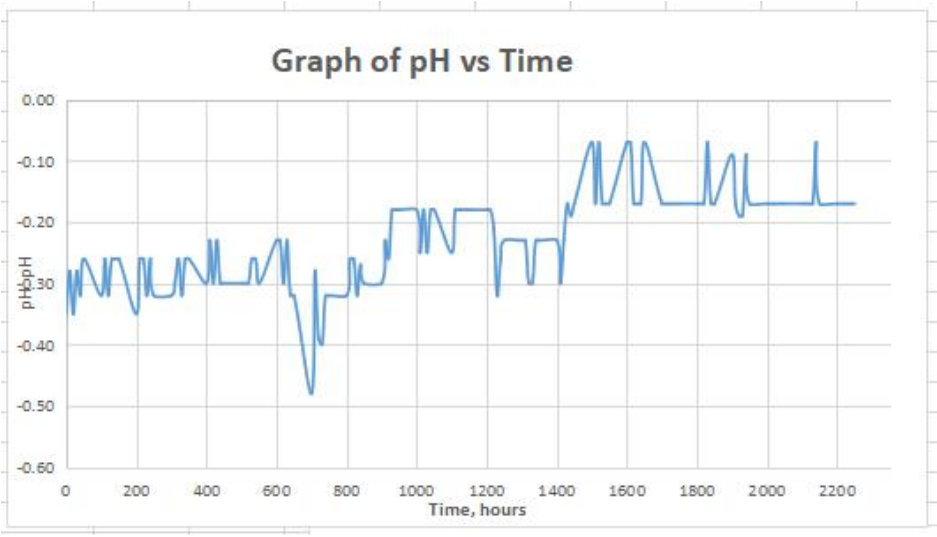


Figure B-7-3: pH Graph at 10-03-2020

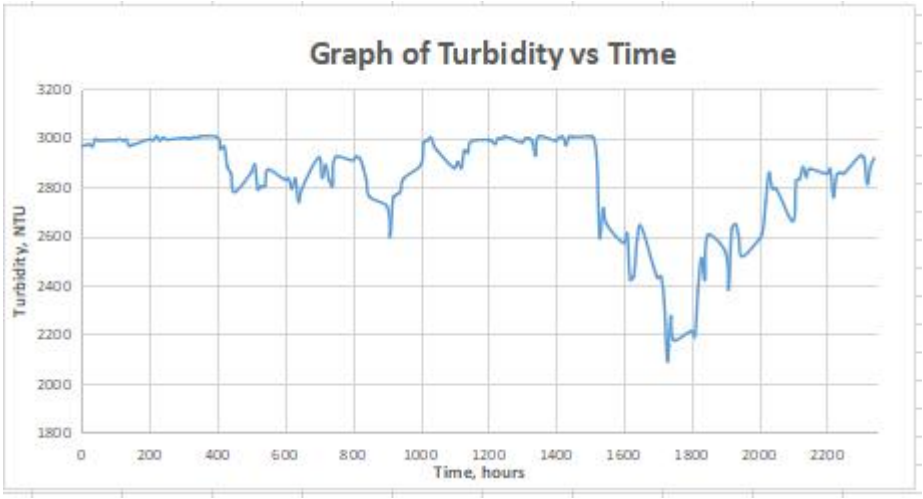


Figure B-8-1: Turbidity Graph at 11-03-2020

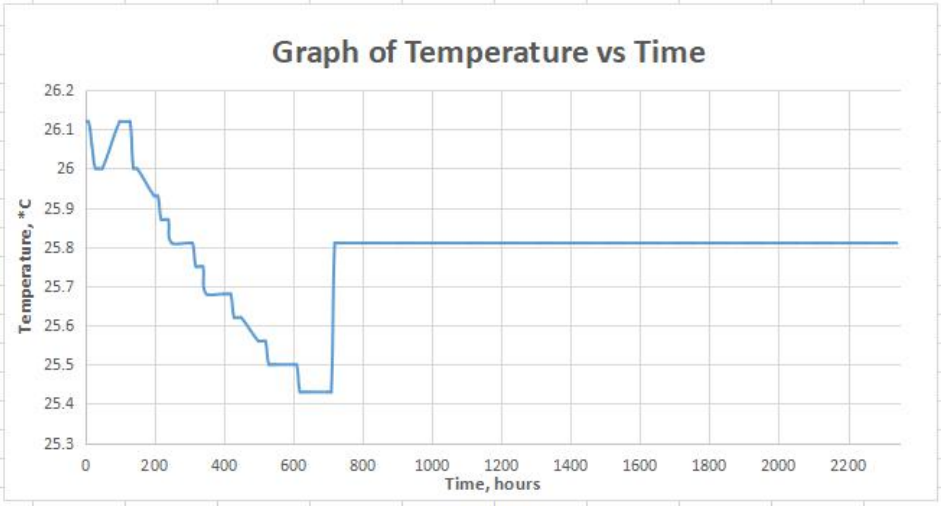


Figure B-8-2: Temperature Graph at 11-03-2020

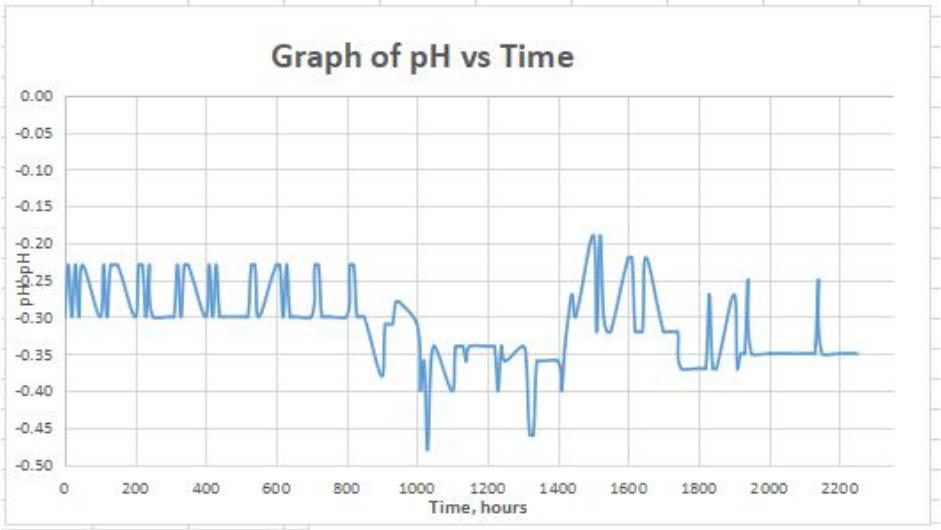


Figure B-8-3: pH Graph at 11-03-2020

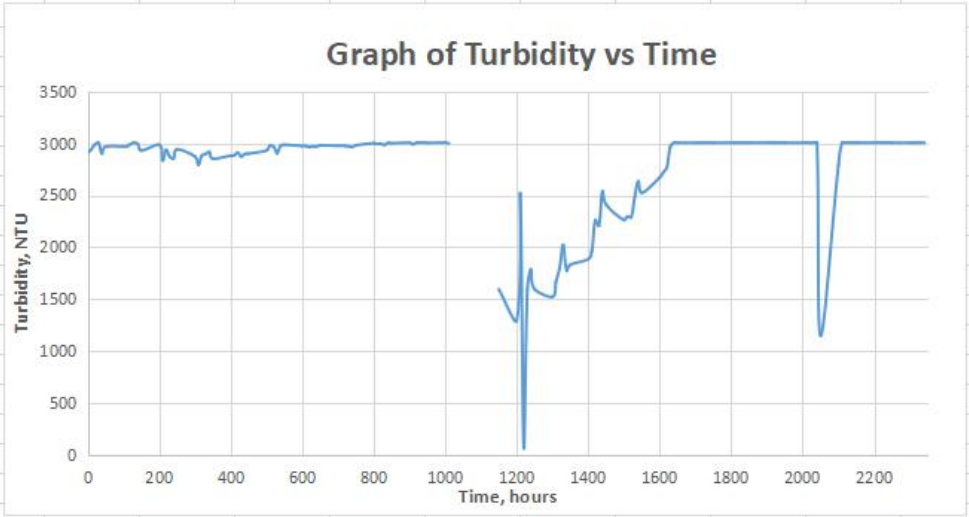


Figure B-9-1: Turbidity Graph at 12-03-2020

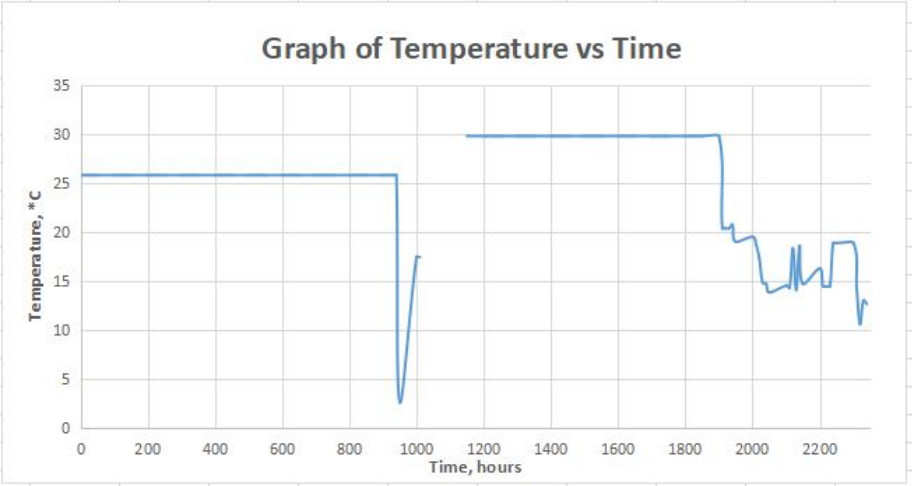


Figure B-9-2: Temperature Graph at 12-03-2020

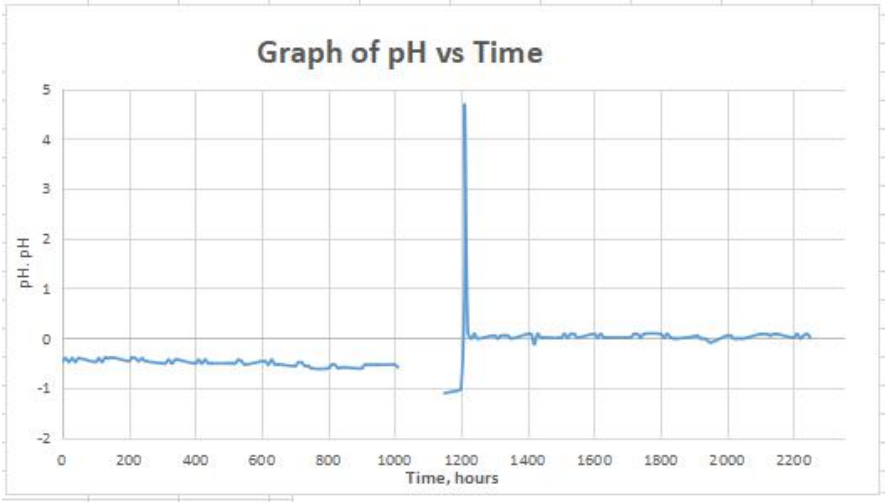


Figure B-9-3: pH Graph at 12-03-2020

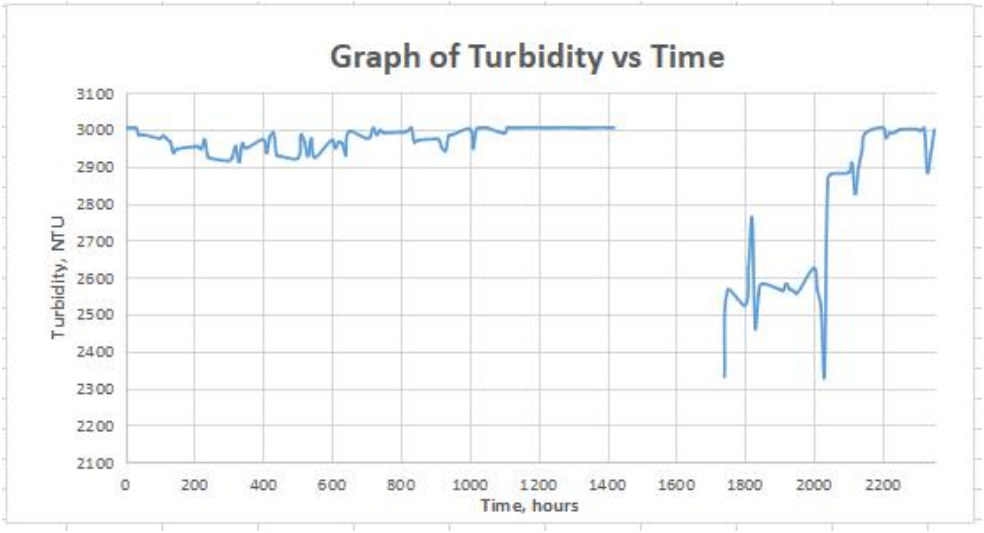


Figure B-10-1: Turbidity Graph at 13-03-2020

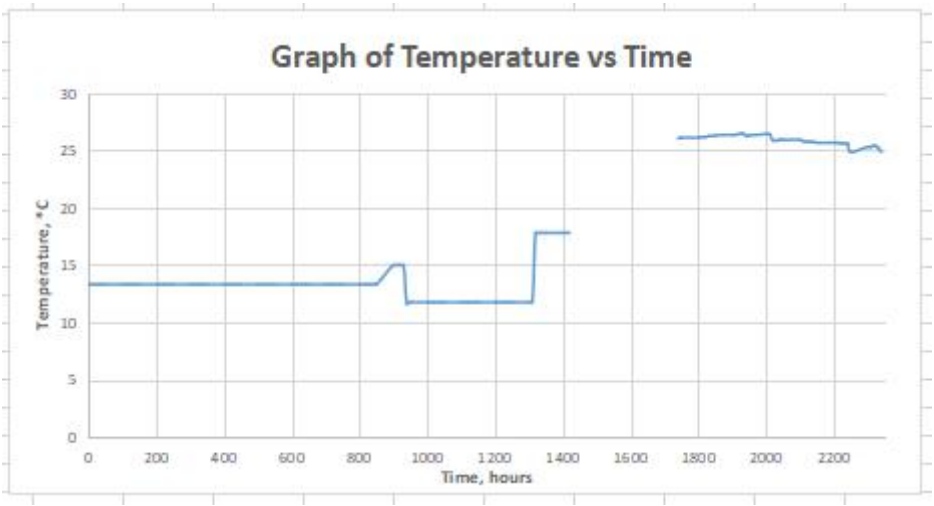


Figure B-10-2: Temperature Graph at 13-03-2020

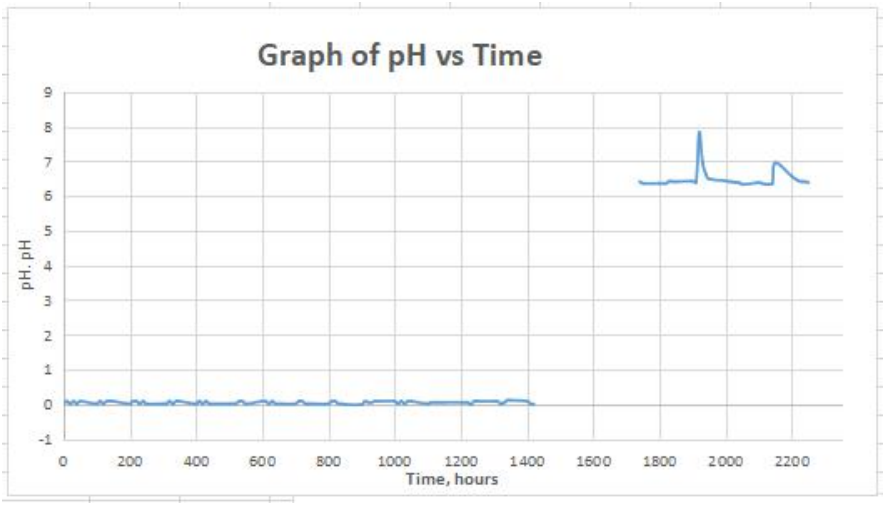


Figure B-10-3: pH Graph at 13-03-2020

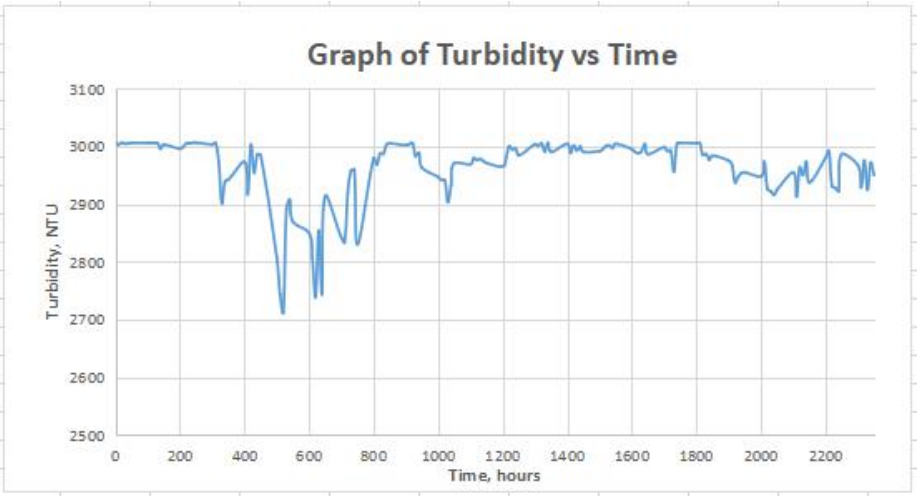


Figure B-11-1: Turbidity Graph at 14-03-2020

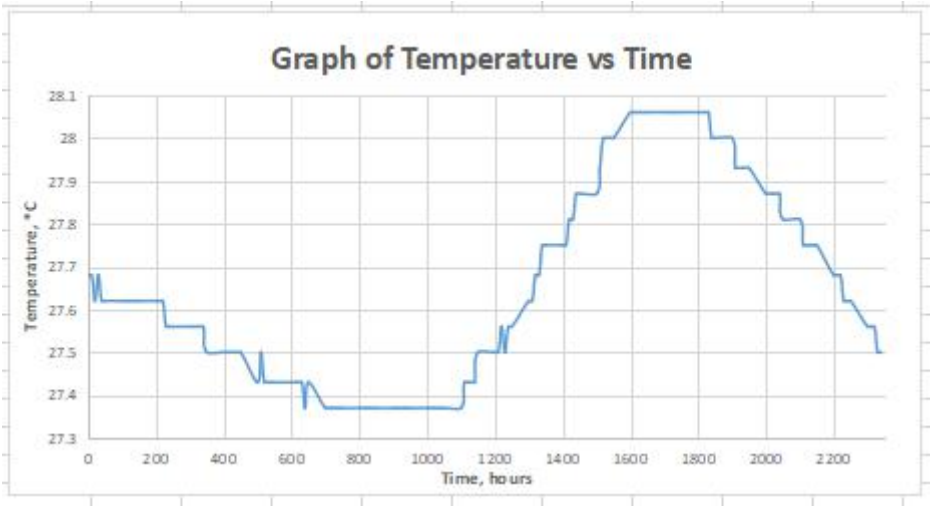


Figure B-11-2: Temperature Graph at 14-03-2020

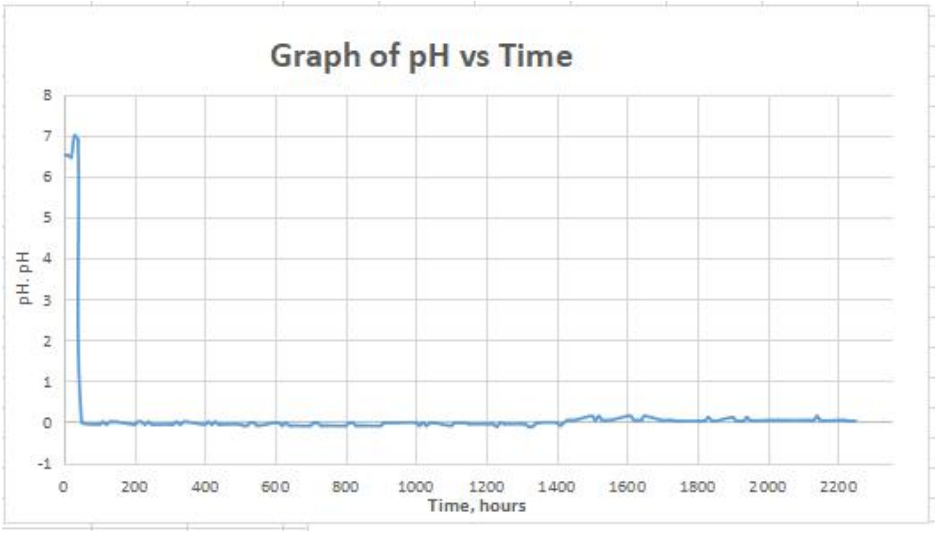


Figure B-11-3: pH Graph at 14-03-2020

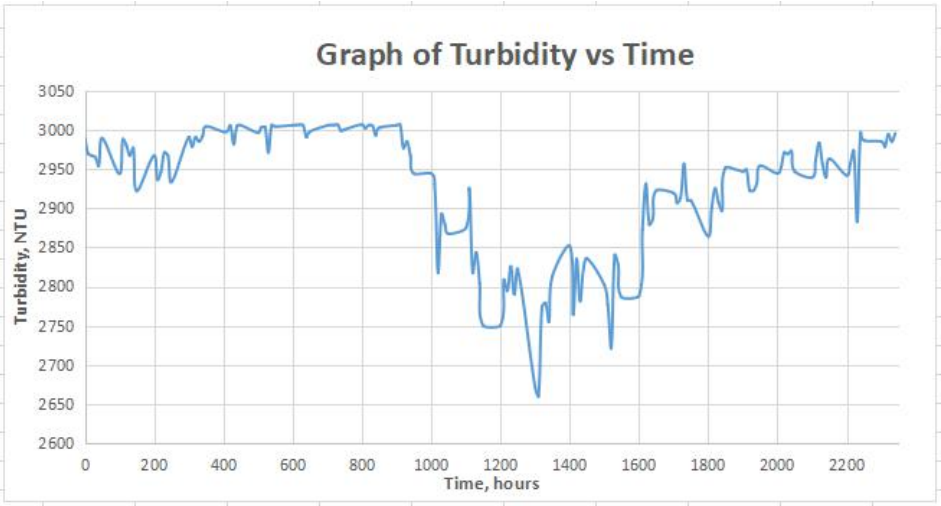


Figure B-12-1: Turbidity Graph at 15-03-2020

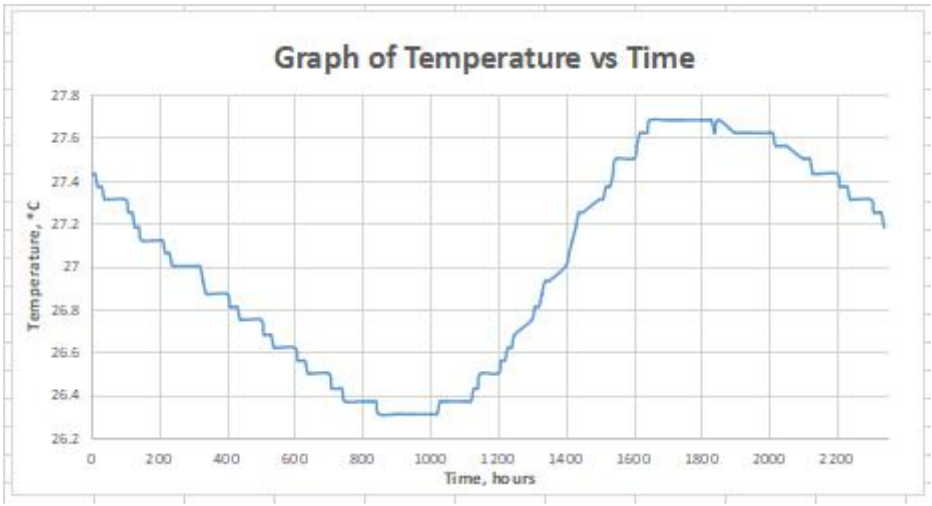


Figure B-12-2: Temperature Graph at 15-03-2020

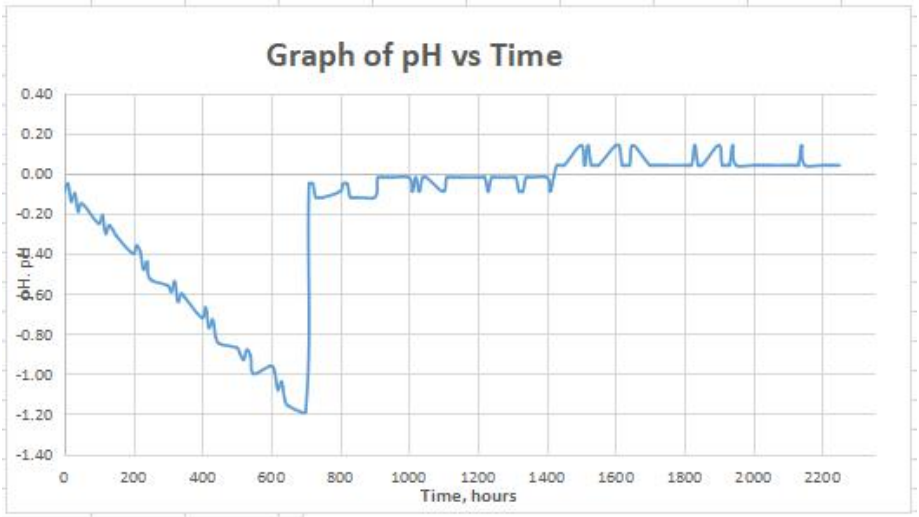


Figure B-12-3: pH Graph at 15-03-2020

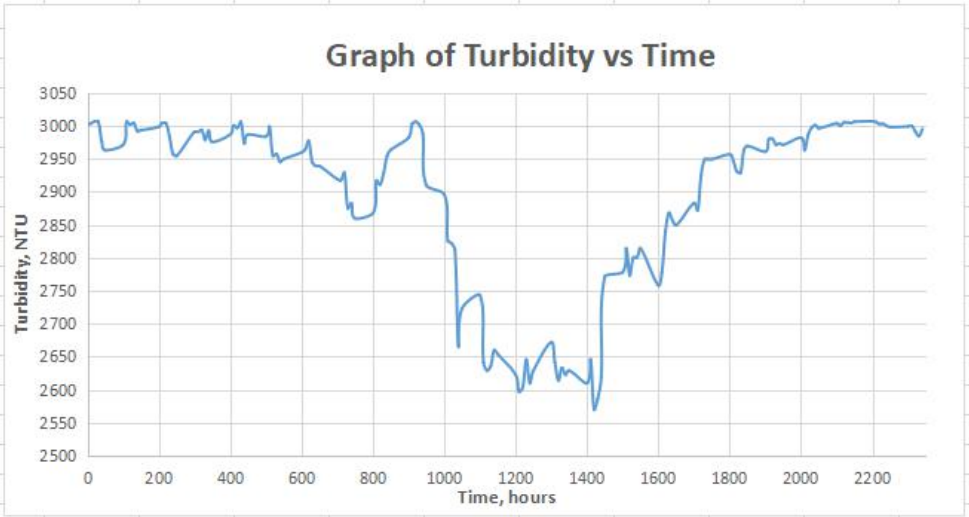


Figure B-13-1: Turbidity Graph at 16-03-2020

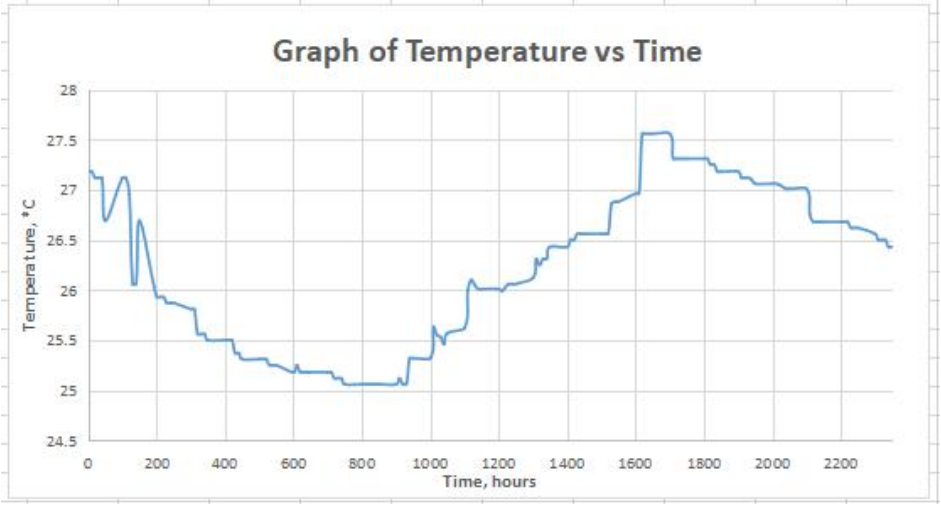


Figure B-13-2: Temperature Graph at 16-03-2020

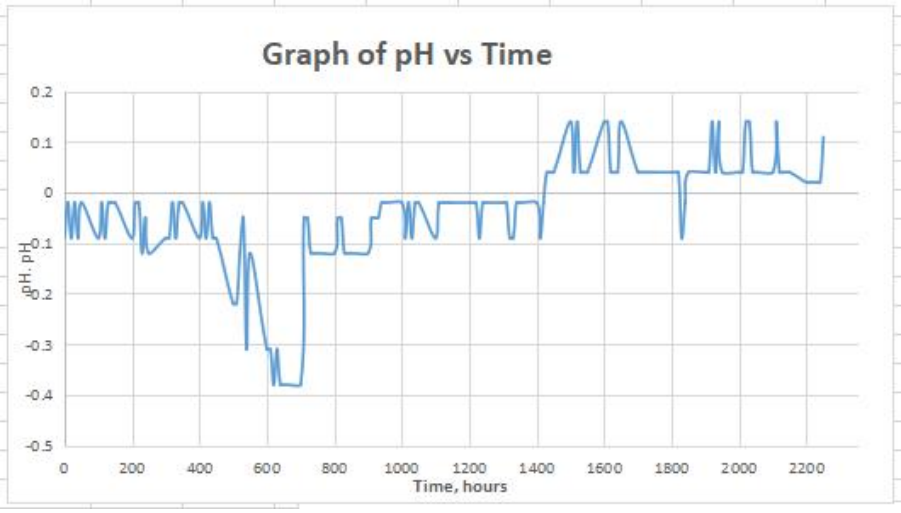


Figure B-13-3: pH Graph at 16-03-2020

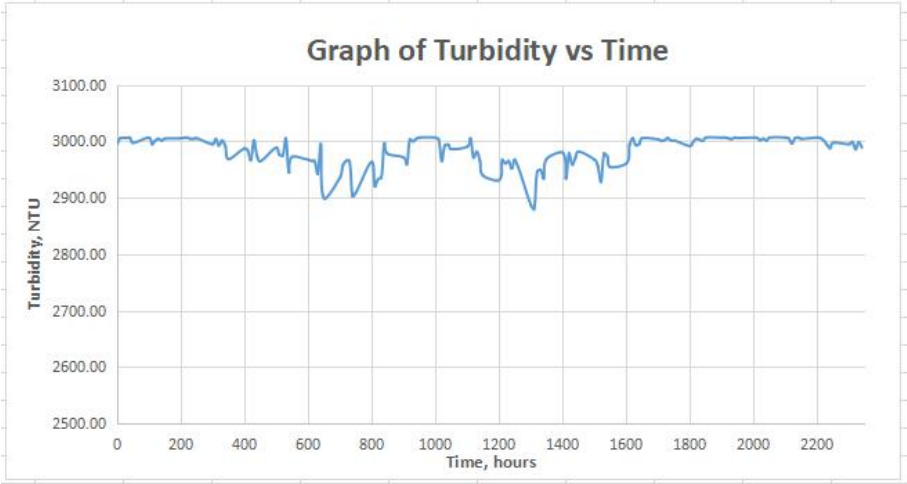


Figure B-14-1: Turbidity Graph at 17-03-2020

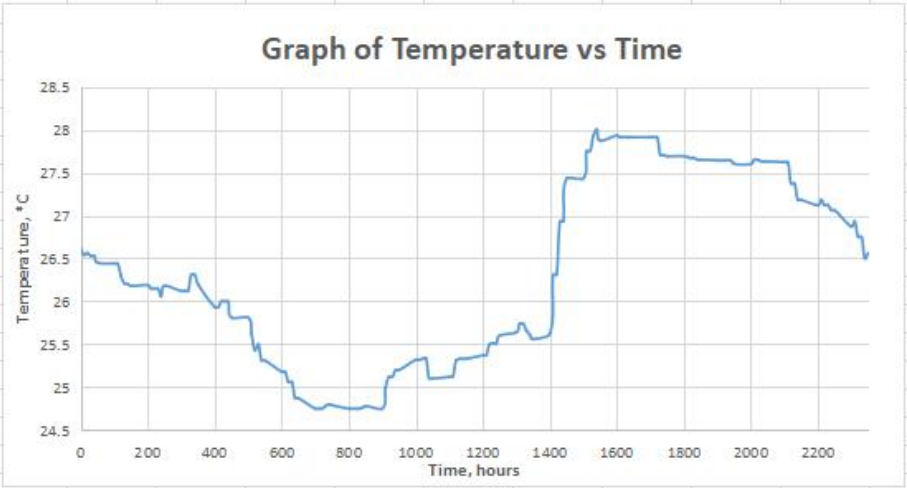


Figure B-14-2: Temperature Graph at 17-03-2020

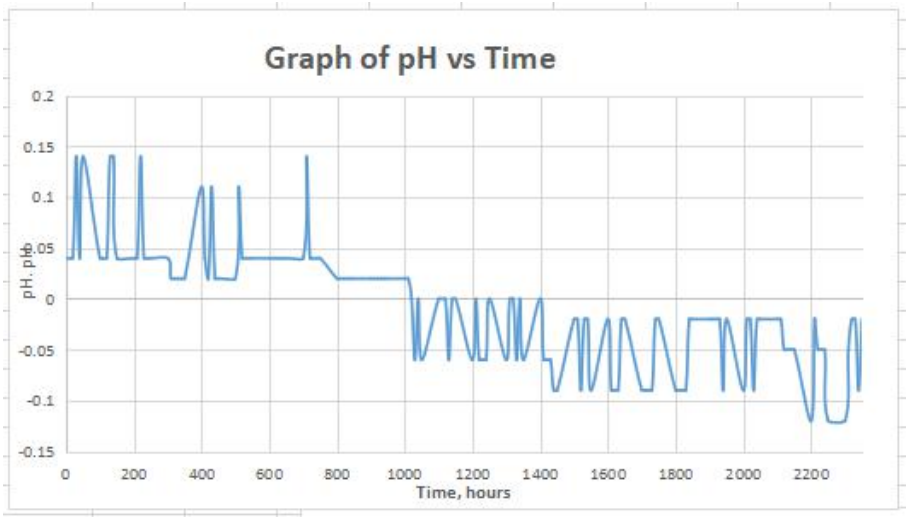


Figure B-14-3: pH Graph at 17-03-2020

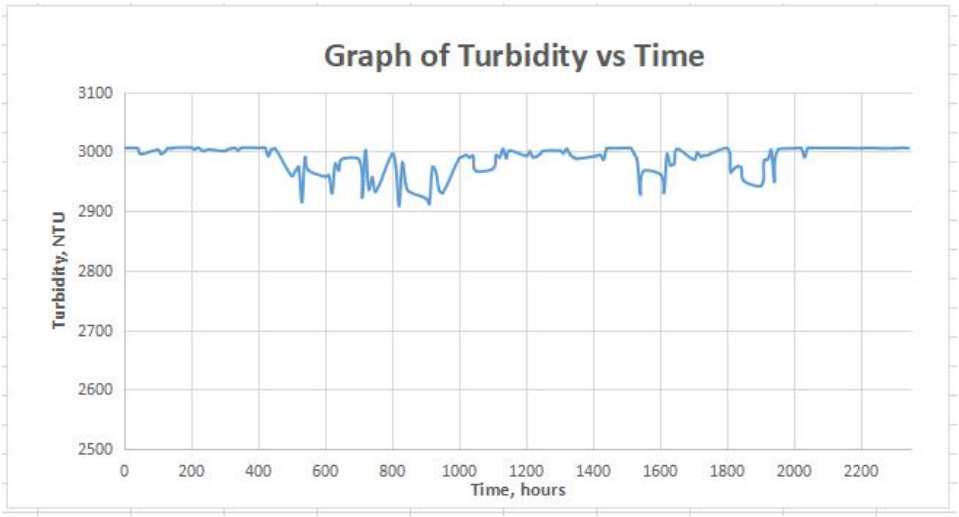


Figure B-15-1: Turbidity Graph at 18-03-2020

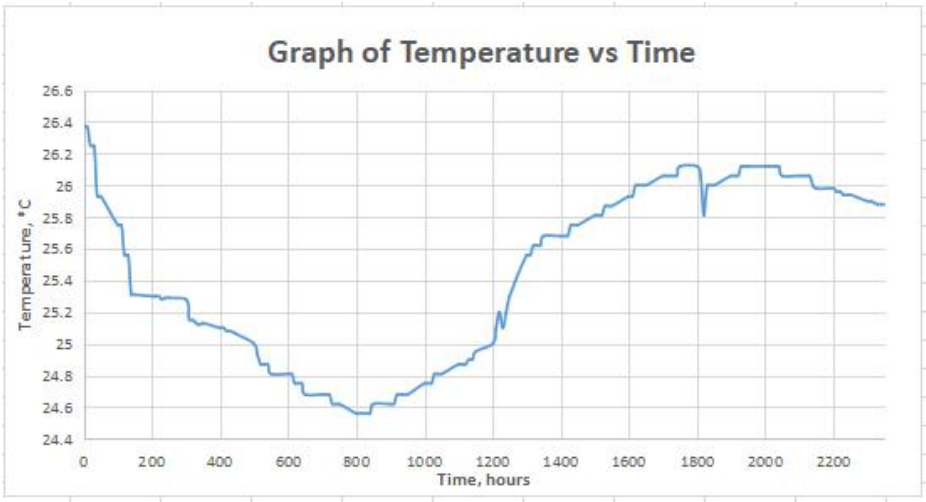


Figure B-15-2: Temperature Graph at 18-03-2020

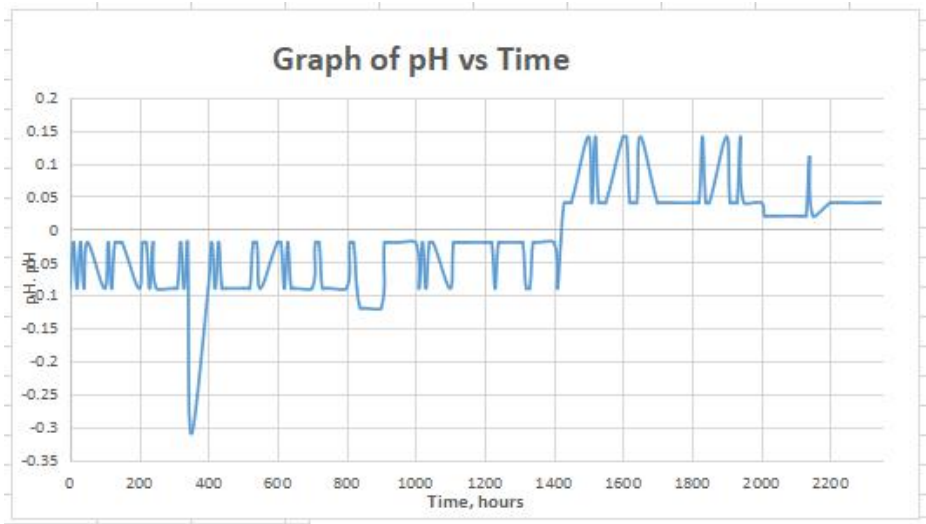


Figure B-15-3: pH Graph at 18-03-2020

APPENDIX C: Codings

```
1
2  //Include Libraries
3  #include <Wire.h>
4  #include "RTClib.h"
5  #include <OneWire.h>
6  #include <DallasTemperature.h>
7  #include <SD.h>
8  #include <SoftwareSerial.h>
9  #include <EEPROM.h>
10
11 //Define Pins Name
12 #define tub_sen_pin A7 //turbidity sensor
13 #define tem_sen_pin 7 //temperature sensor
14 #define ph_sen_pin A0 //ph sensor
15 #define do_sen_pin A4 //dissolved oxygen sensor
16 #define sdcard_pin 10 //chip select pin for the MicroSD Card Adapter (SPI)
17 #define MSFT_tub 2
18 #define MSFT_tem 3
19 #define MSFT_ph 4
20 //#define MSFT_DO 5
21 #define MSFT_esp 6
22 #define test_button 13
23 #define RX 9
24 #define TX 8
25 #define ESP_baudrate 9600
26 //#define WDT_scale 75 //base is 8sec = 375
27 //#define WDT_scale 350
28
29
30 //Declare Global variable
31 uint8_t count_i;
32 int i_temp;
33 int i_sensorValue;
34 float f_sensorValue;
35 int sensorValue_integerPart;
36 int sensorValue_decimalPart;
37 unsigned char rtcc_interval = 48;
38 //48 = 10minutes, 49 = 30minutes, 50 = 60minutes
39 //['1' = 48, '2' = 49, '3' = 50 in unsigned char]
40 bool CheckInterval = false;
41 bool logData = false;
42 unsigned char log_now = 49;
43
44 //Include Header files
45 #include "rtcc.h"
46 #include "turbidity.h"
47 #include "temperature.h"
48 #include "ph.h"
49 //#include "do.h"
50 #include "ESP8266.h"
51 #include "sdcard.h"
52 #include"battery.h"
53
54 /*********************/
55 /* setup() function */
56 /*********************/
57 void setup() {
58   Serial.begin(115200);
59
60   rtcc_setup();
61   tem_setup();
62   esp_setup();
63
64   //Set Input Output Pins
65   pinMode(tub_sen_pin, INPUT);
66   pinMode(tem_sen_pin, INPUT);
67   pinMode(ph_sen_pin, INPUT);
68   pinMode(do_sen_pin, INPUT);
69   pinMode(sdcard_pin, OUTPUT); //SS pin
70   pinMode(RX, INPUT); //RX
```

Figure C-1-1: Arduino Nano Main Program (1)

```
  pinMode(TX, OUTPUT); //TX
  pinMode(MSFT_tub, OUTPUT);
  pinMode(MSFT_tem, OUTPUT);
  pinMode(MSFT_ph, OUTPUT);
//  pinMode(MSFT_DO, OUTPUT);
  pinMode(MSFT_esp, OUTPUT);
  pinMode(test_button, INPUT);

//  //Setup sleep mode
//  SMCR |= (1<<2); //0bxxxx 010 0 <- for setting sleep mode type
//  SMCR |= 1; //Enable sleep mode

//  //Watch Dog Timer (WDT)
//  //Interrupt mode: WDTON=1 WDE=0 WDIE=1
//  //WDPx is determine the prescale of the timer interupt
//  WDTCSR = 0x18; //0b0001 1000 Enable WDCE WDE
//  WDTCSR = 0x21; //0b0010 00001 Enable WDP3 WDP0
//  WDTCSR |= (1<<6); //0bx1xx xxxx Enable WDIE

  Serial.println("Logging System Start");

}

/**************/
/* Main Loop */
/**************/
void loop() {

  char sleep = '0';
  int data_now[25];

  while (1) {
  data_now[25]={'\0'};
  int i=0;

//    if (sleep == '1') {
//      for (int h=0; h<WDT_scale; h++) {
//        Disable BOD for lower power
//        MCUCR |= (3<<5);
//        MCUCR = (MCUCR & ~(1<<5)) | (1<<6);
//        __asm __volatile__("sleep");
//      }
//      sleep = '0';
//    }

    rtcc_check();

    if(ESP8266.available() > 0) {
      log_now = ESP8266.read();
      Serial.print(log_now);
      //Clear receive before begin buffer
      while(ESP8266.available()) {ESP8266.read();}
    }

    if ((logData == true) or (log_now == 56) ) {

     get_rtcc(&data_now[0]);

     get_rtcc(&data_now[0]);
//     Serial.println(F("RTCC Done"));

     get_turbidity(&data_now[dataLength(&data_now[0])]);
//     Serial.println(F("Turbidity Done"));

     get_tem(&data_now[dataLength(&data_now[0])]);
//     Serial.println(F("Temperature Done"));

     get_ph(&data_now[dataLength(&data_now[0])]);
//     Serial.println(F("pH Done"));

//     get_do(&data_now[dataLength(&data_now[0])]);
```

Figure C-1-2: Arduino Nano Main Program (2)

```
//     Serial.println(F("DO Done"));

     get_batt(&data_now[dataLength(&data_now[0])]);
//     Serial.println(F("Battery Done"));

//     count_i=0;
//     while (data_now[count_i] != 1027) {
//        Serial.print(data_now[count_i++]);
//     }
//     Serial.println(F(" "));

     sd_exist();

//     MOSFET(MSFT_esp,'1');

     if (esp_connection('c')) {     //If connection establish
       if (esp_connection('w')) {   //If WiFi Available
         if (log_now != 56) {
          sd_write(&data_now[0]);  //write data for graph
          sd_read();               //upload data for graph
         } else {
          Serial.println("Log once");
          esp_transmit_once(&data_now[0]); //Upload data once as user request
         }
       } else {
         sd_write(&data_now[0]);
       }
     } else {
       sd_write(&data_now[0]);
     }

    //Check logging interval at 12am everyday
    if (CheckInterval == true) {
      if (esp_connection('w')) {   //If WiFi Available
        check_interval();
        CheckInterval = false;
      }
    }

//    MOSFET(MSFT_esp,'0');

    logData = false;
    log_now = 48;

//    Serial.println(F("Done"));
//      sleep = '1';
//      delay (5000);
    }  //if (rtcc_check)

  } //while(1)

}

byte dataLength(int *data) {
  byte count=0;

  while (data[count] != 1027) {
    count++;
  }

  return count;
}

void MOSFET(int mos_select, char on_off) {
  if (on_off == '1') {
    digitalWrite(mos_select,HIGH);
  } else {
    digitalWrite(mos_select,LOW);
  }
}

//ISR for WDT
ISR(WDT_vect) {}
```

Figure C-1-3: Arduino Nano Main Program (3)

```
1
2  SoftwareSerial ESP8266(RX,TX);
3
4  void esp_setup();
5  bool esp_connection(char type);
6  void esp_transmit_once(char* data);
7  void check_interval();
8
9  /******************/
10 /* ESP8266 setup */
11 /******************/
12 void esp_setup() {
13   ESP8266.begin(ESP_baudrate);
14 }
15
16 /**********************/
17 /* Check  connection */
18 /**********************/
19 bool esp_connection(char type) {
20
21   count_i=0;
22   unsigned char connection = 48;
23
24   do {
25     delay(5000);
26
27     while(ESP8266.available()) {ESP8266.read();} //Clear receive before begin buffer
28     ESP8266.print(type);
29     delay(200);
30     connection = ESP8266.read();
31
32     if (connection == 49) {
33       Serial.println(connection);
34       return true;
35     }
36
37     Serial.println(connection);
38
39     digitalWrite(MSFT_esp, LOW);
40     delay(500);
41     digitalWrite(MSFT_esp, HIGH);
42
43     count_i++;
44   } while (count_i < 15);
45
46     return false;
47 }
48
49 /*****************************/
50 /* Transmit data to ESP8266 */
51 /*****************************/
52 bool esp_transmit(char type, String data, int &success, bool &fail) {
53   ESP8266.print(type); //Tell ESP8266 is upload time
54   delay(20);
55
56   char temp[37] = {'\0'};
57
58   data.toCharArray(temp,37);
59   count_i=0;
60
61   while (temp[count_i] != NULL) {
62     ESP8266.print(temp[count_i++]);
63   }
64
65   unsigned char firebase = '2';
66   count_i=0;
67   //Read Firebase status from D1
68     do {
69       firebase = ESP8266.read(); //get the raw data
70       Serial.println(firebase);
71       if(firebase == 49) {
72         success++;
73         return true;
74       } else if (firebase == 48 or count_i == 100) {
```

Figure C-2-1: ESP8266.h Program (1)

```
75          fail = true;
76          return true;
77        } else if (firebase == 255 or firebase == 0) {
78        }
79        delay(1000);
80        count_i++;
81        firebase = '2';
82      } while (firebase == '2');
83  }
84
85  /*********************************/
86  /* Transmit data to ESP8266 ONCE*/
87  /*********************************/
88  void esp_transmit_once(int *data) {
89
90    String temp;
91    char tempArray[100];
92
93    count_i=0;
94    while (data[count_i] != 1027) { //1027 = END OF LINE
95      if (data[count_i] == 1026) { //1026 = ,
96        temp += ",";
97      } else if (data[count_i] == 1025) { //1025 = -
98        temp += "-";
99      } else {
100       temp += data[count_i];
101     }
102     count_i++;
103   }
104   temp += '/';
105
106   temp.toCharArray(tempArray,50);
107
108   count_i=0;
109    while (tempArray[count_i] != '/') {
110      Serial.print(tempArray[count_i++]);
111    }
112    Serial.println(" ");
113
114   ESP8266.print('n'); //Tell ESP8266 is upload time
115   delay(20);
116
117   count_i=0;
118   while (tempArray[count_i] != '/') {
119 //    Serial.print(data[count_i++]);
120     ESP8266.print(tempArray[count_i++]); //send the value pointed by the pointer
121   }
122
123 }
124
125 /*************************************/
126 /* Get Latest Interval From Firebase */
127 /*************************************/
128 void check_interval() {
129
130   unsigned char temp = rtcc_interval;
131   count_i=0;
132   rtcc_interval = 51;
133
134 //  Serial.println("Checking Interval");
135
136   do {
137     while(ESP8266.available()) {ESP8266.read();} //Clear receive before begin buffer
138     ESP8266.print('i');
139     delay(3000);
140     rtcc_interval = ESP8266.read();
141
142     if (rtcc_interval == 48 || rtcc_interval == 49 || rtcc_interval == 50) {
143       Serial.println(rtcc_interval);
144       return;
145     }
146     count_i++;
147
148   } while (count_i <= 15);
149
150 //  Serial.println("NOPE");
151   rtcc_interval = temp;
152 }
```

Figure C-2-2: ESP8266.h Program (2)

```
#define iRef 1.077

void get_batt(int *data) {
  ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
  delay(3);
  ADCSRA |= _BV(ADSC);
  while (bit_is_set(ADCSRA, ADSC)); //wait until converion complete
  uint16_t result = ADCL; //get first half of result
  result |= ADCH << 8; //get second half of result

//  Serial.println(result);

  float batVolt = (iRef/result) * 1023;

  *data = 1026; //Convert previous logged 1027 to 1026
  data++;
  *data = (int)batVolt;
  data++;
  *data = (int)(batVolt*100)%100;
  data++;
  *data = 1027; //1026 = ,

}
```

Figure C-3: battery.h Program

```
void get_turbidity(char *data);

/************************************/
/* Get TURBIDITY value from sensor */
/************************************/
void get_turbidity(int *data) {

  uint16_t volt=0;

  for(uint8_t i=0; i<255; i++)
  {
      volt += analogRead(tub_sen_pin);
      volt /= 2;
  }

  *data = 1026; //Convert previous logged 1027 to 1026
  data++;
  *data = volt;
  data++;
  *data = 1027;

  return;
}
```

Figure C-4: turbidity.h Program

```
#define N_DECIMAL_POINTS_PRECISION (100) // n = 3. Three decimal points.

/*****************************/
/* Get pH value from sensor */
/*****************************/
void get_ph (int *data) {

  uint16_t volt=0;

  for(uint8_t i=0; i<255; i++)
  {
      volt += analogRead(ph_sen_pin);
      volt /= 2;
  }

//  Serial.println(volt);

  *data = 1026; //Convert previous logged 1027 to 1026
  data++;
  *data = volt;
  data++;
  *data = 1027;

  return;
}
```

Figure C-5: ph.h Program

```

#define N_DECIMAL_POINTS_PRECISION (100) // n = 3. Three decimal points.

OneWire oneWire(tem_sen_pin);        // Setup a oneWire instance to communicate with any OneWire devices
DallasTemperature sensors(&oneWire); // Pass our oneWire reference to Dallas Temperature sensor


void tem_setup();
void get_tem(int *data);

/***********************************/
/* call to SETUP temperature sensor */
/***********************************/
void tem_setup() {
    sensors.begin();
}

/*************************************/
/* Get TEMPERATURE value from sensor */
/*************************************/
void get_tem(int *data) {

  static float tmp_temp = 28.88;

  // Call sensors.requestTemperatures() to issue a global
  //temperature and Requests to all devices on the bus
  sensors.requestTemperatures();

    f_sensorValue = sensors.getTempCByIndex(0);

    *data = 1026; //Convert previous logged 1027 to 1026
    data++;
    *data = (int)f_sensorValue;
    data++;
    i_temp = (int)(f_sensorValue*100)%100;
    if (i_temp < 10) {
      *data = 0;
      data++;
      *data = i_temp;
    } else {
      *data = i_temp;
    }
    data++;
    *data = 1027;

    return;
}
```

Figure C-6: temperature.h Program

```
 File file;

bool sd_exist();
//bool sd_txt_check();
void sd_write(String data);
void sd_read();

/***************************/
/* Check if SD CARD exist */
/***************************/
bool sd_exist() {
  if (!SD.begin(sdcard_pin)) { //Cloud on select slave
    return false;
  } else {
    return true;
  }
}

/*********************/
/* write to SD CARD */
/*********************/
void sd_write(int *data) {

   String dateS;
   count_i=0;

   file = SD.open("logdata.txt", FILE_WRITE);

   if (file) {
//       Serial.println("Open file (write)");

      while (data[count_i] != 1027) { //1027 = END OF LINE
        if (data[count_i] == 1026) { //1026 = ,
          file.print(',');
        }  else if (data[count_i] == 1025) { //1025 = -
          file.print('-');
        } else {
          file.print(data[count_i]);
        }
        count_i++;
      }

      file.print("\n");
//       Serial.println(" ");

      file.close();
   } else {
//       Serial.println("Could not open file (write)");
   }

   //Create a backup file for data processing using computer
   dateS += data[0];
   dateS += data[2];
   dateS += data[4];
   dateS += ".csv";
//   Serial.println(dateS);
```

Figure C-7-1: sdcard.h Program (1)

```
58
59    file = SD.open(dateS, FILE_WRITE);
60
61    count_i=0;
62    if (file) {
63      while (data[count_i] != 1027) { //1027 = END OF LINE
64        if (data[count_i] == 1026) { //1026 = ,
65          file.print(',');
66        } else if (data[count_i] == 1025) { //1025 = -
67          file.print('-');
68        } else {
69          file.print(data[count_i]);
70        }
71        count_i++;
72      }
73      file.print("\n");
74      file.close();
75  //      Serial.println(F("create"));
76    } else {
77  //    Serial.println(F("no create"));
78    }
79
80  }
81
82  /*************************************/
83  /* Read from SD CARD                 */
84  /* This function only work with UART */
85  /*************************************/
86  String data;
87  int row=0;
88  int success=0;
89  bool fail=false;
90  void sd_read() {
91
92   file = SD.open("logdata.txt", FILE_READ);
93   if (file) {
94  //   Serial.println("Open file (read)");
95  //   Serial.println("--- Upload start ---");
96
97     while (file.available()) {
98       data = file.readStringUntil('\n');
99       Serial.println(data);
100 //      esp_transmit('u', data, success, fail);
101      delay(500);
102      while (esp_transmit('u', data, success, fail) == 0);
103      row++;
104      if (fail == true) {
105        break;
106      }
107      //delay(1000);
108    }
109      file.close();
110
111      if (row == success) {
112        SD.remove("logdata.txt");
113        Serial.println(F("Upload   - Success"));
114      } else {
115        Serial.println(F("Upload   - Fail"));
116      }
117 //   Serial.println("--- Upload end ---");
118
119   } else {
120      Serial.println(F("Could not open file (read)"));
121   }
122
123 }
```

Figure C-7-2: sdcard.h Program (2)

```
DS1307 RTC;

void rtcc_setup();
void get_rtcc(char *data_now);
void rtcc_check();

/***********************/
/* call to SETUP RTCC */
/***********************/
void rtcc_setup() {
  Wire.begin();
  RTC.begin();
  if (! RTC.isrunning()) {
    //auto update from computer time
    RTC.adjust(DateTime(__DATE__, __TIME__));
  }
}

/*****************************/
/* Call to GET current RTCC */
/*****************************/
void  get_rtcc(int *data) {

  byte temp;

//  Serial.println(F("GET RTCC INITIATE"));

  DateTime now = RTC.now();

//  Serial.println(F("GET RTCC SUCCESS"));

  *data = now.day();
  data++;

  *data = 1025; //1025 = -
  data++;

  *data = now.month();
  data++;

  *data = 1025; //1025 = -
  data++;

  *data = now.year();
  data++;

  *data = 1026; //1026 = ,
  data++;

  temp = now.hour();
  if (temp < 10) {
    *data = 0;
    data++;
    *data = temp;
    data++;
  } else {
    *data = temp;
    data++;
  }

  temp = now.minute();
  if (temp == 0) {
    *data = 0;
    data++;
    *data = temp;
    data++;
  } else {
    *data = temp;
    data++;
  }

//  *data = now.second();
//  data++;

  *data = 1027; //1027 = to check array length
  data++;

}

void rtcc_check() {
  DateTime now = RTC.now();
  byte temp;

  temp = now.second();
  if ((temp%10+48) == '0') {
    if ((temp/10+48) == '0') {
      temp = now.minute();
      if ((temp%10+48) == '0') {
        if (rtcc_interval == 48) { //Log data every 10 minutes
          logData = true;
        }
        if ((temp/10+48) == '0') {
          if (rtcc_interval == 50 || rtcc_interval == 49) { //Log data every 60 minutes or every 30 minutes
            logData = true;
          }
          temp = now.hour(); //Check for any interval changes
          if ((temp/10+48) == '0') {
            if ((temp%10+48) == '0') {
              CheckInterval = true;
//              Serial.println("CheckInterval");
            }
          }
        } else if ((temp/10+48) == '3') {
          if (rtcc_interval == 49) { //Log data every 30 minutes
            logData = true;
          }
        }
      }
    }
  }

//  CheckInterval = true;
//  logData = true;

}
```

Figure C-8: rtcc.h Program

```
#include "FirebaseESP8266.h"
#include <ESP8266WiFi.h>
//#include <SoftwareSerial.h>

//Include class
#include "Firebase.h";
#include "Wifi.h"

//Declare class and object
FIREBASE firebase;
WIFI wifi;

//SoftwareSerial Serial(D6,D7); //RX,TX for NANO

#define FIREBASE_HOST "missi-82163.firebaseio.com"
#define FIREBASE_AUTH "g8OndO73bbgPRhC02AmQEsKdCMF1zwiSArr7xmXC"

void setup() {
  Serial.begin(9600);

  wifi.setup();

}

void loop() {
//Declare variable for main loop
char check_pipe;
bool wifi_connection=false, nano_connection=false,
firebase_connection=false, upload_status=false;
bool newData = false;
String data_string;
String date;
String rtc;
String measurement;
String YEAR;
String MONTH;

check_firebase_log();

if (Serial.available() > 0) {

  //Read from Nano
  check_pipe = Serial.read();

  switch(check_pipe) {
    case 'c': {
//       Serial.println("------------Communication Begin---------------");
      Serial.print(true); //send the connection result
//       Serial.println("Nano     - Connected");
      break; }
    case 'w': {
      wifi_connection = wifi.status(); //check wifi connection
      Serial.print(wifi_connection); //send wifi result
//       Serial.println("WiFi     - Connected");
      break; }
     case 'u': {
      delay(500);
      data_string = read_data_string(newData);
      newData = false;
      data_sorting(data_string, date, rtc, measurement);
      date_sorting(date,YEAR,MONTH);
      upload_status = firebase.upload(date, rtc, measurement, YEAR, MONTH);
      Serial.print(upload_status);
      break; }
     case 'i': {
       Serial.print(firebase.get_interval());
//         Serial.print('1');
     break; }
     case 'n': {
      delay(500);
      data_string = read_data_string(newData);
      newData = false;
      data_sorting(data_string, date, rtc, measurement);
      date_sorting(date,YEAR,MONTH);
      upload_status = firebase.upload_now(date, rtc, measurement, YEAR, MONTH);
```

Figure C-9-1: ESP8266 Main Program (1)

```
//       Serial.print(upload_status);
     break; }
   }

 }
}

String read_data_string(bool &newData) {
 char endMarker = '\n';
 char rc;
 String str;

    while (Serial.available() > 0 && newData == false) {
        rc = Serial.read();
        //Serial.println(rc);
        if (rc != endMarker) {
            if (rc == '0' || rc == '1' || rc == '2' || rc == '3' || rc == '4'
            || rc == '5' || rc == '6' || rc == '7' || rc == '8' || rc == '9'
            || rc == ',' || rc == '-' || rc == ':' || rc == '.') {
              str += rc;
            }
        } else {
            newData = true;
        }
    }
  return str;
}

void data_sorting(String data, String &date, String &rtc, String &measurement) {
  char *strings[8];
  char *ptr = NULL;
  String date2;
  char char_array[15];
  char temp[40];
  data.toCharArray(temp, 50);

    byte index = 0;
    ptr = strtok(temp, ",");  // takes a list of delimiters
    while(ptr != NULL) {
        strings[index] = ptr;
        index++;
        ptr = strtok(NULL, ",");  // takes a list of delimiters
    }
    //Serial.println(index);

   date = String(strings[0]);
   rtc = String(strings[1]);

   //Change the + according to how many sets of parameters*****
   measurement = String(strings[2]) + "," + String(strings[3])
   + "," + String(strings[4]) + "," + String(strings[5]);
}

void date_sorting(String date, String &YEAR, String &MONTH) {
  char *strings[8];
  char *ptr = NULL;
  char temp[40];
  date.toCharArray(temp, 50);

    byte index = 0;
    ptr = strtok(temp, "-");  // takes a list of delimiters
    while(ptr != NULL) {
        strings[index] = ptr;
        index++;
        ptr = strtok(NULL, "-");  // takes a list of delimiters
    }
    //Serial.println(index);
   /* for(int n = 0; n < index; n++)
   {
    Serial.println(strings[n]);
   }*/

   YEAR = String(strings[2]);
   MONTH = String(strings[1]);
}
```

Figure C-9-2: ESP8266 Main Program (2)

```
#include "FirebaseESP8266.h"
FirebaseData firebaseData;

class FIREBASE{
public:

bool upload(String date, String rtc, String data, String YEAR, String MONTH) {
#define FIREBASE_HOST "missi-82163.firebaseio.com"
#define FIREBASE_AUTH "g8OndO73bbgPRhC02AmQEsKdCMF1zwiSArr7xmXC"
Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);

  String path = "/Farm1/RawData";
  //Serial.println(data);
  String tub;
  String tem;
  String ph;
  String DO;

  data_sorting(data,tub,tem,ph,DO);

   if (Firebase.setString(firebaseData, path + "/Graph/" + YEAR + "/" +
     MONTH + "/" + date + "/" + rtc, data)) {
      Firebase.setString(firebaseData, path + "/Notification/date", date);
      Firebase.setString(firebaseData, path + "/Notification/ph", ph);
      Firebase.setString(firebaseData, path + "/Notification/DO", DO);
      Firebase.setString(firebaseData, path + "/Notification/turbidity", tub);
      Firebase.setString(firebaseData, path + "/Notification/temperature", tem)
      delay(50);
      Firebase.setString(firebaseData, path + "/Notification/time", rtc);
//      Serial.println("Upload   - Success");
      return true;
   } else {
//      Serial.println("Upload   - Fail");
      return false;
   }
}

char get_interval() {
#define FIREBASE_HOST "missi-82163.firebaseio.com"
#define FIREBASE_AUTH "g8OndO73bbgPRhC02AmQEsKdCMF1zwiSArr7xmXC"
Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);

  String path = "/Farm1/Others";
  char Array[2];

  if (Firebase.getString(firebaseData, path + "/Interval")) {
    if (firebaseData.dataType() == "string") {

      firebaseData.stringData().toCharArray(Array,2);

      return(Array[0]);
//     return(true);
    }
  } else {
    return('q');
  }

}

bool upload_now(String date, String rtc, String data, String YEAR, String MONTH
#define FIREBASE_HOST "missi-82163.firebaseio.com"
#define FIREBASE_AUTH "g8OndO73bbgPRhC02AmQEsKdCMF1zwiSArr7xmXC"
Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);

  String path = "/Farm1/RawData/Notification";
  //Serial.println(data);
  String tub;
  String tem;
  String ph;
  String DO;

  data_sorting(data,tub,tem,ph,DO);
```

Figure C-10-1: firebase.h Program (1)

```
73
74      if (Firebase.setString(firebaseData, path + "/date", date)) {
75        Firebase.setString(firebaseData, path + "/turbidity", tub);
76        Firebase.setString(firebaseData, path + "/temperature", tem);
77        Firebase.setString(firebaseData, path + "/ph", ph);
78        Firebase.setString(firebaseData, path + "/DO", DO);
79        delay(50);
80        Firebase.setString(firebaseData, path + "/time", rtc);
81  //      Serial.println("Upload   - Success");
82        return true;
83      } else {
84  //      Serial.println("Upload   - Fail");
85        return false;
86      }
87  }
88
89  void data_sorting(String data, String &tub, String &tem, String &ph, String &DO
90    char *strings[10];
91    char *ptr = NULL;
92    char temp[40];
93    data.toCharArray(temp, 50);
94
95      byte index = 0;
96      ptr = strtok(temp, ",");  // takes a list of delimiters
97      while(ptr != NULL) {
98          strings[index] = ptr;
99          index++;
100         ptr = strtok(NULL, ",");  // takes a list of delimiters
101     }
102     //Serial.println(index);
103    /* for(int n = 0; n < index; n++)
104    {
105     Serial.println(strings[n]);
106    }*/
107    tub = String(strings[0]);
108    tem = String(strings[1]);
109    ph = String(strings[2]);
110    DO = String(strings[3]);
111 }
112
113 };
114
115 void check_firebase_log() {
116 #define FIREBASE_HOST "missi-82163.firebaseio.com"
117 #define FIREBASE_AUTH "g8OndO73bbgPRhC02AmQEsKdCMFlzwiSArr7xmXC"
118 Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
119
120   String path = "/Farm1/Others";
121   char Array[2];
122
123   if (Firebase.getString(firebaseData, path + "/Log")) {
124     if (firebaseData.dataType() == "string") {
125
126       firebaseData.stringData().toCharArray(Array,2);
127
128       if (Array[0] == '1') {
129         Firebase.setString(firebaseData, path + "/Log", "0");
130         Serial.print('8');
131       }
132
133     }
134   }
135
```

Figure C-10-2: firebase.h Program (2)

```
class WIFI{
public:

#define WIFI_SSID "DESKTOP-Q15U6KM 8221"
#define WIFI_PASSWORD "52-4vD82"
//#define WIFI_SSID "Chew Family"
//#define WIFI_PASSWORD "22Jp8+b9^cyL5AZ"

void setup(){
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
//  Serial.print("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED) { //wait until Wifi is connected only proceed
    Serial.print(".");
    delay(50);
  }
//  Serial.println();
//  Serial.print("connected: ");
//  Serial.println(WiFi.localIP());
}

bool status() {
  if(WiFi.status()== WL_CONNECTED){   //Check WiFi connection status
//    Serial.println("WiFi     - Connected");
    return true;
  }else{
//    Serial.println("WiFi     - Disconnected");
    return false;
  }
}


};
```

Figure C-11: wifi.h Program

```
'use-strict'

const functions = require('firebase-functions');
const admin = require('firebase-admin');
admin.initializeApp(functions.config().firebase);

//Check sensor value if above threshold
exports.sendNotification = functions.database.ref('/Farm1/Notification')
.onWrite((change, context) => {

    const topic = "waterParameters";

    const before = change.before.val()
    const after = change.after.val()

    if (!(before.DO === after.DO)) {

        // const topic = "dissovledOxygen";
        const DO_value = after.DO; //EDIT HERE

            console.log('Dissolved Oxygen Currently at : ', DO_value); //EDIT HERE
            console.log('Sending to topic : ', topic); //EDIT HERE

            if (DO_value < 4) { //EDIT HERE

                const payload = {
                    notification: {
                    title: "Dissolved Oxygen Alert!!!", //EDIT HERE
                    body: "Your Dissolved Oxygen is currently below 4mg/L", //EDIT HERE
                    icon: "default",
                    sound: "default"
                    }
                };

                return admin.messaging().sendToTopic(topic,payload).then(response => {
                    return console.log('This was the notification feature');
                    });

            } else {
                return null;
            }
    }

    if (!(before.ph === after.ph)) {

        // const topic = "pH";
        const ph_value = after.ph; //EDIT HERE

            console.log('pH Currently at : ', ph_value); //EDIT HERE
            console.log('Sending to topic : ', topic); //EDIT HERE

            if (ph_value > 9) { //EDIT HERE

                const payload = {
                    notification: {
                    title: "pH Alert!!!", //EDIT HERE
                    body: "Your pH is currently above 9" , //EDIT HERE
                    icon: "default",
                    sound: "default"
                    }
                };

                return admin.messaging().sendToTopic(topic,payload).then(response => {
                    return console.log('This was the notification feature');
                    });

            } else {
                return null;
            }
    }

    if (!(before.temperature === after.temperature)) {

        // const topic = "temperature";
        const tem_value = after.temperature; //EDIT HERE

            console.log('Temperature Currently at : ', tem_value); //EDIT HERE

            if (tem_value > 35) { //EDIT HERE

                const payload = {
                    notification: {
                    title: "Temperature Alert!!!", //EDIT HERE
                    body: "Your Temperature is above 35 degree celsius",
                    icon: "default",
                    sound: "default"
                    }
                };

                return admin.messaging().sendToTopic(topic,payload).then(response => {
                    return console.log('This was the notification feature');
                    });

            } else {
                return null;
            }
    }

});

```

Figure C-12-1: Firebase Function Program(1)

```
//Reformat data
exports.updateNotification = functions.database.ref('/Farml/RawData/Notification')
.onWrite((change, context) => {

    const before = change.before.val()
    const after = change.after.val()

    if (!(before.time === after.time)) {

        var date = after.date;
        var time = after.time;
        var DO = after.DO;
        var ph = after.ph;
        var turbidity = after.turbidity;
        var temperature = after.temperature;

        //Format Date
        admin.database().ref("/Farml/Notification").child("date").set(date);

        //Format Time
        // if (time < 1000) {
        //     const array = temperature.split('');
        //     time = "0";
        //     time += array[0];
        //     time += array[1];

        //     if (array[2] === undefined) {
        //         time += "0";
        //     } else {
        //         time += array[2];
        //     }
        // } else {

        // }
        admin.database().ref("/Farml/Notification").child("time").set(time);

        //Format DO
        DO = (DO/1023.00)*5.00;
        DO = (DO*28.80)/5.0;
        DO = Math.round(parseFloat((DO * Math.pow(10, 2)).toFixed(2))) / Math.pow(10, 2);
        DO = DO.toString();
        admin.database().ref("/Farml/Notification").child("DO").set(DO);

        //Format temperature
        array = temperature.split('');
        temperature = array[0];
        temperature += array[1];
        temperature += '.';
        temperature += array[2];
        if (array[3] === undefined) {
            temperature += "0";
        } else {
            temperature += array[3];
        }
        admin.database().ref("/Farml/Notification").child("temperature").set(temperature);

        //Format pH
        ph = (ph/1023.00)*5.00;
        ph = (-5.17*ph)+15.85;
        ph = Math.round(parseFloat((ph * Math.pow(10, 2)).toFixed(2))) / Math.pow(10, 2);
        ph = ph.toString();
        admin.database().ref("/Farml/Notification").child("ph").set(ph);

        //Format turbidity
        turbidity = (turbidity/1023)*5;
        turbidity = -1120*((turbidity)*(turbidity))+5742*turbidity-4353;
        if (turbidity < 0) {
            turbidity = "0";
        } else {
            turbidity = Math.round(parseFloat((turbidity * Math.pow(10, 2)).toFixed(2))) / Math.pow(10, 2);
            turbidity = turbidity.toString();
        }
        admin.database().ref("/Farml/Notification").child("turbidity").set(turbidity);

        return NULL;
    }

});
```

Figure C-12-2: Firebase Function Program(2)

```
//Format Graph
exports.updateGraph = functions.database.ref('/Farml/RawData/Graph/{YEAR}/{MONTH}/{DATE}/{TIME}')
.onCreate((snapshot, context) => {

    const YEAR = context.params.YEAR;
    const MONTH = context.params.MONTH;
    const DATE = context.params.DATE;
    const TIME = context.params.TIME;

    var data = snapshot.val();
    var array = data.split(',');

    var turbidity = array[0];
    var temperature = array[1];
    var ph = array[2];
    var DO = array[3];

    //Format turbidity
    turbidity = (turbidity/1023)*5;
    turbidity = -1120*((turbidity)*(turbidity))+5742*turbidity-4353;
    if (turbidity < 0) {
        turbidity = "0";
    } else {
        turbidity = Math.round(parseFloat((turbidity * Math.pow(10, 2)).toFixed(2))) / Math.pow(10, 2);
        turbidity = turbidity.toString();
    }
    data = turbidity;
    data += ',';

    //Format temperature
    array = temperature.split('');
    temperature = array[0];
    temperature += array[1];
    temperature += '.';
    temperature += array[2];
    if (array[3] === undefined) {
        temperature += "0";
    } else {
        temperature += array[3];
    }
    data += temperature;
    data += ',';

    //Format pH
    ph = (ph/1023.00)*5.00;
    ph = (-5.17*ph)+15.85;
    ph = Math.round(parseFloat((ph * Math.pow(10, 2)).toFixed(2))) / Math.pow(10, 2);
    ph = ph.toString();
    data += ph;
    data += ',';

    //Format DO
    DO = (DO/1023.00)*5.00;
    DO = (DO*28.80)/5.0;
    DO = Math.round(parseFloat((DO * Math.pow(10, 2)).toFixed(2))) / Math.pow(10, 2);
    DO = DO.toString();
    data += DO;

    return admin.database().ref("/Farml/Graph").child(YEAR).child(MONTH).child(DATE).child(TIME).set(data);

});
```

Figure C-12-3: Firebase Function Program(3)

```java
package com.example.fyp;

import android.app.Notification;
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.os.Build;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.v4.app.NotificationCompat;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.google.firebase.messaging.FirebaseMessaging;

import java.util.Map;
import java.util.Random;

public class MainActivity extends AppCompatActivity {

    Button btn_hour,btn_log,btn_10,btn_30,btn_60;

    //Cloud messaging SETUP
    private static final String CHANNEL_ID= "Channel ID";
    private static final String CHANNEL_NAME = "Android Push Notification";
    private static final String CHANNEL_DESC = "Android Push Notification";

    //Firebase SETUP
    private FirebaseDatabase mFirebaseDatabase;
    private DatabaseReference myRef;
    private ValueEventListener listener;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_page);

        btn_hour = (Button) findViewById(R.id.button_hour);
        btn_log = (Button) findViewById(R.id.log_now);
        btn_10 = (Button) findViewById(R.id.min10);
        btn_30 = (Button) findViewById(R.id.min30);
        btn_60 = (Button) findViewById(R.id.min60);
        final TextView date = (TextView) findViewById(R.id.date);
        final TextView time = (TextView) findViewById(R.id.time);
        final TextView turbidity = (TextView) findViewById(R.id.turbidity);
        final TextView temperature = (TextView) findViewById(R.id.temperature);
        final TextView ph = (TextView) findViewById(R.id.ph);
        final TextView DO = (TextView) findViewById(R.id.DO);

        //Firebase
        mFirebaseDatabase = FirebaseDatabase.getInstance();
        myRef = mFirebaseDatabase.getReference().child("Farm1/");
            //date
        listener = myRef.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
//                String LATEST_DATA = dataSnapshot.child("LATEST_DATA").getValue().toString();
//
//                String dataArray[] = LATEST_DATA.split(",");

                String temp_date = dataSnapshot.child("Notification/date").getValue().toString();
                String temp_time = dataSnapshot.child("Notification/time").getValue().toString();
                String temp_turbidity = dataSnapshot.child("Notification/turbidity").getValue().toString();
                String temp_temperature = dataSnapshot.child("Notification/temperature").getValue().toString();
                String temp_ph = dataSnapshot.child("Notification/ph").getValue().toString();
                String temp_DO = dataSnapshot.child("Notification/DO").getValue().toString();
                float temp;

                date.setText("Last log date: " + temp_date);
                time.setText("Last log time: " + temp_time);
                turbidity.setText("Turbidity: " + temp_turbidity + " NTU");
                temperature.setText("Temperature: " + temp_temperature + " Celsius");
                ph.setText("pH: " + temp_ph);
                DO.setText("Dissolved Oxygen: " + temp_DO +" mg/L");

                //Set colour of the text
                date.setTextColor(Color.parseColor("#000000"));
                time.setTextColor(Color.parseColor("#000000"));

                temp = Float.parseFloat(temp_temperature);
                if (temp > 42) {
                    temperature.setTextColor(Color.parseColor("#cc0000"));
                } else {
                    temperature.setTextColor(Color.parseColor("#009933"));
                }

                temp = Float.parseFloat(temp_ph);
                if (temp > 9) {
                    ph.setTextColor(Color.parseColor("#cc0000"));
                } else {
                    ph.setTextColor(Color.parseColor("#009933"));
                }
```

Figure C-13-1: Android Main Activity Program (1)

```
103
104                 temp = Float.parseFloat(temp_DO);
105                 if (temp < 4) {
106                     DO.setTextColor(Color.parseColor("#cc0000"));
107                 } else {
108                     DO.setTextColor(Color.parseColor("#009933"));
109                 }
110             }
111             @Override
112             public void onCancelled(@NonNull DatabaseError databaseError) {
113             }
114         });
115
116         //Cloud messaging
117         createNotificationChannel();
118         String topic = "waterParameters"; //TOPIC TITLE CAN EDIT HERE!!!
119         FirebaseMessaging.getInstance().subscribeToTopic(topic);
120
121         //Button
122         btn_hour.setOnClickListener(new View.OnClickListener(){
123             @Override
124             public void onClick(View view) {
125                 //Intent intent = Hour_Graph.makeIntent(MainActivity.this, "49-05-2019",1);
126                 //startActivity(intent);
127                 Intent intent = new Intent(MainActivity.this, Year.class);
128                 startActivity(intent);
129             }
130         });
131         btn_log.setOnClickListener(new View.OnClickListener(){
132             @Override
133             public void onClick(View view) {
134                 myRef.child("Others/Log").setValue("1");
135             }
136         });
137         btn_10.setOnClickListener(new View.OnClickListener(){
138             @Override
139             public void onClick(View view) {
140                 myRef.child("Others/Interval").setValue("0");
141                 Toast.makeText(getApplicationContext(),"Interval set to 10 Minutes", Toast.LENGTH_SHORT).show();
142 //                Toast.makeText(getApplicationContext(),"Changes will take effect on 12AM", Toast.LENGTH_SHORT).show();
143             }
144         });
145         btn_30.setOnClickListener(new View.OnClickListener(){
146             @Override
147             public void onClick(View view) {
148                 myRef.child("Others/Interval").setValue("1");
149                 Toast.makeText(getApplicationContext(),"Interval set to 30 Minutes", Toast.LENGTH_SHORT).show();
150 //                Toast.makeText(getApplicationContext(),"Changes will take effect on 12AM", Toast.LENGTH_SHORT).show();
151             }
152         });
153         btn_60.setOnClickListener(new View.OnClickListener(){
154             @Override
155             public void onClick(View view) {
156                 myRef.child("Others/Interval").setValue("2");
157                 Toast.makeText(getApplicationContext(),"Interval set to 30 Minutes", Toast.LENGTH_SHORT).show();
158 //                Toast.makeText(getApplicationContext(),"Changes will take effect on 12AM", Toast.LENGTH_SHORT).show();
159             }
160         });
161
162     }
163
164     private void createNotificationChannel() {
165         if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.O){
166             NotificationManager manager = getSystemService(NotificationManager.class);
167             NotificationChannel channel
168                     = new NotificationChannel(CHANNEL_ID,CHANNEL_NAME, NotificationManager.IMPORTANCE_DEFAULT);
169
170             channel.setDescription(CHANNEL_DESC);
171
172             manager.createNotificationChannel(channel);
173         }
174     }
175
176 }
177
```

Figure C-13-2: Android Main Activity Program (2)

```java
package com.example.fyp;

import android.content.Context;
import android.content.Intent;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

import java.util.ArrayList;

public class Hour extends AppCompatActivity {

    private static final String TAG = "ViewDatabase";
    private ListView mListView;

    //Firebase SETUP
    private FirebaseDatabase mFirebaseDatabase;
    private DatabaseReference myRef;
    private ValueEventListener listener;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hour);

        mListView = (ListView) findViewById(R.id.listview);

        //Parse data
        String YEAR = extractYearFromIntent();
        String MONTH = extractMonthFromIntent();
//        Toast.makeText(Hour.this, MONTH,Toast.LENGTH_SHORT).show();

        //Firebase
        mFirebaseDatabase = FirebaseDatabase.getInstance();
        myRef = mFirebaseDatabase.getReference().child("Farml/Graph").child(YEAR).child(MONTH);

        listener = myRef.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                showData(dataSnapshot);
                myRef.removeEventListener(this);
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {

            }
        });

    }

    //Firebase
    private void showData(DataSnapshot dataSnapshot) {

        //Parse data
        final String YEAR = extractYearFromIntent();
        final String MONTH = extractMonthFromIntent();

        //Retrieve date(s) from firebase
        int h=0;
        ArrayList<String> arraylist = new ArrayList<>();
        ArrayList<String> listview = new ArrayList<>();
        for(DataSnapshot ds : dataSnapshot.getChildren()) {
            //for backend process
            String name = ds.getKey();
            arraylist.add(name);

            //for display purpose
//              name = name.replace("-19","-2019");
            listview.add(name);
            h++;
        }
        ArrayAdapter adapter = new ArrayAdapter(this,android.R.layout.simple_list_item_1,listview);
        mListView.setAdapter(adapter);

        //Make list view clickable
        String[] temp = new String[arraylist.size()];
        temp = arraylist.toArray(temp);
        final String[] array = temp;

```

Figure C-14-1: Android Hour Program (1)

```
        mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                //Toast.makeText(Hour.this, array[position],Toast.LENGTH_SHORT).show();
                Intent intent = Hour_Graph.makeIntent(Hour.this, YEAR, MONTH, array[position]);
                startActivity(intent);
            }
        });
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        myRef.removeEventListener(listener);
    }

    //Parsing data function
    public static Intent makeIntent(Context context, String year, String month, int num) {
        Intent intent = new Intent(context, Hour.class);
        intent.putExtra("Hour_year", year);
        intent.putExtra("Hour_month", month);
        return intent;
    }
    private String extractYearFromIntent() {
        Intent intent = getIntent();
        String YEAR = intent.getStringExtra("Hour_year");
        return YEAR;
    }
    private String extractMonthFromIntent() {
        Intent intent = getIntent();
        String MONTH = intent.getStringExtra("Hour_month");
        return MONTH;
    }

}
```

Figure C-14-2: Android Hour Program (2)

```
package com.example.fyp;

import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.support.annotation.NonNull;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.jjoe64.graphview.DefaultLabelFormatter;
import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.series.DataPoint;
import com.jjoe64.graphview.series.DataPointInterface;
import com.jjoe64.graphview.series.LineGraphSeries;
import com.jjoe64.graphview.series.OnDataPointTapListener;
import com.jjoe64.graphview.series.PointsGraphSeries;
import com.jjoe64.graphview.series.Series;

import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.ArrayList;

public class Hour_Graph extends AppCompatActivity {

    //private ListView mListView;

    //Firebase SETUP
    private FirebaseDatabase mFirebaseDatabase;
    private DatabaseReference myRef;

    //Graph SETUP
    GraphView graphView;
    LineGraphSeries series;

    float[][] dataRow = new float[(int)200][(int)200];
    String [] hourAxis = new String [(int)200];
//    int x[] = {0,1,2,3,4};
//    int y[] = {0,1,2,3};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hour_graph);

        //Parse data
        String YEAR = extractYearFromIntent();
        String MONTH = extractMonthFromIntent();
        String DATE = extractDateFromIntent();
//        Toast.makeText(Hour_Graph.this, DATE,Toast.LENGTH_SHORT).show();

        //Firebase
        mFirebaseDatabase = FirebaseDatabase.getInstance();
        myRef = mFirebaseDatabase.getReference("Farm1/Graph").child(YEAR).child(MONTH).child(DATE);

    }

    //Firebase
    @Override
    public void onStart() {
        super.onStart();
        myRef.addValueEventListener(new ValueEventListener() {

            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {

                ArrayList<String> AhourValue = new ArrayList<>();
                ArrayList<String> AdataValue = new ArrayList<>();
                String[] dataColumn;
                //String[][] dataRow = new String[(int)100][(int)100];
                //int[][] dataRow = new int[(int)100][(int)100];

                //Retrieve hour(s) from firebase
                for(DataSnapshot ds : dataSnapshot.getChildren()) {
                    AhourValue.add(ds.getKey());
                }

                //Retrieve data(s) from firebase
                int i=0;
                for(String hour : AhourValue) {
                    AdataValue.add(dataSnapshot.child(hour).getValue(String.class));
                    hourAxis[i] = hour;
                    i++;
                }

                //Data Sorting
                //AdataValue = (turbidity,Temperature,ph) <- sequence
                i=0;

                for(String data : AdataValue) {
                    dataColumn = data.split(","); //Seperate data to column
                    dataRow[i][0] = Float.parseFloat(dataColumn[0]); // Store data to row, column 0
                    dataRow[i][1] = Float.parseFloat(dataColumn[1]); // Store data to row, column 1
                    dataRow[i][2] = Float.parseFloat(dataColumn[2]); // Store data to row, column 2
                    dataRow[i][3] = Float.parseFloat(dataColumn[3]); // Store data to row, column 2
                    dataRow[i][4] = 500; //turbidity maximun
                    dataRow[i][5] = 0; //turbidity minumum
                    dataRow[i][6] = 35; //tempeture maximun
                    dataRow[i][7] = 25; //temperature minumum
                    dataRow[i][8] = 8; //ph maximun
                    dataRow[i][9] = 1; //ph minumum
                    dataRow[i][10] = 10; //do maximun
                    dataRow[i][11] = 7; //do minumum
                    i++;
                }
```

Figure C-15-1: Android Hour_Graph Coding (1)

```
115
116             LineGraphSeries<DataPoint> series;       //an Object of the PointsGraphSeries for plotting scatter graphs
117             LineGraphSeries<DataPoint> series1;
118             //Graph
119                 //Turbidity
120             GraphView tub_graph = (GraphView) findViewById(R.id.tub_graph);
121             series= new LineGraphSeries<>(data(i,0));   //initializing/defining series to get the data from the method 'data()'
122             tub_graph.addSeries(series);                  //adding the series to the GraphView
123             series.setOnDataPointTapListener(new OnDataPointTapListener() {
124                 @Override
125                 public void onTap(Series series, DataPointInterface dataPoint) {
126                     Toast.makeText(Hour_Graph.this, "[Position/Turbidity] -> "+dataPoint, Toast.LENGTH_SHORT).show();
127                 }
128             });
129             series1= new LineGraphSeries<>(data(i,4));
130             tub_graph.addSeries(series1);
131             series1.setColor(Color.RED);
132             series1= new LineGraphSeries<>(data(i,5));
133             tub_graph.addSeries(series1);
134             series1.setColor(Color.RED);
135                 //Temperature
136             GraphView tem_graph = (GraphView) findViewById(R.id.tem_graph);
137 //            LineGraphSeries<DataPoint> series;       //an Object of the PointsGraphSeries for plotting scatter graphs
138             series= new LineGraphSeries<>(data(i,1));   //initializing/defining series to get the data from the method 'data()'
139             tem_graph.addSeries(series);                  //adding the series to the GraphView
140             series.setOnDataPointTapListener(new OnDataPointTapListener() {
141                 @Override
142                 public void onTap(Series series, DataPointInterface dataPoint) {
143                     Toast.makeText(Hour_Graph.this, "[Position/Temperature] -> "+dataPoint, Toast.LENGTH_SHORT).show();
144                 }
145             });
146             series1= new LineGraphSeries<>(data(i,6));
147             tem_graph.addSeries(series1);
148             series1.setColor(Color.RED);
149             series1= new LineGraphSeries<>(data(i,7));
150             tem_graph.addSeries(series1);
151             series1.setColor(Color.RED);
152                 //pH
153             GraphView ph_graph = (GraphView) findViewById(R.id.ph_graph);
154             series= new LineGraphSeries<>(data(i,2));   //initializing/defining series to get the data from the method 'data()'
155             ph_graph.addSeries(series);                  //adding the series to the GraphView
156             series.setOnDataPointTapListener(new OnDataPointTapListener() {
157                 @Override
158                 public void onTap(Series series, DataPointInterface dataPoint) {
159                     Toast.makeText(Hour_Graph.this, "[Position/pH] -> "+dataPoint, Toast.LENGTH_SHORT).show();
160                 }
161             });
162             series1= new LineGraphSeries<>(data(i,8));
163             ph_graph.addSeries(series1);
164             series1.setColor(Color.RED);
165             series1= new LineGraphSeries<>(data(i,9));
166             ph_graph.addSeries(series1);
167             series1.setColor(Color.RED);
168                 //Dissolved Oxygen
169             GraphView do_graph = (GraphView) findViewById(R.id.do_graph);
170             series= new LineGraphSeries<>(data(i,3));   //initializing/defining series to get the data from the method 'data()'
171             do_graph.addSeries(series);                  //adding the series to the GraphView
172             series.setOnDataPointTapListener(new OnDataPointTapListener() {
173                 @Override
174                 public void onTap(Series series, DataPointInterface dataPoint) {
175                     Toast.makeText(Hour_Graph.this, "[Position/Dissolved Oxygen -> "+dataPoint, Toast.LENGTH_SHORT).show();
176                 }
177             });
178             series1= new LineGraphSeries<>(data(i,10));
179             do_graph.addSeries(series1);
180             series1.setColor(Color.RED);
181             series1= new LineGraphSeries<>(data(i,11));
182             do_graph.addSeries(series1);
183             series1.setColor(Color.RED);
184
185             //Format graph layout
186                 //Turbidity
187             tub_graph.getViewport().setXAxisBoundsManual(true); //Set manual range
188             tub_graph.getViewport().setMinX(0);
189             tub_graph.getViewport().setMaxX(10);
190             tub_graph.getViewport().setScrollable(true); // enables horizontal scrolling
191 //            tub_graph.getViewport().setScrollableY(true); // enables vertical scrolling
192 //            graph.getViewport().setScalable(true); // enables horizontal zooming and scrolling
193             series.setTitle("Turbidity");
194             tub_graph.getGridLabelRenderer().setVerticalAxisTitle("NTU");
195             tub_graph.getGridLabelRenderer().setHorizontalAxisTitle("Time");
196                 //Temperature
197             tem_graph.getViewport().setXAxisBoundsManual(true); //Set manual range
198             tem_graph.getViewport().setMinX(0);
199             tem_graph.getViewport().setMaxX(10);
200             tem_graph.getViewport().setScrollable(true); // enables horizontal scrolling
201 //            graph.getViewport().setScalable(true); // enables horizontal zooming and scrolling
202             series.setTitle("Temperature");
203             tem_graph.getGridLabelRenderer().setVerticalAxisTitle("DEGREE CELSIUS");
204             tem_graph.getGridLabelRenderer().setHorizontalAxisTitle("Time");
205                 //pH
206             ph_graph.getViewport().setXAxisBoundsManual(true); //Set manual range
207             ph_graph.getViewport().setMinX(0);
208             ph_graph.getViewport().setMaxX(10);
209             ph_graph.getViewport().setScrollable(true); // enables horizontal scrolling
210 //            graph.getViewport().setScalable(true); // enables horizontal zooming and scrolling
211             series.setTitle("pH");
212             ph_graph.getGridLabelRenderer().setVerticalAxisTitle("PH");
213             ph_graph.getGridLabelRenderer().setHorizontalAxisTitle("Time");
214                 //Dissolved Oxygen
215             do_graph.getViewport().setXAxisBoundsManual(true); //Set manual range
216             do_graph.getViewport().setMinX(0);
217             do_graph.getViewport().setMaxX(10);
218             do_graph.getViewport().setScrollable(true); // enables horizontal scrolling
219 //            graph.getViewport().setScalable(true); // enables horizontal zooming and scrolling
220             series.setTitle("Dissolved Oxygen");
221             do_graph.getGridLabelRenderer().setVerticalAxisTitle("mg/L");
222             do_graph.getGridLabelRenderer().setHorizontalAxisTitle("Time");
223
224             //Format graph X-axis
225                 //Turbidity
226             tub_graph.getGridLabelRenderer().setLabelFormatter(new DefaultLabelFormatter() {
227                 @Override
228                 public String formatLabel(double value, boolean isValueX) {
229                     if (isValueX) {
230                         // show normal x values
231                         return hourAxis[(int) value];
```

Figure C-15-2: Android Hour_Graph Coding (2)

```
                    } else {
                        // show currency for y values
                        return super.formatLabel(value, isValueX);
                    }
                }
            });
                //Temperature
            tem_graph.getGridLabelRenderer().setLabelFormatter(new DefaultLabelFormatter() {
                @Override
                public String formatLabel(double value, boolean isValueX) {
                    if (isValueX) {
                        // show normal x values
                        return hourAxis[(int) value];
                    } else {
                        // show currency for y values
                        return super.formatLabel(value, isValueX);
                    }
                }
            });
                //pH
            ph_graph.getGridLabelRenderer().setLabelFormatter(new DefaultLabelFormatter() {
                @Override
                public String formatLabel(double value, boolean isValueX) {
                    if (isValueX) {
                        // show normal x values
                        return hourAxis[(int) value];
                    } else {
                        // show currency for y values
                        return super.formatLabel(value, isValueX);
                    }
                }
            });
                //Dissolved Oxygen
            do_graph.getGridLabelRenderer().setLabelFormatter(new DefaultLabelFormatter() {
                @Override
                public String formatLabel(double value, boolean isValueX) {
                    if (isValueX) {
                        // show normal x values
                        return hourAxis[(int) value];
                    } else {
                        // show currency for y values
                        return super.formatLabel(value, isValueX);
                    }
                }
            });

        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
            Toast.makeText(Hour_Graph.this, "Failed to read data", Toast.LENGTH_SHORT).show();
            myRef.removeEventListener(this);
        }
    });
}

//Graph function
public DataPoint[] data(int n, int m){
    DataPoint[] values = new DataPoint[n];     //creating an object of type DataPoint[] of size 'n'
    for(int i=0;i<n;i++){
        DataPoint v = new DataPoint(i,dataRow[i][m]);
        values[i] = v;
    }
    return values;
}

//Parsing data function
public static Intent makeIntent(Context context, String year, String month, String date) {
    Intent intent = new Intent(context, Hour_Graph.class);
    intent.putExtra("Hour_Graph_year", year);
    intent.putExtra("Hour_Graph_month", month);
    intent.putExtra("Hour_Graph_date", date);
    return intent;
}
private String extractDateFromIntent() {
    Intent intent = getIntent();
    String DATE = intent.getStringExtra("Hour_Graph_date");
    return DATE;
}
private String extractYearFromIntent() {
    Intent intent = getIntent();
    String YEAR = intent.getStringExtra("Hour_Graph_year");
    return YEAR;
}
private String extractMonthFromIntent() {
    Intent intent = getIntent();
    String MONTH = intent.getStringExtra("Hour_Graph_month");
    return MONTH;
}
}
```

Figure C-15-3: Android Hour_Graph Coding (3)