

STATIC WATER QUALITY MONITORING SYSTEM

LIEW HAU RUN

**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**

Sept 2021

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 : Liew Hau Run


ID No. : 17UEB06622

Date : 22/09/2021

APPROVAL FOR SUBMISSION


I certify that this project report entitled “**STATIC WATER QUALITY MONITORING SYSTEM**” was prepared by **LIEW HAU RUN** 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 : 23/9/ 2021

Signature :  _____

Co-Supervisor : Ir. Danny Ng Wee Kiat

Date : 23/9/2021

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2021, Liew Hau Run. All right reserved.

ABSTRACT

This project discusses about the static water monitoring system using Internet of Things (IoT). As the population in the world increases, the demand for seafood also increases accordingly. Therefore, there is a need to have a low cost, low power consumption and compact water monitoring system that allows aquaculture farmers to evaluate and monitor the quality of the water at all times. The purpose of the water monitoring system is to ensure the quality of the water is kept at an acceptable level so that the aquaculture are not affected. The parameter that will be monitored is temperature of the water. The embedded system used consists of the Arduino platform along with the ThingSpeak cloud server. Arduino Mega is the chosen microcontroller and is used to log data from the temperature sensor, then the data is stored into the SD card and uploaded to the ThingSpeak Cloud server if the connection to data network is established. The Arduino SIM900A GSM GPRS is the communication module used to for data transmission between the Arduino Mega and the ThingSpeak Cloud server. The water parameters are monitored and the data can be shown on the ThingShow mobile application in graphical form. The mobile application can show historical data according to the date and time. The data will be logged at a fixed interval of one hour. The temperature data is collected at two distinct locations where one location is significantly more exposed to sunlight than the other location. Additional sensors can be added in the future to collect more water parameters. The system also consists of a hoist mechanism to lower the sensor into the water when measurement is needed and lift the sensor when the measurement is done. In conclusion, the static water monitoring system can help aquaculture farmers monitor water parameters regularly with less human workers. This will save aquaculture farmers a lot of time and cost. Aquaculture farmers can also find countermeasures to solve the problems that arises before the aquaculture are affected.

TABLE OF CONTENTS

DECLARATION		i
APPROVAL FOR SUBMISSION		ii
ABSTRACT		iv
TABLE OF CONTENTS		v
LIST OF TABLES		vii
LIST OF FIGURES		viii
LIST OF SYMBOLS / ABBREVIATIONS		ix
LIST OF APPENDICES		x
CHAPTER		
1	INTRODUCTION	1
	1.1 Introduction to Static Water Quality Monitoring System	1
	1.2 The Importance of Static Water Quality Monitoring System	2
	1.3 Problem Statement	3
	1.4 Aims and Objectives	4
	1.5 Scope and Limitation of the Study	4
	1.6 Outline of the Report	5
2	LITERATURE REVIEW	6
	2.1 Introduction	6
	2.2 Static Water Quality Monitoring System Architecture	7
	2.3 Water Quality Parameters	9
	2.4 Hardware	12
	2.4.1 Embedded Devices	12
	2.4.2 Data Transmission Protocol	13
	2.4.3 pH Sensor	14
	2.4.4 Temperature Sensor	15
	2.4.5 Dissolved Oxygen (DO) Sensor	16
	2.4.6 Turbidity Sensor	16

3	METHODOLOGY AND WORK PLAN	18
3.1	Project Planning and Milestone	18
3.2	System Architecture Block Diagram	21
3.3	Water Quality Monitoring System Program Flow Chart	22
3.4	Hardware	23
3.4.1	Temperature Sensor	23
3.4.2	Arduino Mega	24
3.4.3	Real-Time Clock (RTC) data logger Shield	24
3.4.4	Micro SD Card adapter	25
3.4.5	SIM900a GSM/GPRS module	25
3.5	ThingSpeakCloud Server	26
3.5.1	ThingShow mobile application	27
3.6	IDE and Debugging Tools	28
3.6.1	Arduino IDE	28
3.6.2	Proteus	29
4	RESULTS AND DISCUSSION	30
4.1	Hardware Design	30
4.1.1	Circuit Diagram and Pin Connection	30
4.1.2	Power System Architecture	32
4.2	Software Development	33
4.2.1	Arduino Mega	33
4.2.2	SIM900a GSM/GPRS module	38
4.2.3	Micro SD card	45
4.3	Project Prototype	46
4.3.1	Electronics Box	46
4.3.2	Power motor coupler and power motor stand	46
4.3.3	Aluminium Beam, Puller and Fishing Line	48
4.3.4	Sensor Base	49
4.3.5	Final Prototype	50
4.4	Features and Performance of Prototype	51
4.4.1	Data Storage	51

4.4.2	Data Monitoring through cloud server and mobile	52
4.5	Problems encountered and Solution	53
4.5.1	Power Supply	53
4.5.2	Power Motor Stand	54
5	CONCLUSION AND RECOMMENDATIONS	56
5.1	Conclusion	56
5.2	Limitation of The Prototype	56
5.3	Recommendations For Future Work	57
	REFERENCES	58
	APPENDICES	60

LIST OF TABLES

Table 2.1:	Advantage(s) and disadvantages different system architecture	8
Table 2.2:	Temperature range for the survival of different species	10
Table 2.3:	DO level with temperature, salinity and altitude change	11
Table 2.4:	Advantages and disadvantages of different embedded systems	13
Table 2.5:	Comparison between different communication protocols	14
Table 2.6:	Advantages and disadvantages of different pH sensors	14
Table 2.7:	Advantages and disadvantages of different temperature sensors	15
Table 2.8:	Advantages and disadvantages of different DO sensors	17
Table 2.9:	Advantages and disadvantages of different turbidity sensors	19
Table 3.1:	Gantt Chart for FYP Part-1	20
Table 3.2:	Gantt Chart for FYP Part-2	20
Table 4.1:	Pin connections of the system	32
Table 4.2:	AT commands for the function Modemsetup()	40
Table 4.3:	AT commands for the function ConnectionServiceProv()	42
Table 4.4:	AT commands for the function Senddata()	46
Table 4.5:	Power consumption of various modules	55

LIST OF FIGURES

Figure 1.1:	Fish utilization and supply of the world, excluding China	2
Figure 2.1:	IoT applications	6
Figure 2.2:	Acceptable pH range of aquatic life	9
Figure 2.3:	Temperature range for the growth of fishes	10
Figure 3.1:	Flowchart of the project approach	18
Figure 3.2:	Block diagram of the water quality monitoring system	21
Figure 3.3:	Main program flow chart	22
Figure 3.4:	DS18B20 Temperature Sensor	23
Figure 3.5:	Arduino Mega	24
Figure 3.6:	DS3231 RTC module	24
Figure 3.7:	Micro SD Card adapter	25
Figure 3.8:	SIM900a GSM/GPRS module	25
Figure 3.9:	ThingSpeak cloud server	26
Figure 3.10:	Screenshot of the ThingShow mobile application	27
Figure 3.11:	Arduino IDE	28
Figure 3.12:	Proteus homepage	29
Figure 4.1:	Schematic diagram of the system architecture	30
Figure 4.2:	Power system architecture	32
Figure 4.3:	Main program flow chart of Arduino Mega	34
Figure 4.4:	Flow chart of rtc_setup()	35
Figure 4.5:	Flow chart of sd_card_setup()	36
Figure 4.6:	Flow chart for Going_to_sleep function	37
Figure 4.7:	Flow chart for the function Modemsetup()	39
Figure 4.8:	Flow chart for the function ConnectionServiceProv()	41
Figure 4.9:	Flow chart for the function CheckError()	42
Figure 4.10:	Flow chart for the function Senddata()	44
Figure 4.11:	Flow chart for the function CheckSD()	45
Figure 4.12:	Electronics box	46
Figure 4.13:	Complete assembly of the power motor, power motor coupler and the power motor stand	46
Figure 4.14:	Top view of power motor coupler	47

Figure 4.15:	Isometric view of power motor stand	47
Figure 4.16:	Side view of power motor stand	48
Figure 4.17:	Lifting and lowering mechanism of the system	48
Figure 4.18:	Top view of sensor base	49
Figure 4.19:	Side view of sensor base	49
Figure 4.20:	Isometric view of final prototype	50
Figure 4.21:	Top view of final prototype	50
Figure 4.22:	Export of data to JSON, XML or CSV file	51
Figure 4.23:	Temperature data on ThingSpeak cloud server	52
Figure 4.24:	Temperature data point	52
Figure 4.25:	Temperature data on ThingShow mobile application	53
Figure 4.26:	Crack marks at the power motor stand	54

LIST OF SYMBOLS / ABBREVIATIONS

°C	degree celsius
A	ampere
ppm	parts per million
%	percentage
V	voltage
m	milli
k	kilo
M	meg
G	giga
Hz	Hertz
mg/L	milligram per liter
s	second
bps	bit per second
c	centi
±	plus-minus
Ω	ohm
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
UART	Universal Asynchronous Receiver-Transmitter
IoT	Internet of Things
PCB	Printed Circuit Board
RTC	Real-Time Clock
SD	Secure Digital
MISO	Master In Slave Out
MOSI	Master Out Slave In
SCK	Serial Clock

CS	Chip Select
SCL	Serial Clock
SDA	Serial Data
SQW	Square Wave
TXD	Transmit Data
RXD	Receive Data
DQ	Data Quality
PWM	Pulse-Width Modulation
DIR	Direction
Li-Ion	Lithium-Ion
AT	Attention
LED	Light-Emitting Diode
mAh	milliamp Hour
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol

LIST OF APPENDICES

APPENDIX A	Datasheets	34
APPENDIX B	Graphs	35
APPENDIX C	Coding	36

CHAPTER 1

INTRODUCTION

1.1 Introduction to Static Water Quality Monitoring System

The earliest form of aquaculture practices includes capturing wild aquatic life in ponds or lakes. This supply of food is readily available. Man has been acquiring natural resources since the Neolithic age, which is around 4000 B.C. in Europe. But in recent decades, the aquaculture namely, the husbandry and farming of aquatic animals and plants has expanded much faster than any other livestock sector. In between the years 1990 and 2009, aquaculture has achieved a 7.5% annual growth rate and has far surpassed the growth rates of the pig and poultry sectors which are less than 2.5% and 5% respectively (Troell, 2014).

Due to the increase in human population over the years, demand for food is also increasing. Thus, there must be a safe and reliable way to evaluate the suitability of a water source for aquaculture. The static water quality monitoring system in this project will help farmers achieve this. Important water parameters such as pH value, temperature, turbidity and Dissolved Oxygen (DO) level will be monitored every hour and uploaded to the cloud server. The water parameters will be monitored 24/7 and the farmers do not need to be physically present to obtain the data. Data can be read through an Android application at any time and anywhere.

The popularity of IoT in recent years has led to the development of different water quality monitoring systems. The working principles are similar whereby, the microcontroller or microprocessor receives signals from the various sensors and data is uploaded to the cloud server via a communication module. In this case, the communication module is a GSM GPRS module. However, the main differences are in the system architecture and the program flow which will be discussed in the literature review.

This study is to develop a low cost, low power consumption and compact static water monitoring system that is able to monitor water parameters and upload the data to the cloud server to be viewed by the aquaculture farmers.

1.2 The Importance of Static Water Quality Monitoring System

Aquaculture is defined as the breeding, raising and harvesting fish, shellfish and aquatic plants. It has a similar concept to agriculture except that it is done with fish and aquatic plants instead of livestock or land-based plants. Aquaculture is also referred to fish farming. Over the years, overfishing is occurring around the globe and it is increasing. Thus, there must an alternate way to feed the world's growing population. According to the United Nations, nearly 90% of the world's marine fish stock is fully exploited, over exploited or depleted, this means that the ocean cannot sustain the demand for seafood and alternative ways to meet this demand must be explored (Kituyi, 2018).

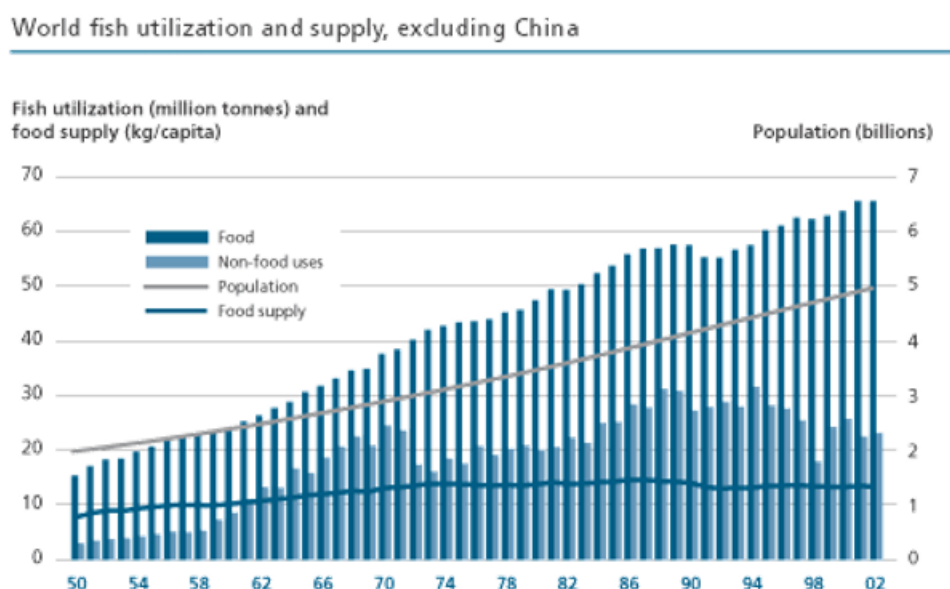


Figure 1.1: Fish utilization and supply of the world, excluding China

Based on Figure 1.1, it is evident that the population of the world is increasing steadily but the food supply is constant. This shows that as the population increases, the demand of fish as a food supply is increasing too. Approximately 30 to 35% of fish populations are fished unsustainably, with an additional 60% fully fished (FAO, 2016). Overfishing has caused many fishes to go extinct or become an endangered species. When certain species of fishes are missing from the ecosystem, a lot of problems may occur as a result. For example, sharks and tunas are susceptible to overfishing and when this happens other species of fishes down the food chain may grow larger and this affects the overall ecosystem. Thus, one way to solve it to increase food supply through aquaculture. In order to achieve this, water monitoring systems are essential to determine if a water source is suitable for aquaculture.

1.3 Problem Statement

When performing aquafarming, one of the major concerns faced by aquaculture farmers is the water quality. To evaluate the water quality, certain water parameters must be taken such as temperature, DO, pH, conductivity, turbidity, etc. The traditional way to evaluate these water parameters is to go to the aquaculture farm physically to check the water quality. This is very labor intensive and time consuming. Not to mention, in between the interval of checking the water quality, the water parameters could have changed and there is no way for the farmers to be notified. Environmental conditions may also affect the water quality parameters. Ensuring the water quality is at its best will reduce the likelihood of diseases that may cause the fishes to die.

Most of the aquaculture monitoring system are not connected to the IoT and some developing countries rely heavily on aquaculture. With the integration of IoT into the water monitoring system, aquaculture farmers can monitor the water quality from a remote area without the need to be physically present at the aquaculture farm. Farmers can also access data from the ThingShow mobile application and data can be shown in graphical format in the mobile application. Data analysis can also be conducted to predict the water quality in the near future. Aquaculture farmers are able to view historical data based on the date and time.

A low cost, low power consumption and compact water quality system can greatly benefit a lot of aquaculture farmers because most aquaculture farmers may not have the financial capability to afford an expensive water monitoring system. A low power consumption water quality monitoring system also means that the water quality can be monitored for a longer period of time before replacing the power supply. One of the benefits to have a compact water monitoring system is the ease of transport. Thus, it is essential to develop a water monitoring system that is cost efficient, consumes less power and compact in size.

1.4 Aims and Objectives

In this project, the aim is to develop a static water quality system that is low cost, consumes less power and compact in size while integrating Internet of Things (IoT) technology into the system. The objectives of the project are:

1. Develop an embedded system architecture that performs hourly monitoring of the water quality parameters
2. Integrating IoT into the water quality monitoring system
3. Link the ThingSpeak cloud server to an ThingShow mobile application that can display water quality parameters in graphical format based on the date and time

1.5 Scope and Limitation of Study

The scope of this project is to:

1. Measure the temperature of the water and upload it to the cloud server
2. Ensure the total amount of power consumption of the system is below 1W
3. Ensure the total cost of the water quality monitoring system is below RM600

Limitation of this project includes:

1. The system is stationary and therefore can only monitor the water parameters in one location at a time
2. Limited to temperature sensor for testing purposes. Can add additional sensors
3. ThingShow mobile application has limited features. Since the project uses a pre-existing android application, additional features cannot be added to the ThingShow mobile application

1.5 Outline of the Report

This project mainly focus on the development of the water quality monitoring system while integrating IoT technology and making sure the water quality system is low cost, consumes less power and more compact. In Chapter 1, it consists of the introduction and importance of the water quality system, problem statement, aims and objectives, scope and limitation of the study and the overview of the project. In Chapter 2, a literature review is done on different researchers about different types of system architectures used and type of water quality parameters to be measured and also the hardware that is to be used in this project. In Chapter 3, the methodology and work plan to develop the prototype is discussed. In Chapter 4, the results of the data that is collected is discussed along with the code flow and the system architecture used. Lastly in Chapter 5, a conclusion of the project, limitation of the prototype and recommendation for future work is discussed.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Internet of Things (IoT) has become very popular in the recent years because of its ability to collect data and send it to the database to undergo data analysis or data prediction. Water monitoring systems are also starting to implement IoT because it is cost effective and can save time.

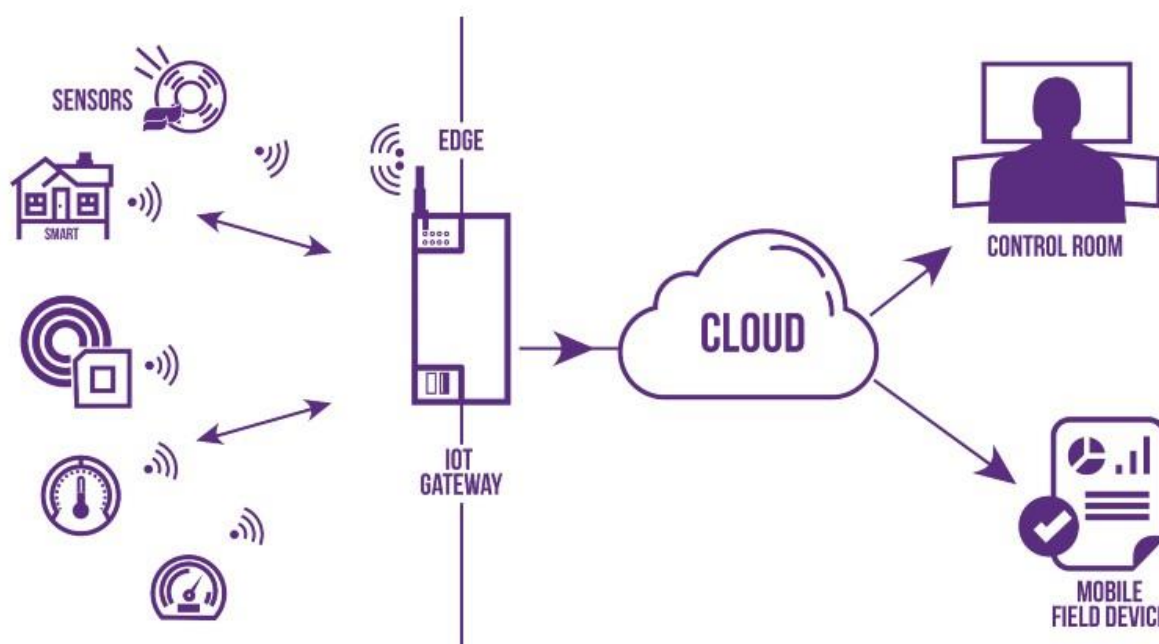


Figure 2.1 IoT applications

IoT based water quality monitoring systems follow a similar working principle. Based on Figure 2.1, the sensors receive data from the surroundings and are sent to IoT gateway which usually consists of a microcontroller and a communication module, then it is uploaded to a cloud server. Users can now access the data from a control room or a mobile device anywhere in the world at any time.

Even though most of the system architecture are similar, the type of sensors, microcontroller, communication module and even cloud server used can play a significant role. For example, for the work of (N.Vijayakumar, 2015), the Raspberry Pi B+ consumes more power than the Arduino Uno used by (Tarik, 2020). Therefore, it is essential to design a system that is low cost and consumes less power.

2.2 Static Water Quality Monitoring System Architecture

In the work of (Alexander T. Demetillo, 2019), the researcher uses Wireless Sensor Network (WSN) which is very suitable for monitoring water parameter in remote area at low cost and reduce the manpower. WSN has many advantages such as its portability and real-time data acquisition and data logging capability. The sensed data from the sensor node is sent to the sink node, then the data is sent to a base station which is uploaded to the cloud server. A GSM module is used as a communication module for long distance communication and the ZigBee communication protocol is used for short distance node to node communication. The water parameters measured are temperature, pH and DO.

Based on the work of (Faustine, A, 2014), the researcher also uses WSN which consists of the WSN sensor node, WSN Gateway Node and WaGoSy database. The WSN sensor node consists of the sensors, microcontroller, Global Positioning System (GPS) receiver and Radop Frequency (RF) transceiver. The water parameters being measured are the pH, electrical conductivity (EC), DO and temperature. The sensed data is then transmitted from the WSN sensor node to the WSN Gateway node. The gateway node collects all the data from different sensor nodes and either stores it in the memory card or sends it to the WaGoSy database for data storage or data analysis. The gateway node consists of General Packet Radio Service (GPRS) module.

As for the work of (Tarik, 2020), the system architecture consists of a microcontroller, sensors, an external Wi-Fi module and a cloud server. The system architecture is fairly simple where the Arduino Uno receives data from the sensors and sends it to the cloud server via the ESP 8266 Wi-Fi module and if the sensor value is beyond a certain threshold a notification is sent to the user's mobile phone. The sensors used are the ultrasonic, pH, turbidity and temperature sensor. The ultrasonic sensor is used to measure the depth of the water. Several types of water conditions are also investigated in this study which includes river, tap, pond and lake water.

For the work of (N.Vijayakumar, 2015), the researcher uses Raspberry Pi B+ as the main controller. Sensors include temperature, conductivity, turbidity, DO and pH sensor. The data from the sensors are transmitted to the Raspberry Pi B+ and sent to either the cloud server or the mobile device via an IoT module (USR-WIFI232-X-V4.4). The communication protocol used between Raspberry Pi B+ and USR-WIFI232 is UART.

Table 2.1: Advantages and disadvantages of different system architectures

	Advantages	Disadvantages
Faustine, A, 2014	<ul style="list-style-type: none"> • Sensor node is scalable (more sensors can be added) • Provides real-time notification 	<ul style="list-style-type: none"> • High power consumption because sensor and gateway node require power supply
N.Vijayakumar, 2015	<ul style="list-style-type: none"> • Simple to implement 	<ul style="list-style-type: none"> • High cost • High power consumption due to using Raspberry Pi and Wi-Fi module • No real time notification capability
Alexander T. Demetillo, 2019	<ul style="list-style-type: none"> • Low cost • Can be implemented in remote areas 	<ul style="list-style-type: none"> • More complicated to implement as there are several nodes • No real time notification capability
Tarik, 2020	<ul style="list-style-type: none"> • Low cost • Simple to implement • Has real-time notification through SMS • 	<ul style="list-style-type: none"> • Not suitable to be used in remote areas • High power consumption because Wi-Fi is used to upload data • Data may be lost when connection is unstable

2.3 Water Quality Parameters

For fishes to live in a healthy environment and do not develop diseases, the water quality is very important. Certain water parameters can help determine the water quality which includes pH value, temperature, dissolved oxygen level, turbidity, conductivity, ammonia percentage in water, etc. By monitoring these water parameters and fixing the problem, the health of the fishes can be checked. The health of the fishes is also very important as human consumes the flesh of the fishes and could pose potential threat to the health of the human.

One of the water parameters discussed in this project is the pH value of the water. The pH stands for 'power of hydrogen' and the value is determined by the molar concentration of hydrogen ions (H^+)³. The higher the concentration of the H^+ , the lower the pH (more acidic) and the higher the OH^- concentration, the higher the pH (more alkaline). The importance of the pH value is that if the pH of the water is too low or too high, the aquatic organisms will die as the water is not a suitable environment to live in. Most aquatic animals prefer a pH range of 6.5 to 9.0 (Fondriest Environmental Inc, 2013).

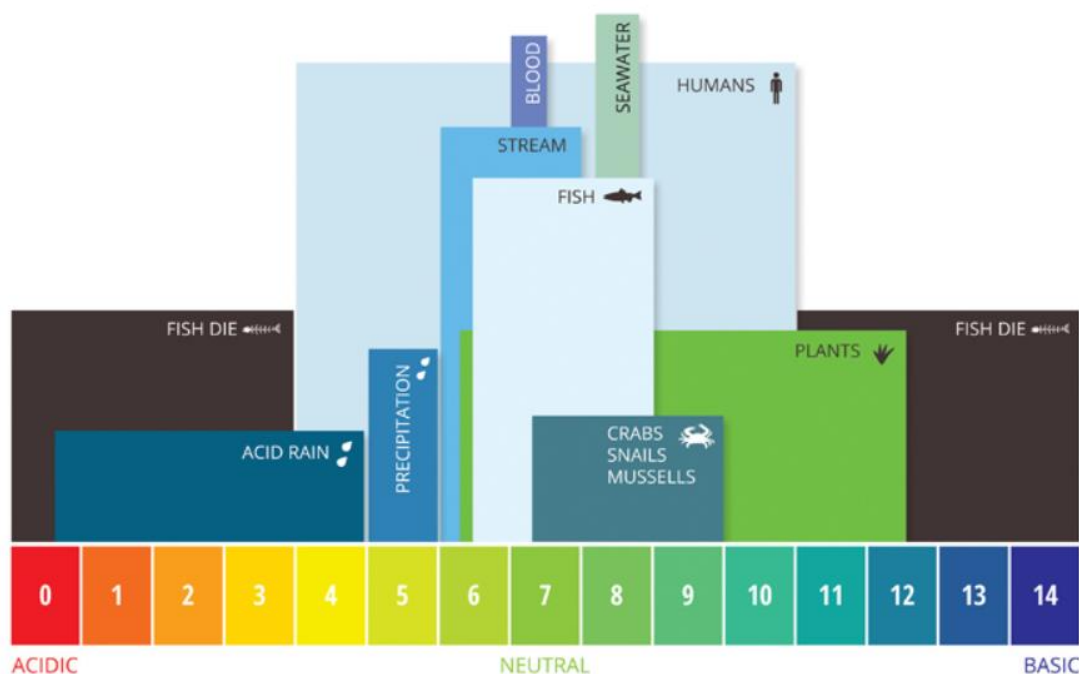


Figure 2.2 Acceptable pH range of aquatic life

As pH levels increases or decreases outside of the range of acceptable pH value, the aquatic animals get stressed and reduce breeding and their survival rate decreases. Therefore, it is essential to measure the pH value and monitor the value at all times.

Another water parameter to be measured is the temperature of the water. Temperature is an important factor in the growth and survival of aquatic organisms. The aquatic organisms are classified into 3 types which are coldwater, warmwater and tropical species.

Table 2.2 Temperature range for the survival of different species

Species	Temperature Range/°C
Coldwater	+20 °C to +25 °C
Warmwater	+25 °C to +27 °C
Tropical	+25 °C to +28 °C

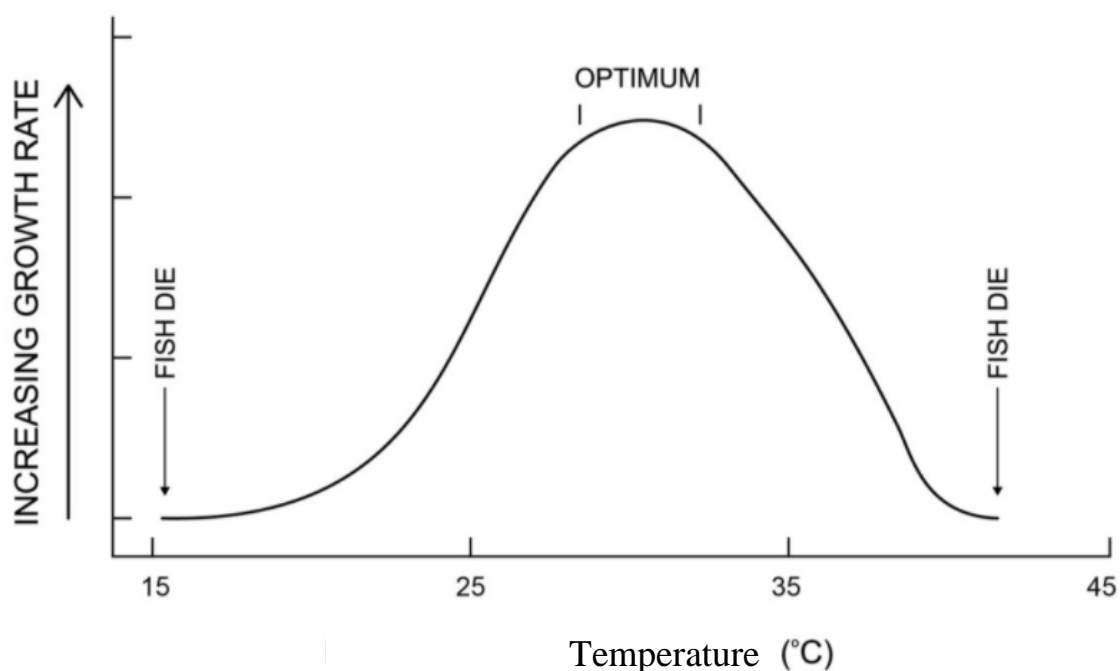


Figure 2.3 Temperature range for the growth of fishes

Based on the Figure 2.3, the optimum temperature range for fish growth is in between +27 °C to +30 °C. If the temperature falls below or is above this range, the fishes will begin to get stressed and might die if the temperature is too high or low (Claude E. Boyd, 2018). Therefore, it is essential to maintain the temperature of the water at an acceptable range.

Next, the dissolved oxygen (DO) level is also discussed in this project. DO is the most important chemical parameter in aquaculture. Fishes like humans require oxygen for respiratory functions. Low-DO levels can cause the death of fishes. The DO level is also a

function of temperature which means that if the temperature increases, the amount of DO in the water decreases (Towers, 2015).

Table 2.3 DO level with temperature, salinity and altitude change

Variable	Temperature (F)				
	68.0	71.6	78.8	82.4	86.0
Salinity (ppm)					
0	9.2	8.8	8.2	7.9	7.6
5,000	8.7	8.4	7.8	7.5	7.3
10,000	8.3	8.0	7.4	7.1	6.9
Altitude (ft)					
0 (Sea Level)	9.2	8.8	8.2	7.9	7.6
1,000	8.8	8.5	7.9	7.6	7.4
2,000	8.5	8.2	7.6	7.3	7.1

Based on Table 2.3, as the temperature increases, the DO levels in ppm decreases. Increase in salinity and altitude also causes the DO levels to decrease. As a general rule, it is best to maintain DO levels at saturation or at least 5ppm.

The last water parameter to be discussed is turbidity of the water. Turbidity is the measure of the amount of suspended particles that exists in the water. The more suspended particles that exist in the water, the murkier the water seems and thus higher the turbidity. High turbidity means that sunlight cannot reach the aquatic plants more easily, this will cause the plants to not undergo photosynthesis and may die as a result. Turbidity levels of more than 100,000 NTU will also kill the fishes (Rowe D.K. 2002). Therefore, turbidity levels must be kept at an acceptable level.

2.4 Hardware

2.4.1 Embedded Devices

According to the work of (Tarik, 2020), Arduino Uno is a microcontroller board built on ATmega328P. It has 14 digital I/O pins (6 can be used for Pulse-Width Modulation), 6 analog input pins, 16 MHz crystal oscillator, USB connection, a power jack, a reset button and an In Circuit Serial Programming (ICSP) header. The Arduino architecture consists of two memories which are the program memory and data memory. The ATmega328 chip has 32 kb of flash memory, 2kb of SRAM, 1kb of EEPROM and operates with a clock speed of 16MHz. The reason Arduino Uno is chosen is because of its low cost and low power consumption. Arduino is also an open-source prototyping platform that has a lot of libraries and examples online, this makes the development of the embedded system faster and easier.

As for the work of (N. Vijayakumar, 2015), Raspberry Pi 3 is used as the core controller. Raspberry Pi 3 has huge processing power and many interfaces such as HDMI, multiple USB, Ethernet, onboard Wi-Fi, Bluetooth 4.1 Low Energy (BLE) and 40-PIN GPIO connectors. Raspberry Pi 3 comes with a range of drivers for interfacing but as the system boots up each driver is loaded and this increases the boot time and wastes considerable amount of resources for redundant processes. However, the power consumption of the Raspberry Pi 3 is also significantly higher than Arduino Uno.

In the work of (Faustine, A, 2014), the microcontroller used is the Arduino Mega 2560. Arduino Mega 2560 has 54 Digital I/O pins (15 of them have PWM) and 16 input analog pins. Arduino Mega 2560 also has 256 kb of flash memory, 8kb of SRAM, 4kb of EEPROM which is much more than Arduino Mega. (Faustine, A, 2014) also uses the LoRa communication module, LoRa devices can transmit data up to a long range which consuming very little power and it uses the LoRaWAN communication protocol which is different from other communication protocols such as Wi-Fi and Zigbee. But due to its larger size (4" x 2.1") compared to Arduino Mega (2.7" x 2.1"), Arduino Mega is more favourable. Arduino Mega 2560 also consumes more power than Arduino Mega.

Lastly in the work of (S. Geetha, 2017), Texas Instruments CC3200 is chosen as the main controller. TI CC3200 has built in Wi-Fi module with dedicated ARM MCU for wireless communication purposes. This means that the speed of operation is faster as compared to controllers with external Wi-Fi modules. But because of the high cost of the TI CC3200, it is not a viable option for this project.

Table 2.4: Advantages and disadvantages of different embedded systems

	Advantages	Disadvantages
Arduino Uno	<ul style="list-style-type: none"> • Low power consumption • Low Cost • Open-source prototyping platform 	<ul style="list-style-type: none"> • Require external communication module
Raspberry Pi 3	<ul style="list-style-type: none"> • 802.11n Wireless LAN • Bluetooth 4.1 Low Energy (BLE) • 40-PIN GPIO connectors for interfacing with external sensors or modules 	<ul style="list-style-type: none"> • Requires SD card for internal storage • High power consumption
Arduino Mega 2560 with LoRa	<ul style="list-style-type: none"> • Low power consumption with fast start-up • Open-source prototyping platform • Long range communication for remote areas 	<ul style="list-style-type: none"> • Require external communication module • Expensive
TI CC3200	<ul style="list-style-type: none"> • Built in Wi-Fi module • Low power consumption 	<ul style="list-style-type: none"> • Expensive

2.4.2 Data Transmission Protocol

According to the work of (Vinod and Sushama, 2016), the ZigBee communications protocol is used. ZigBee is a technology for data transfer in wireless network. It is designed for multi-channel control systems and alarm system which suitable for IoT applications. ZigBee is built based on the IEEE standard 802.15.4 for low-rate WPANs and its power consumption is low compared to Wi-Fi.

In the work of (Peng et al., 2009), communication between the controller and the cloud server for data storage is done using GSM/GPRS. This method is suitable for this application because of its long range capabilities and also its low power consumption when in sleep mode. Though there are some drawbacks which are these systems require additional SIM card and larger quantities of data storage and retrieval is not possible. But because GSM/GPRS meets the requirements of low cost and low power consumption, it is chosen for this project.

Next, (Tarik, 2020) uses the external Wi-Fi module ESP8266 to communicate between the microcontroller and the cloud server. The advantages of Wi-Fi is that the speed of data transmission is very fast as compared to ZigBee and GSM/GPRS but one of the major concerns is Wi-Fi consumes a lot of power. Wi-Fi is also not readily accessible in remote and rural areas and the transmission range of Wi-Fi is short as compared to GSM/GPRS or LoRa.

Table 2.5: Comparison between different communication protocols

	ZigBee	GSM/GPRS	Wi-Fi
IEEE Standard	IEEE 802.15.4	IEEE 802.21	IEEE 802.11n
Operating Frequency	784/ 868/ 915MHz, 2.4GHz	900MHz, 1.8GHz	2.4GHz, 5GHz
Transfer Speed	250kps	42.8kbps	150Mbps, 450Mbps
Power Consumption	Low	Low	High
Applications	Medium range IoT applications	Remote monitoring, data connectivity, etc	Video-based apps

2.4.3 pH sensor

As for the pH sensor, there are several models from different manufacturers that are being discussed in this project. OMRON PH-1's outer sheath is made out of PVC and is therefore water proof and this is an important feature for water quality monitoring systems. Its operating temperature range is $-10\text{ }^{\circ}\text{C}$ to $+60\text{ }^{\circ}\text{C}$ which is suitable for most climates. Another advantage is the low cost of the sensor and the sensor is also commercially available in Malaysia.

According to the research done by (Hall, 2007), GLI PHD pH sensor has an operating temperature of $-5\text{ }^{\circ}\text{C}$ to $+95\text{ }^{\circ}\text{C}$, this means that the sensor is not suitable for very low temperatures. The sensitivity of the sensor is less than 0.005 pH. Though GLI PHD pH sensor is very sensitive and reliable, its disadvantage is that it is very expensive.

As for the work of (Xylem Inc, 2015), WQ 201 from Global Water Instrumentation is used, its operating temperature is from $-5\text{ }^{\circ}\text{C}$ to $+55\text{ }^{\circ}\text{C}$ which is a smaller range than both the OMRON PH-1 and GLI PHD. One of the advantages of WQ 201 is that it can operate under pressure of 40 psi and has an accuracy of 2% full scale. But the water quality monitoring system generally does not operate under high pressure and WQ 201 is very costly.

Lastly, (Alexander T. Demetillo, 2019) used pH sensor manufactured by Atlas Scientific. Sensors produced by Atlas Scientific are of high quality and are also compatible with Arduino, its accuracy is $\pm 0.02\%$ and Atlas Scientific also provides calibration solutions. Atlas Scientific pH sensor has a lot of advantages but it is the most expensive out of all the other 3 options.

Table 2.6: Advantages and disadvantages of different pH sensors

	Advantages	Disadvantages
OMRON PH-1	<ul style="list-style-type: none"> • Low Cost • Waterproof 	<ul style="list-style-type: none"> • Small operating temperature range
GLI PHD	<ul style="list-style-type: none"> • Reliable • Less regular maintenance 	<ul style="list-style-type: none"> • Expensive
WQ201	<ul style="list-style-type: none"> • Waterproof • Reliable 	<ul style="list-style-type: none"> • Expensive
Atlas Scientific	<ul style="list-style-type: none"> • Compatible with Arduino • Accuracy of $\pm 0.02\%$ • Fast response time 	<ul style="list-style-type: none"> • Expensive

2.4.4 Temperature sensor

Several temperature sensors were investigated and compared, (M Cho Zin, 2019) uses the digital thermometer DS18B20, there are several advantages by using this sensor. The advantages include that it is low cost, has an accuracy of ± 0.5 °C and is waterproof. But there is drawback whereby accuracy decreases when the temperature range increases which happened when extreme climate changes occur.

As for (Xylem Inc, 2015), WQ101 is used as the temperature sensor. This temperature sensor has an accuracy of ± 0.1 °C which is more accurate than DS1820. The housing is also stainless steel which means rust is less likely to form on the housing. WQ 101 is also waterproof but it is very expensive.

Next, (Kofi, 2020) uses a temperature model Pt-1000, this temperature sensor has range of 0 °C to +100 °C which is suitable for most water quality monitoring applications except for low temperature measurements. The sensor has an accuracy of ± 0.1 °C, it is suitable for long term usage as it has low distortion but is expensive to purchase.

Table 2.7: Advantages and disadvantages of different temperature sensors

	Advantages	Disadvantages
Digital Thermometer DS18B20	<ul style="list-style-type: none"> • Low Cost • Widely Available • Accuracy of ± 0.5 °C • Waterproof 	<ul style="list-style-type: none"> • Accuracy changes to ± 2 °C when in the temperature range increases
WQ101	<ul style="list-style-type: none"> • Waterproof • Reliability 	<ul style="list-style-type: none"> • Expensive

Pt-1000	<ul style="list-style-type: none"> • Reliable due to low distortion • Suitable for long term usage 	<ul style="list-style-type: none"> • Expensive
---------	--	---

2.4.5 Dissolved Oxygen (DO) sensor

As discussed in the work of (Xylem Inc, 2015), the DO sensor used is of the model WQ 401, the measurement range is from 0 to 8 ppm. The accuracy of the sensor is $\pm 0.1\%$ full scale and the operating temperature range is $-40\text{ }^{\circ}\text{C}$ to $+55\text{ }^{\circ}\text{C}$ which is suitable for monitoring the aquaculture. It is low cost and easy to maintain.

As for the (Alexander T. Demetillo, 2019), most of the sensors used are from Atlas Scientific. Atlas Scientific is an American company known for producing reliable but costly high quality sensors. Atlas Scientific also provides calibration solution for the user to check if the DO sensor is working properly.

Table 2.8: Advantages and disadvantages of different DO sensors

	Advantages	Disadvantages
WQ401	<ul style="list-style-type: none"> • Low cost • Ease of maintenance • Reliability 	<ul style="list-style-type: none"> • Small operating temperature range
Atlas Scientific	<ul style="list-style-type: none"> • Compatible with Arduino 	<ul style="list-style-type: none"> • Expensive

2.4.6 Turbidity sensor

The sensor used in the work of (M Cho Zin, 2019) is SKU: SEN0189 turbidity sensor. The operating temperature of the sensor is $5\text{ }^{\circ}\text{C}$ to $+90\text{ }^{\circ}\text{C}$. The analog output of the sensor is from 0 - 4.5V and the digital output has a high or low level. The levels can be adjusted by a potentiometer. It is low cost but is not waterproof.

According to the research conducted by (Hall, 2007), Hach 1720D is the turbidity sensor used. It has an accuracy of $\pm 2\%$ and has a resolution of 0.001 NTU, this sensor is very suitable for measuring the turbidity of very murky water because of its high resolution. But Hach 1720D is costly.

Next, WQ770 is also compared in the study by (M Cho Zin, 2019). The range of turbidity to be measured is from 0–50 NTU to 0–1000 NTU. The accuracy is $\pm 1\%$ full scale and the operating temperature is $0\text{ }^{\circ}\text{C}$ to $+50\text{ }^{\circ}\text{C}$ which is smaller range compared to SKU:

SEN0189. Based on its datasheet, the turbidity sensor's calibration interval is between 6 to 12 months which means there is no need for frequent calibrations. It is however very expensive.

Lastly, Confab Model 851 is another turbidity sensor that is discussed in this project. The range of the sensor is from 0–2 NTU to 0–2000 NTU and has a resolution of 0.001 NTU. The sensor requires very low maintenance and has a life expectancy of 13 years. It is also waterproof and dust proof but is very expensive.

Table 2.9: Advantages and disadvantages of different turbidity sensors

	Advantages	Disadvantages
SKU: SEN0189	<ul style="list-style-type: none"> • Low cost 	<ul style="list-style-type: none"> • Not waterproof
Hach 1720D	<ul style="list-style-type: none"> • Accuracy of $\pm 2\%$ • Resolution of 0.001 NTU 	<ul style="list-style-type: none"> • Expensive
WQ770	<ul style="list-style-type: none"> • Waterproof • Reliability due to low interference • Long term stability 	<ul style="list-style-type: none"> • Expensive
Confab Model 851	<ul style="list-style-type: none"> • Fast response time • Resolution of 0.001 NTU • Long term stability 	<ul style="list-style-type: none"> • Expensive

CHAPTER 3

METHODOLOGY AND WORKPLAN

3.1 Project Planning and Milestone

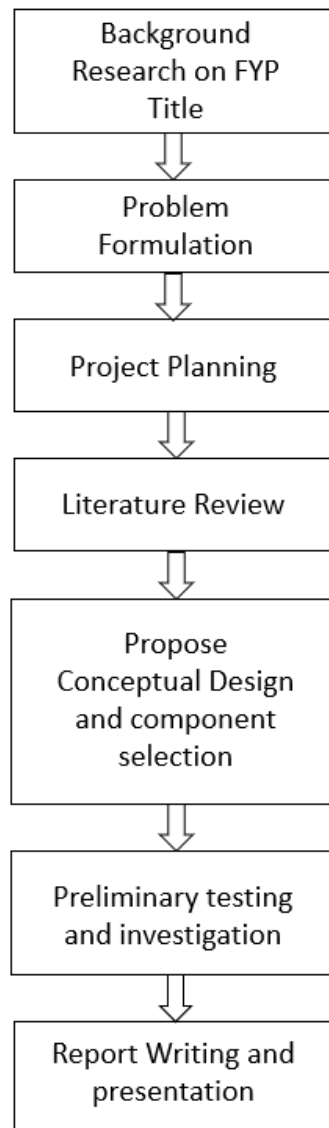


Figure 3.1 Flowchart of the project approach

The project approach taken to finish the final year project is shown in Figure 3.1. Firstly, background research on different final year project titles is done to have a better understanding of the project and the responsibilities that entails. Next, problem

formulation is done, along with setting the objectives and scope of study. After that, a comprehensive literature review is conducted to compare the different methods used by various researchers. The criteria for choosing the method to be used for this project includes cost, power consumption, reliability, etc.

After the literature review is conducted, the conceptual design is proposed. This is the stage where the type of hardware, cloud server platform, GUI, Integrated Development Environment (IDE) and software for debugging is considered. Component selection occurs when the type of hardware is finalised after considering different types of alternatives.

The software used to design the circuit is Proteus. Proteus is chosen because it can be used to simulate the circuit before constructing it. Proteus can also be used to run codes and help with the debugging process. Simulation is conducted to ensure all bugs are fixed. Lastly, the code is uploaded to the embedded system and is debugged subsequently through Arduino IDE. The prototype is then deployed for testing. Problems encountered are recorded and fixed.

Table 3.1 : Gantt Chart for 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	Literature Review														
M3	Propose conceptual design and component selection														
M4	Preliminary testing and investigation														
M5	Report writing and presentation														

Table 3.2: Gantt Chart for FYP Part 2

No	Project Activities	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14
M1	Circuit design using Proteus														
M2	Testing of prototype														
M3	Data gathering														
M4	Report writing														

3.2 System Architecture Block diagram

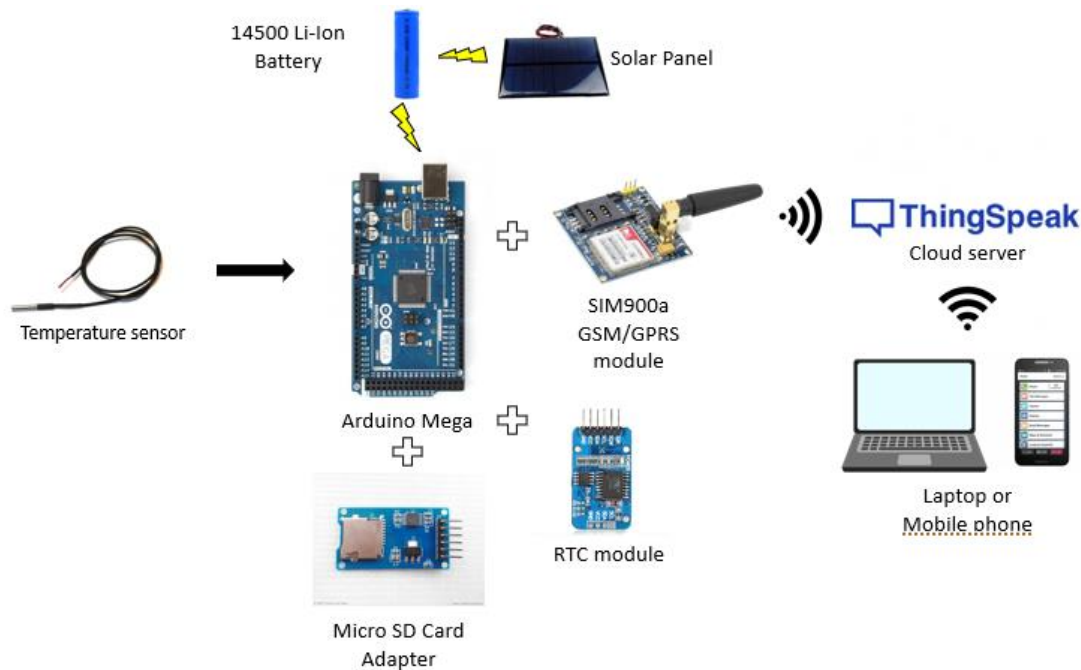


Figure 3.2 Block diagram of the water quality monitoring system

Based on Figure 3.2, the microcontroller used is the Arduino Mega which is powered by a 3.7V 14500 Lithium-Ion Battery. This Lithium-Ion Battery is charged using solar power through the 5V solar panel and the 3.7V is boosted to 5V to power the Arduino Mega through a voltage booster. The sensor that is used to measure the water parameter is the temperature sensor. A Real-Time Clock RTC module is connected to the Arduino Mega and a micro-SD Card adapter is also connected to the Arduino Mega. The main purpose of using the RTC module is to log data at the accurate time and date. The micro-SD Card adapter is used to insert a micro-SD card to store the information in case the SIM900a GPRS module is not able to send data to the cloud server. The communication protocol used between the Arduino Mega and the SIM900a GSM/GPRS module is UART. The Arduino Mega has an onboard voltage regulator that can regulate the voltage supply to 5V and the Arduino Mega is used to supply voltage to the SIM900a GSM/GPRS module. The cloud server platform used is ThingSpeak. Data is sent to the ThingSpeak server via the SIM900a GPRS module and can be read through laptop or mobile devices. For mobile devices, by downloading the mobile application, ThingShow sensor data can be read based on the time and date.

3.3 Water quality monitoring system main program flow chart

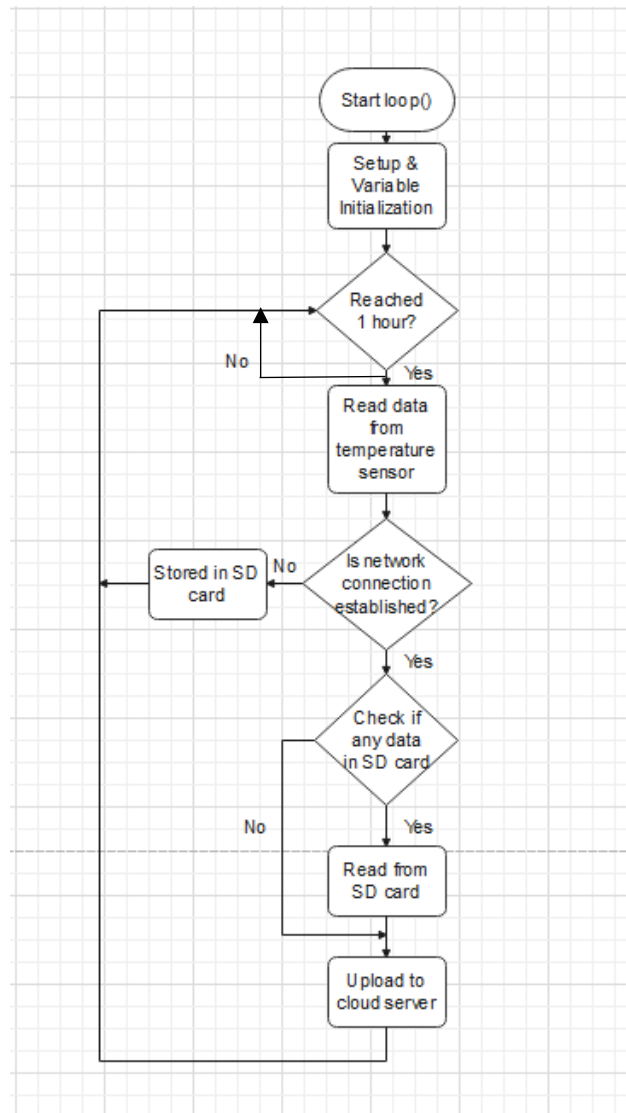


Figure 3.3 Main program flow chart

Figure 3.3 shows the main program flow chart where the program is in a loop. After the microcontroller is powered up, the SIM900a GSM/GPRS module, RTC module, SD card adapter and sensors begins setting up. The program begins by declaring variables. Next, the microcontroller checks the RTC to see if the 1 hour interval is reached. If the 1 hour interval is reached, sensor data will be read from the temperature sensor. Next, the connection of the SIM900a GSM/GPRS module to the network provider is checked. If there is no connection, the sensor data is stored in the SD card and the microcontroller

waits for the next 1 hour interval before repeating the process again. If there is a connection to the network provider, the SD card is then checked to see if there is any stored data. If there is data stored in the SD card, the SD card is read and the data is sent to the cloud server. If there is no data stored in the SD card, the current logged data is uploaded to the cloud server. After the data was uploaded to the cloud server, the program is repeated again by checking if the RTC has reached the 1 hour interval.

3.4 Hardware

3.4.1 Temperature sensor

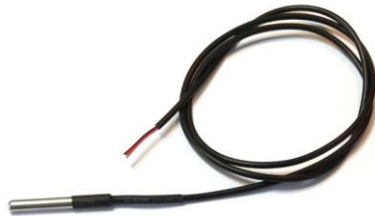


Figure 3.4: DS18B20 Temperature Sensor

The temperature sensor used is the DS18B20. The DS18B20 is waterproof and is submersible under water which is important when used in the water quality monitoring system. The temperature range is from -55°C to 125°C with an accuracy of $\pm 0.5^{\circ}\text{C}$. The accuracy is acceptable since a change of $\pm 0.5^{\circ}\text{C}$ to the temperature will not affect the aquaculture drastically.

3.4.2 Arduino Mega



Figure 3.5: Arduino Mega

Arduino Mega is a microcontroller board based on the ATmega2560 and has 54 digital I/O pins (15 of which has PWM) and 16 analog input pins. The Arduino Mega also has 8kb of SRAM which is very useful in compiling codes. The size and weight of the Arduino Mega is 101.52 x 53.3 mm and 37 g respectively which is very small and compact. The Arduino Mega microcontroller also has sleep mode functions which can reduce the power consumption when not in use.

3.4.3 Real-Time Clock (RTC) module



Figure 3.6: DS3231 RTC module

The DS3231 RTC module can be connected to Arduino Mega via the digital pins and the purpose of the RTC data logger is to keep track of the time and date. When 1 hour is passed, the Arduino Mega will begin to read the sensor data and choose to either store the data in the SD card or upload it to the cloud server. The 14 digital and 6 analog pins can still be used to connect to other devices.

3.4.4 Micro SD Card adapter



Figure 3.7: Micro SD Card adapter

The micro SD adapter has an onboard 3.3 voltage regulator which is compatible with the Arduino Mega. A micro SD card can be inserted into the micro SD adapter and the SD card can be used to store logged data if there is no internet connection. The communication protocol used is the standard SPI protocol.

3.4.5 SIM900a GSM/GPRS module



Figure 3.8: SIM900a GSM/GPRS module

GSM stands for Global System for Mobile Communications and it is created to describe the protocols for 2G digital cellular networks. The SIM900a supports a single SIM card and is able to send and receive messages from another mobile phone. When a SIM card is inserted, the SIM900a can notify the farmer that a water parameter is out of range via a Short Message Service (SMS). The SIM900a has a maximum upload speed of 42.8KBps. The SIM900a uses UART communication protocol to communicate with the Arduino Mega and serves as a communication module to send data to the cloud server.

3.5 ThingSpeak Cloud Server

Created: [5 days ago](#)
 Last entry: [a day ago](#)
 Entries: 28

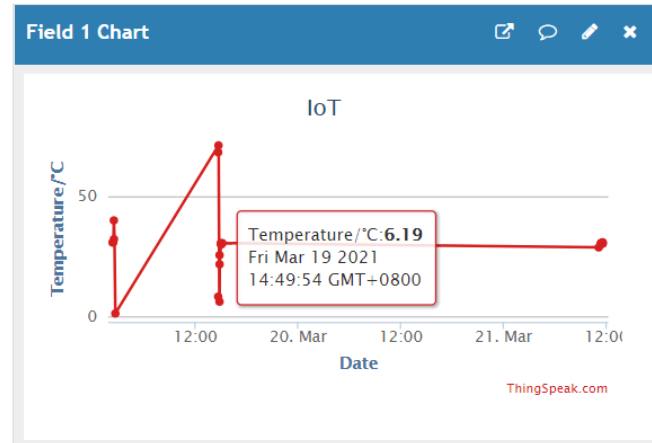


Figure 3.9: ThingSpeak cloud server

The cloud server used is ThingSpeak, ThingSpeak is used in many IoT applications because of its real-time database and data analysis through MATLAB. Figure 3.12 shows the temperature vs time graph and each point can show the exact temperature, time and date of the data. Furthermore, data can be exported in JSON, XML or CSV format. The reason ThingSpeak cloud server is chosen is because a mobile application called ThingShow is linked to the ThingSpeak cloud server. Therefore, users that has downloaded the mobile application is able to view the data on their mobile phones at any time provided there is internet connection.

3.5.1 ThingShow mobile application

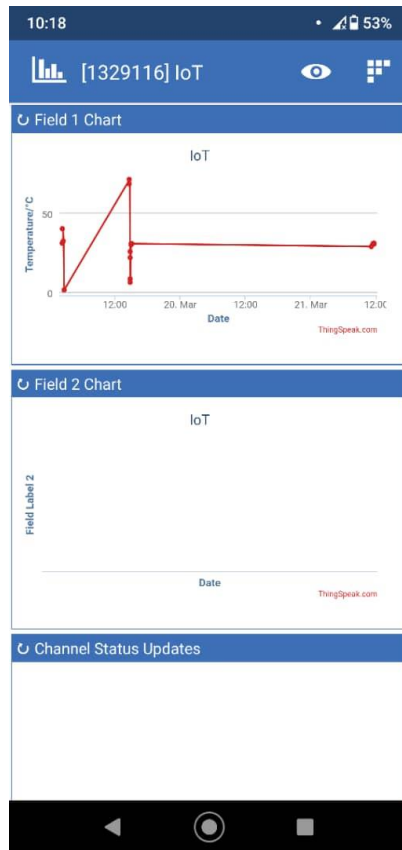


Figure 3.10: Screenshot of the ThingShow mobile application

Figure 3.13 shows the screenshot of the ThingShow mobile application where the field 1 chart shows the temperature vs time graph. ThingShow is chosen because it is linked to the ThingSpeak cloud server. The user can insert the channel ID for example, 1329116 and the user is able to monitor the data on their mobile phone. Multiple fields can also be added for the turbidity, pH and DO sensor data.

3.6 IDE and Debugging Tool

3.6.1 Arduino IDE

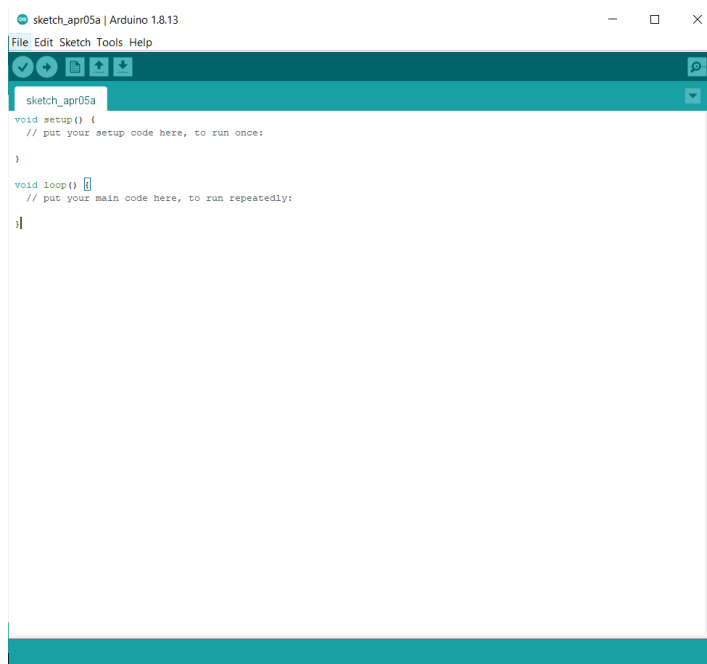


Figure 3.11: Arduino IDE

Figure 3.14 shows the Arduino IDE used to debug and upload codes to the Arduino Mega. Arduino IDE is a cross-platform application which means it is able to operate on Windows, macOS and Linux operating systems. Arduino is an open-source prototyping platform and there is a lot of tutorials and guides online that can help save time in the development of the prototype.

3.6.2 Proteus

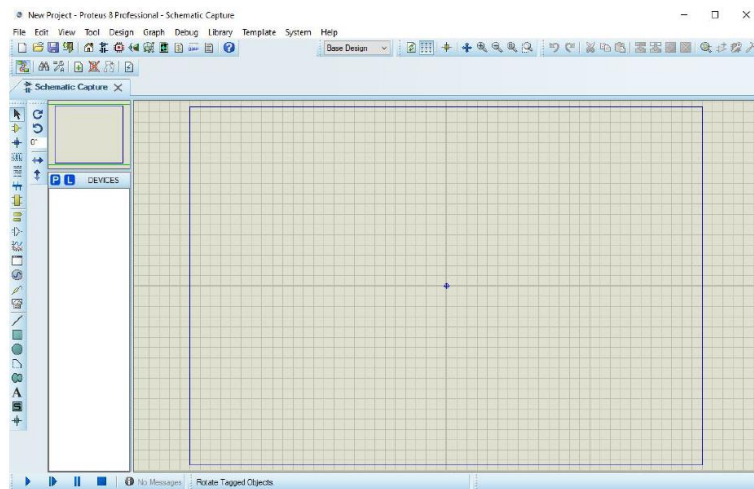


Figure 3.12 Proteus homepage

Proteus is used for circuit design with the relevant hardware. After the circuit design, simulation is conducted to ensure there is no bugs in the system. The circuit design can then be used for the manufacturing of the printed circuit board (PCB). Proteus is also has a lot of debugging tools which are very useful in saving time during the debugging process.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Hardware Design

4.1.1 Circuit Diagram and Pin Connection

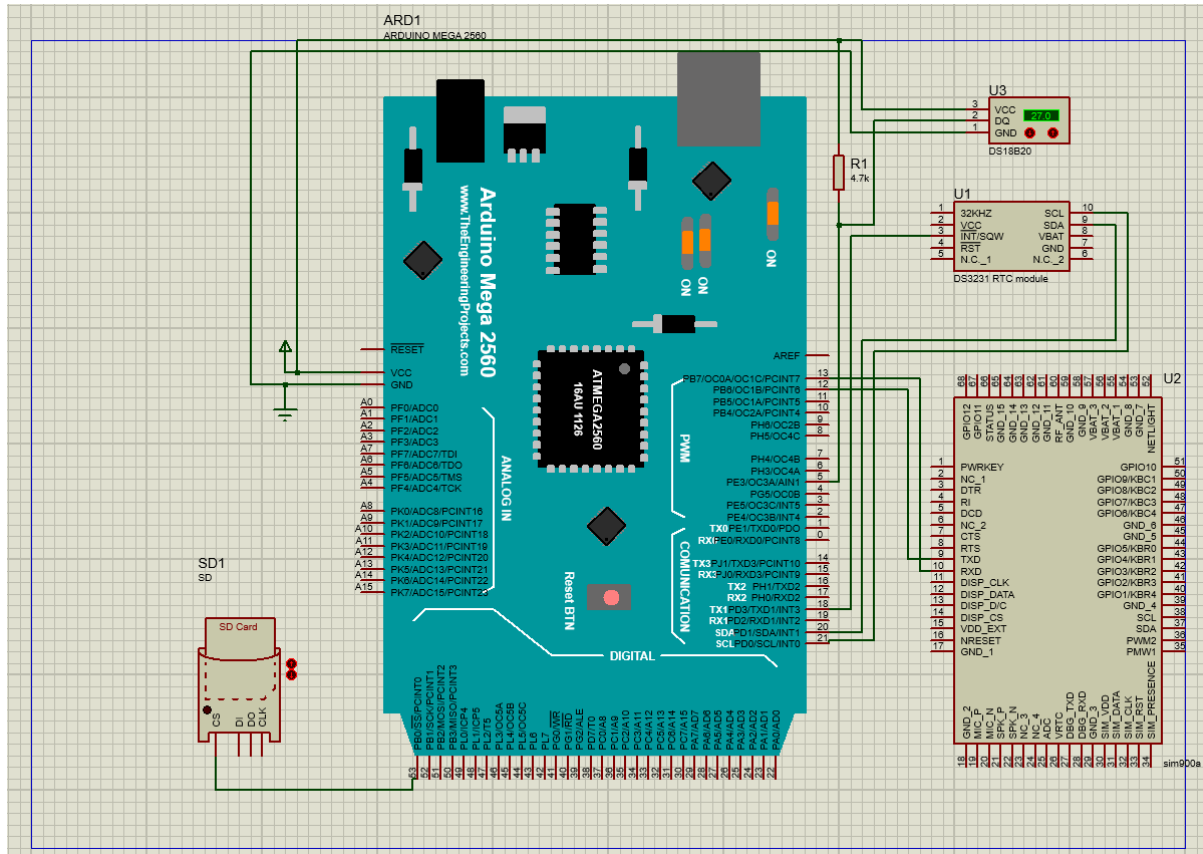


Figure 4.1 Schematic diagram of the system architecture

Figure 4.1 shows the schematic diagram of the system architecture. Arduino Mega acts as the microcontroller and other modules such as the micro SD card module, DS3231 RTC module, SIM900a GPRS/GSM module and DS18B20 temperature sensor are connected to it.

An important to note is there is a $4.7k\Omega$ resistor added between the data line and the Vcc line of the DS18B20 temperature sensor. This is mainly for the calibration of the temperature sensor. The SDA and SCL which are the data and clock lines of the DS3231 RTC module are connected to pin 20 and 21 which are pins for the data and clock lines of the Arduino Mega. The SQW pin is connected to pin 18 of the Arduino Mega which acts as a transmit pin. The SIM900a GPRS/GSM module communicates with the Arduino Mega via UART and the transmit and receive pins of the module is connected to pins 12

and 13 of Arduino Mega. This is to ensure the SIM900a GPRS/GSM module is able to transmit and receive data from the Arduino Mega.

Lastly, the chip select pin of the micro SD card adapter is connected to pin 53. The module used in practice could not be found in the Proteus library, so MISO, MOSI and SCK pins are not shown in the schematic diagram. The DC motor driver with PWM and DIR pins is also not available in the Proteus library but the pin connections are shown in the table below.

Table 4.1: Pin connections of the system

Module	Module Pins	Arduino Mega Pins
Micro SD card adapter	MISO	50
	MOSI	51
	SCK	52
	CS	53
DS3231 RTC module	SCL	21
	SDA	20
	SQW	18
SIM900a GRPS/GSM module	TXD	12
	RXD	13
DS18B20 temperature sensor	DQ	5
Motor driver	PWM	3
	DIR	2

4.1.2 Power System Architecture

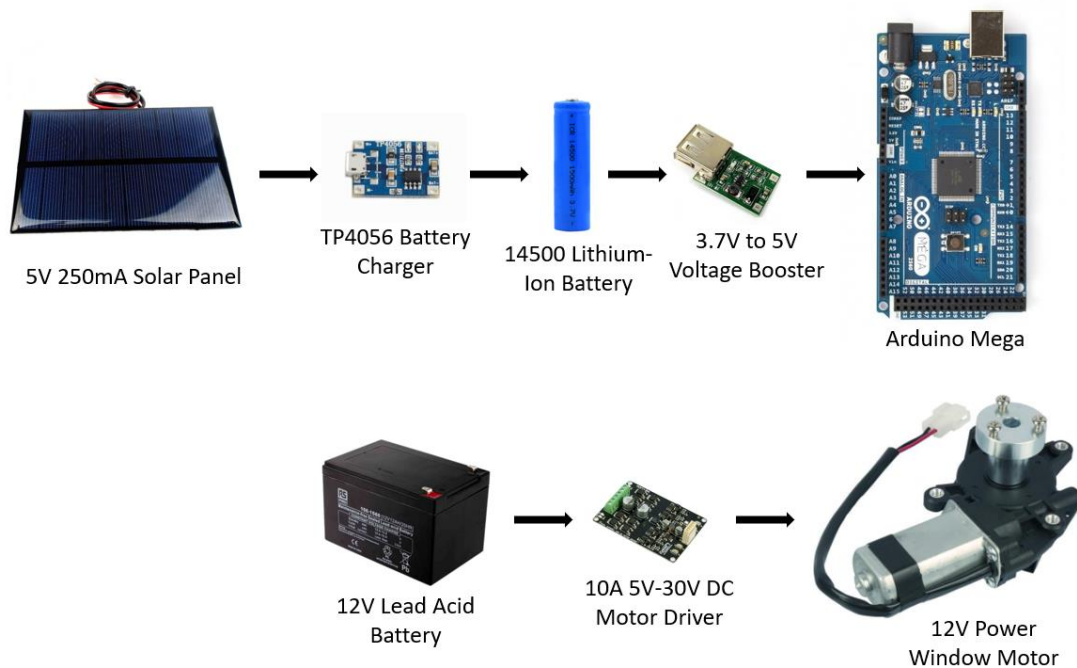


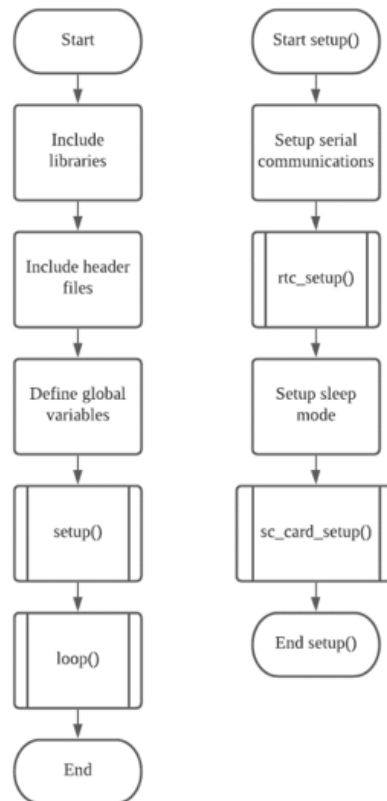
Figure 4.2: Power system architecture

Figure 4.2 shows the power system architecture of the static water quality monitoring system. The 5V solar panel is used to power up the TP4056 battery charger which charges the 14500 Lithium-Ion Battery. The 14500 Lithium-Ion Battery has a voltage rating of 3.7V so it is placed in a battery holder which is connected to a voltage booster. This voltage booster then boosts the 3.7V to 5V to power the Arduino Mega.

Next, the 12V Lead Acid Battery is used to power the 5V-30V DC motor driver. The motor driver is used to control the Pulse-Width-Modulation (PWM) and direction (DIR) of the power window motor. The 12V Lead Acid Battery is not sustainable. Thus, a 12V solar panel can be added to improve the sustainability of the system. This is discussed in Chapter 5 under subsection 5.3 Recommendation For Future Work.

4.2 Software Development

4.2.1 Arduino Mega



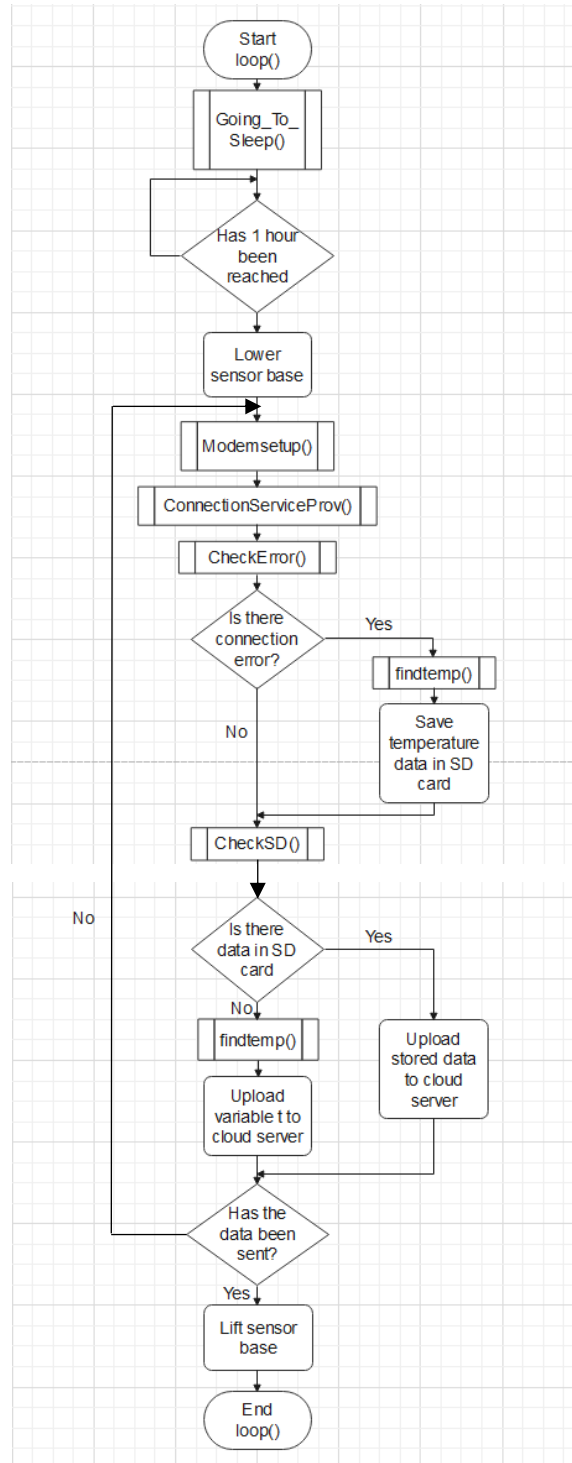


Figure 4.3: Main program flow chart of Arduino Mega

Figure 4.3 shows the main program flow chart for the Arduino Mega. The main program starts by including relevant libraries for the temperature sensor, GPRS/GSM, SD card and RTC modules. After that important header files are defined such as SoftwareSerial.h, String.h, OneWire.h, etc. Global variables are then defined and the setup() can now begin. After the setup(), the main program will run a loop() function which will loop for an infinite amount of time before its power supply is depleted.

For the `setup()`, the first step is to setup serial communications between the Arduino Mega and the SIM900a GSM/GPRS module because the GSM/GPRS module communicates with Arduino Mega through UART. The serial communication between the Arduino Mega and the laptop is also set up for debugging purposes or to upload new code to the system. Next, the `rtc_setup()` begins.

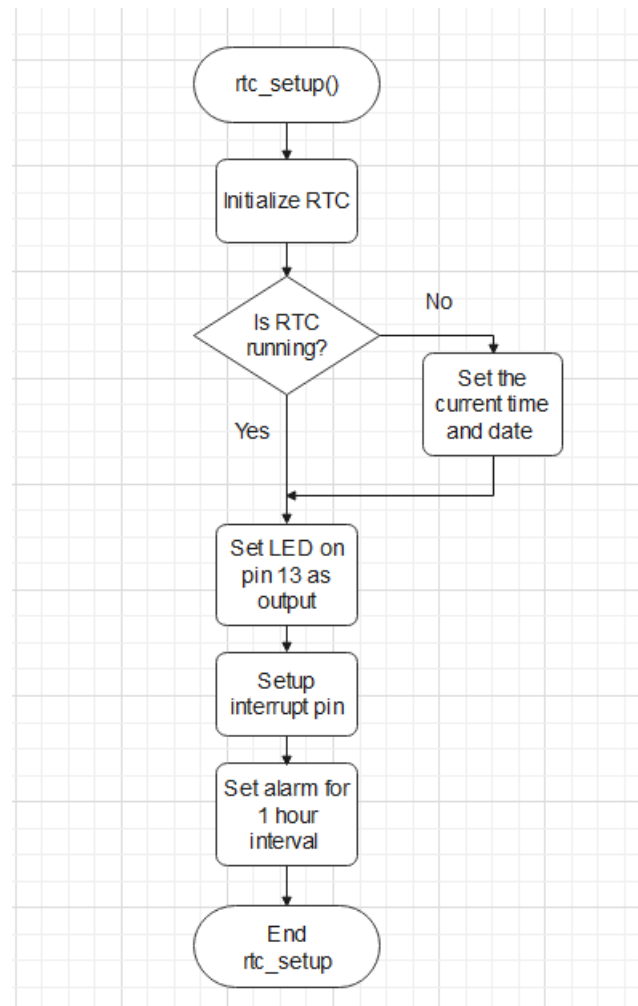


Figure 4.4: Flow chart of `rtc_setup()`

Figure 4.4 shows the flow chart for the RTC module. First, the RTC module is initialized and then the RTC module is checked to see if it is running. If the RTC is not running the current time and date is set. However, if the RTC module is running, pin 13 of the Arduino Mega is set as an output to indicate whether the Arduino Mega is in sleep mode or awake. For example, if LED at pin 13 is high, the Arduino Mega is awake. If LED at pin 13 is low, the Arduino Mega is in deep sleep mode. Next, the interrupt pin is set to interrupt and wake the Arduino Mega from the sleep mode. Lastly, the alarm is set at one hour interval to wake up the Arduino Mega from sleep mode, this is to ensure less power is

consumed when the Arduino Mega is not collecting data. After that, the `sd_card_setup()` begins.

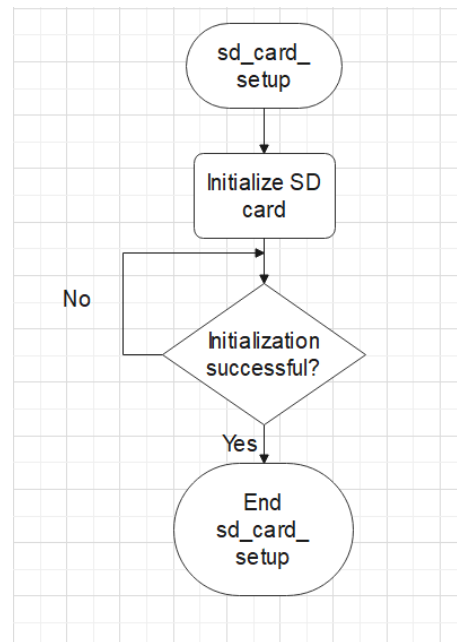


Figure 4.5: Flow chart of `sd_card_setup()`

Figure 4.5 shows the flow chart for the set up of the SD card module. First, the SD card is initialized, then the SD card is checked to see if it is initialized successfully. If the initialization fails, the program will run a `while(1)` loop until the initialization is successful. After the initialization of the SD card is done, the setup has completed.

As for the `loop()` function, the program runs in a loop until the microcontroller runs out of power. The first step is to put the microcontroller to sleep by using the `Going_To_Sleep()` function.

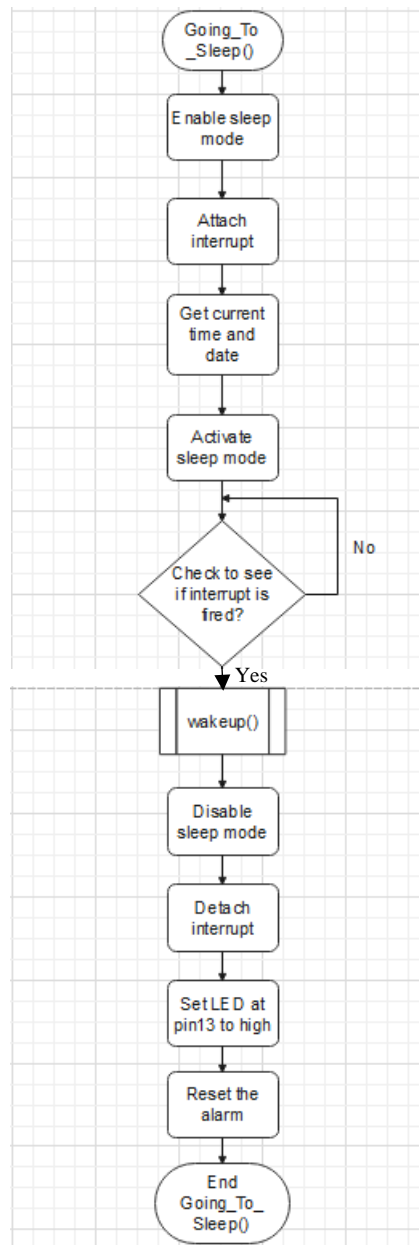


Figure 4.6: Flow chart for Going_to_sleep function

Going_to_sleep function begins with the enabling of the sleep mode feature in the microcontroller and then an interrupt is attached. Next, the sleep mode is activated. The interrupt is set so that it can wake up the microcontroller when the RTC module detects that 1 hour interval has been reached. The program checks if the interrupt has been fired which is only possible if 1 hour has been reached. If the interrupt is fired, the wakeup() function begins where the sleep mode is disabled and the interrupt is detached. The LED at pin 13 is set to high to indicate the microcontroller is awake and the alarm is reset for the next 1 hour interval.

Based on Figure 4.3, when the 1 hour interval has been reached, the sensor base is lowered into the water using the power window motor via a hoist. After that, the setup

for the GPRS/GSM module is done. The setup is separated into 2 functions which are the Modemsetup() and ConnectionServiceProv(). Next, CheckError() function is run to see if there is any error that occurs due to inability to connect to the network provider. If there is an error, the temperature data is taken via the findtemp() function and saved into the SD card. If there is no error, the SD is checked using the function CheckSD() to see if there is any data in the SD card. If there is data in the SD card, the stored data is uploaded to the cloud server. If there is no data in the SD card, the temperature data is taken and saved into the variable t which is then uploaded to the cloud server. Lastly, the program checks to see if the data is sent to the cloud server successfully. If the data is sent successfully, the sensor base is lifted up and the process is repeated for the next hour. If the data is not sent, it will begin with the Modemsetup() function again and the process is repeated until the data is sent.

4.2.2 SIM900a GPRS/GSM module

As for the SIM900a GPRS/GSM module, the program must pass 4 main functions which are Modemsetup(), ConnectionServiceProv(), Senddata() and CheckError() before data can be successfully sent to the ThingSpeak cloud server. The first function is Modemsetup() which is basic AT commands that setup the modem to send data to the cloud server.

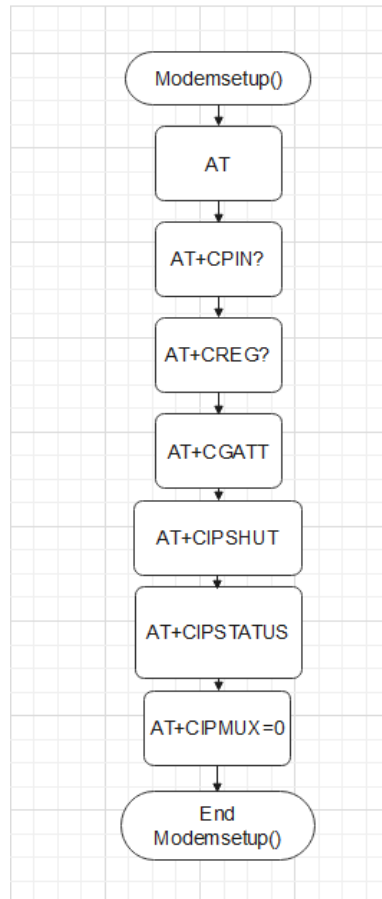


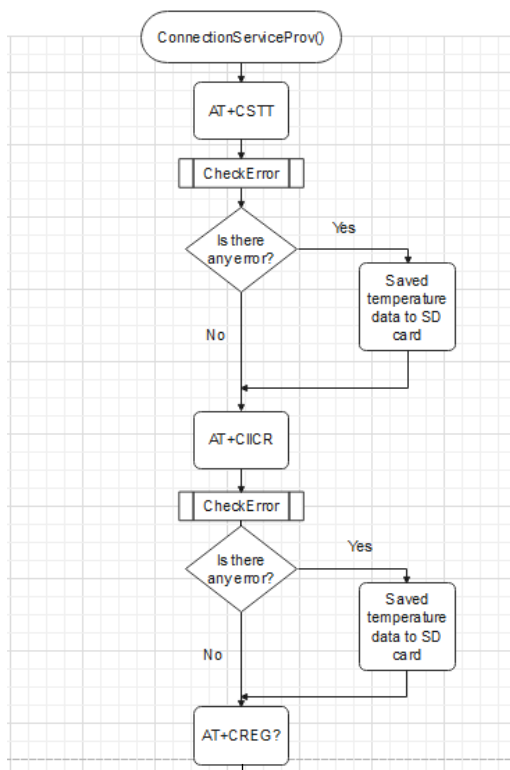
Figure 4.7: Flow chart for the function Modemsetup()

The process for the Modemsetup() is straightforward and each AT command will generate a response and the response will be sent to the microcontroller via UART. In order for the data to be sent successfully, each AT command must produce an ‘OK’ response before proceeding to the next AT command. The table below shows the description for each AT command.

Table 4.2: AT commands for the function Modemsetup()

AT Command	Description
AT	Start the command line
AT+CPIN	Enter PIN for mobile device
AT+CREG	Network registration
AT+CGATT	Attach or detach from GPRS Service
AT+CIPSHUT	Deactivate GPRS PDP Context
AT+CIPSTATUS	Query Current Connection Status
AT+CIPMUX=0	Start up multi-IP connection

Next, we have the function ConnectionServiceProv() where the AT commands under this function is used to establish a connection to the network provider.



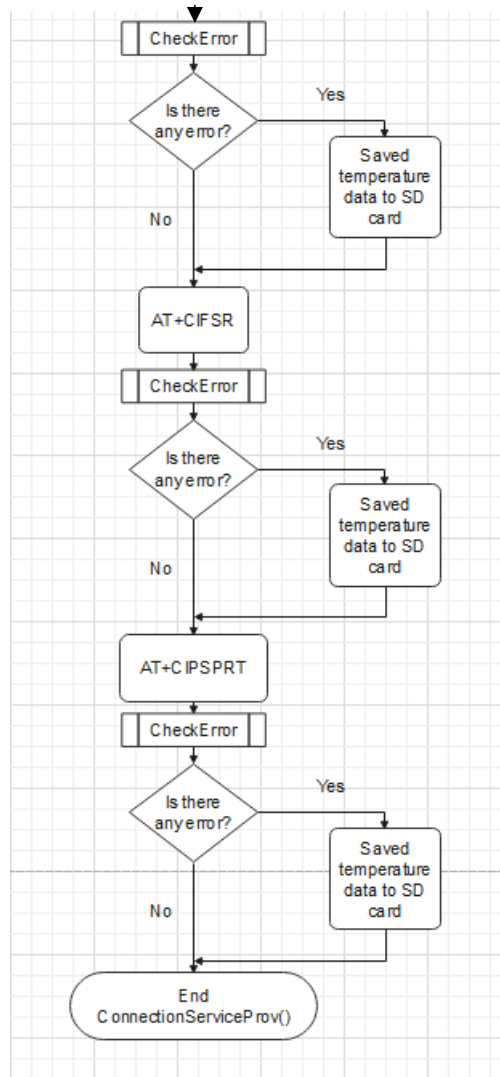


Figure 4.8: Flow chart for the function ConnectionServiceProv()

For the flow chart in Figure 4.7, after an AT command is executed, the program then proceeds to the CheckError() function which checks to see if there is any error produced by the GPRS/GSM module. If there is an error, the temperature data is saved into the SD card.

Table 4.3: AT commands for the function ConnectionServiceProv()

AT Command	Description
AT+CSTT	Start task and set APN, Username and Password
AT+CIICR	Build up wireless connection with GPRS
AT+CIFSR	Get local IP address
AT+CIPSPRT	Set prompt of '>' when module sends data

Based on Table 4.3, there is one important command which is the AT+CSTT command. In order to properly configure the APN, Username and Password, the APN of the network provider written as such 'AT+CSTT="diginet\''. The network provider that is used in this project is Digi Telecommunications. Other network providers can be used but the APN must be changed accordingly. The AT+CIICR and AT+CIFSR commands are used to build up wireless connection and get the local IP address which is very important commands to establish connection to the network provider.

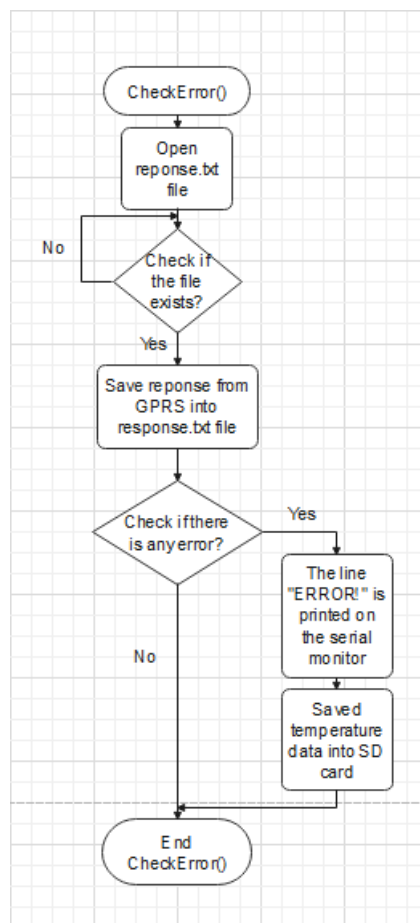
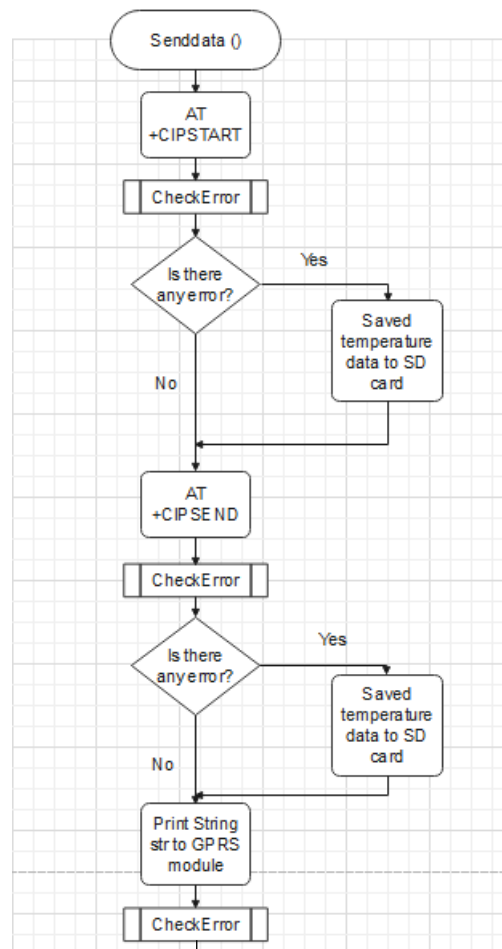


Figure 4.9: Flow chart for the function CheckError()

Figure 4.8 shows the flow chart for the CheckError() function. The first step is to open the text file named 'response.txt'. Each line of response generated from the GPRS module is saved into a String variable which is named as 'list' and the response in the text file is checked to determine if there is an ERROR response. If there is an ERROR response, an "ERROR!" line is printed onto the serial monitor to show the programmer that there is an error that has occurred. After that, the temperature data is saved. If there is no error, the CheckError() function ends.

Lastly, is the Senddata() function which consists of AT commands which is used to send the temperature data to the cloud server via the GPRS/GSM module.



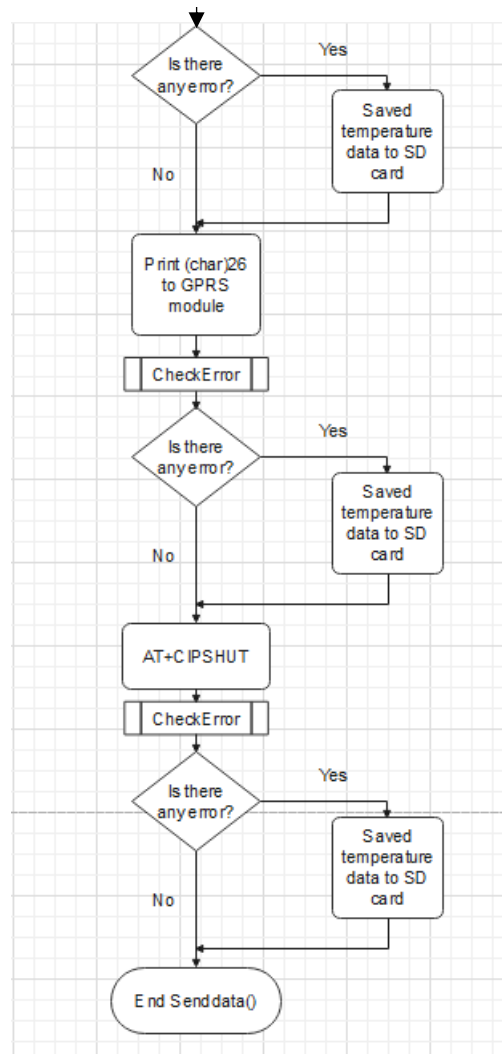


Figure 4.10: Flow chart for the function Senddata()

The first step for the Senddata() function is to start up the Transmission Control Protocol (TCP) connection to the ThingSpeak cloud server using the AT+CIPSTART command. Next, the AT+CIPSEND command is executed to start sending data to the cloud server. After that a String, str is sent to the cloud server. The code for the str is written as such “str = GET https://api.thingspeak.com/update?api_key=ZEF3DQ3PMS3G7LKO&field1=” + String(x,1)”. Once a channel has been created on the ThingSpeak webpage, the API key is given and must be copy and pasted into the above String variable str. The API key is “ZEF3DQ3PMS3G7LKO&field1”. This is to ensure the data is sent to the right channel. The variable x is the temperature variable which will be sent to the cloud server. After that, (char)26 is sent to the GPRS module to start sending data to the cloud server. After the data is sent, AT+CIPSHUT is executed to close the connection and stop sending the data.

Table 4.4: AT commands for the function Senddata()

AT Command	Description
AT+CIPSTART	Start up TCP connection
AT+CIPSEND	Send data through TCP connection
AT+CIPSHUT	Deactivate GPRS PDP Context

4.2.3 Micro SD card

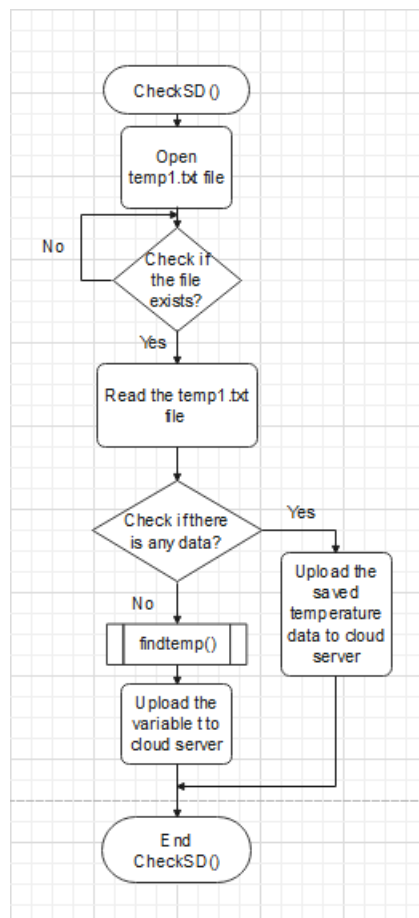


Figure 4.11: Flow chart for the function CheckSD()

As for the SD card, the first step is to open the temp1.txt file. Next, the file is checked to determine if it exists. If the file exists, then the temp1.txt file is read. When there is an error during the CheckError() function, temperature data will be saved into the temp1.txt file. If there is data in the text file, it will be uploaded to the cloud server. If there is no data in the SD card, the temperature is obtained via the findtemp() function and then sent to the cloud server. The findtemp() function is only used to obtain the temperature value.

4.3 Project Prototype

4.3.1 Electronics Box

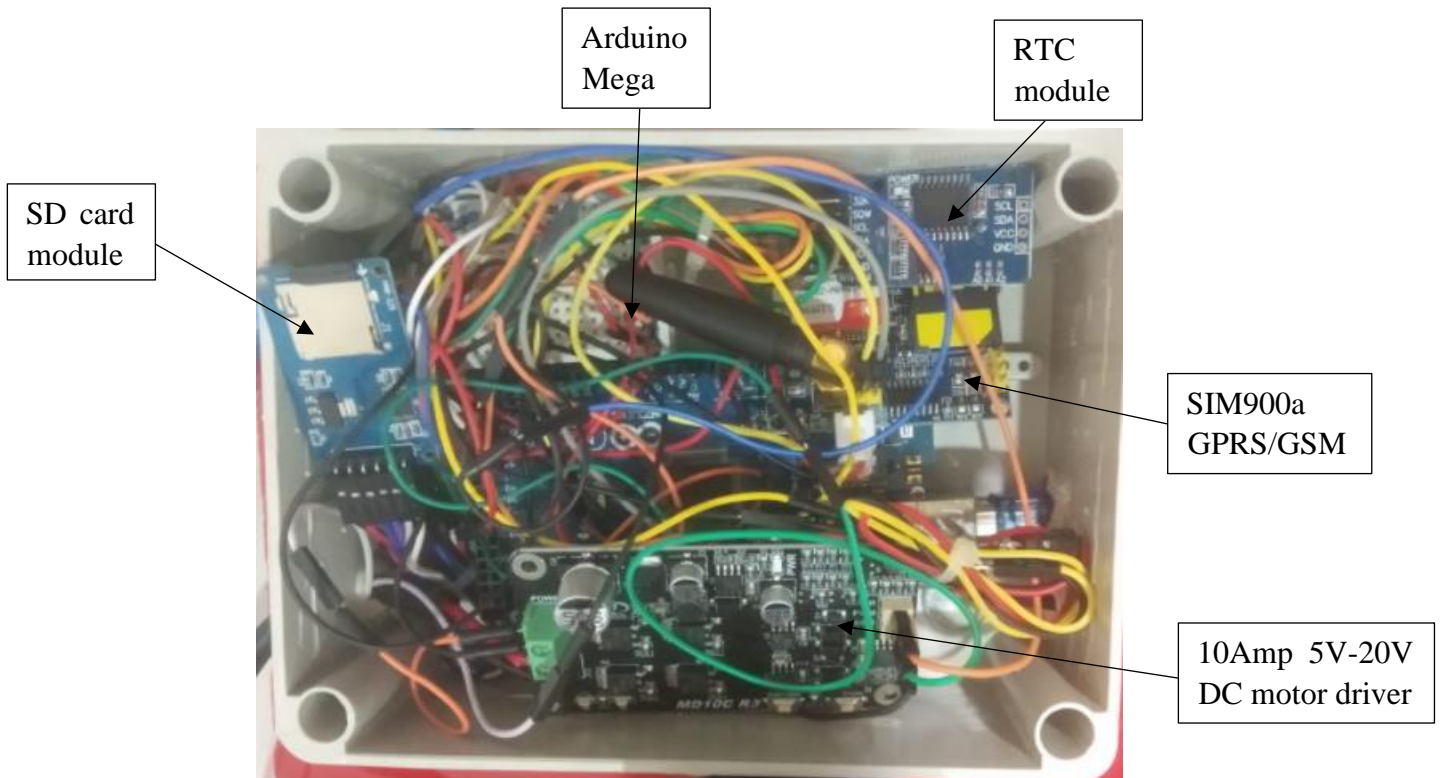


Figure 4.12: Electronics box

Figure 4.11 shows the electronics box used to house most of the electronics and the connection is based on subsection 4.1.1 Hardware design. The electronics box is waterproof to prevent any water from entering and destroying the circuit. The size is 4inch x 6inch x 3inch which is small and compact. The electronics box is made up of PVC and there is holes drilled into it to allow USB cords and power cords to extend to the microcontroller or the power window motor.

4.3.2 Power motor coupler and power motor stand



Figure 4.13: Complete assembly of the power motor, power motor coupler and the power motor stand

The complete assembly for the power motor is shown in Figure 4.12. The power motor coupler is where the fishing line is wound to lift and lower the sensor base that is attached to the end of the fishing line. The power motor stand is to hold the power motor in place via 4 x 6mm screws.



Figure 4.14: Top view of power motor coupler

The power coupler is attached to the rotating section of the power motor and is attached through 3 x 5mm screws. The power coupler was first modelled using the Solidworks software and then printed using an Ender 3 3D printer.

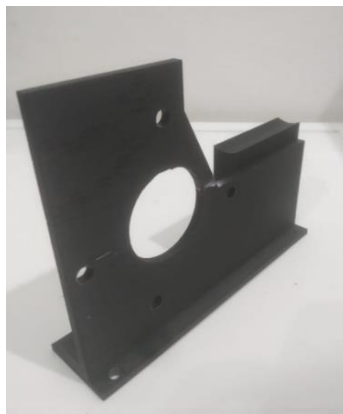


Figure 4.15: Isometric view of power motor stand

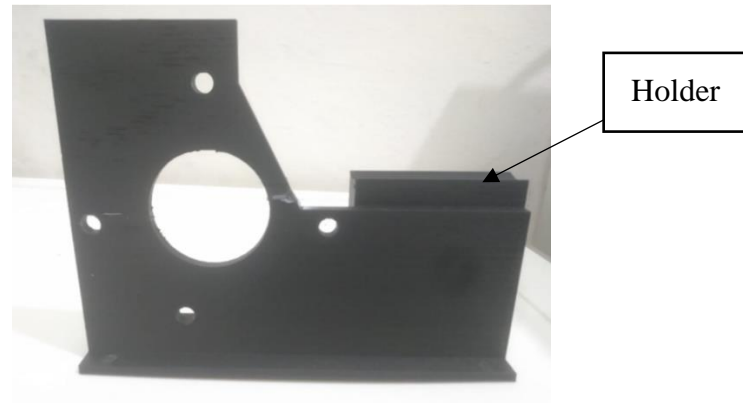


Figure 4.16: Side view of power motor stand

Figure 4.14 and 4.15 shows the isometric and side view of the power motor stand. The power motor is placed on top of the holder because the bulk of the weight of the power motor is in at the back. This prevents the power motor from tilting backwards. The power motor is attached using 4 x 6mm screws to hold the power motor firmly in place. The power motor stand was also first modelled in the Solidworks software and then printed using the Ender 3 3D printer.

4.3.3 Aluminium Beam, Pulley and Fishing Line

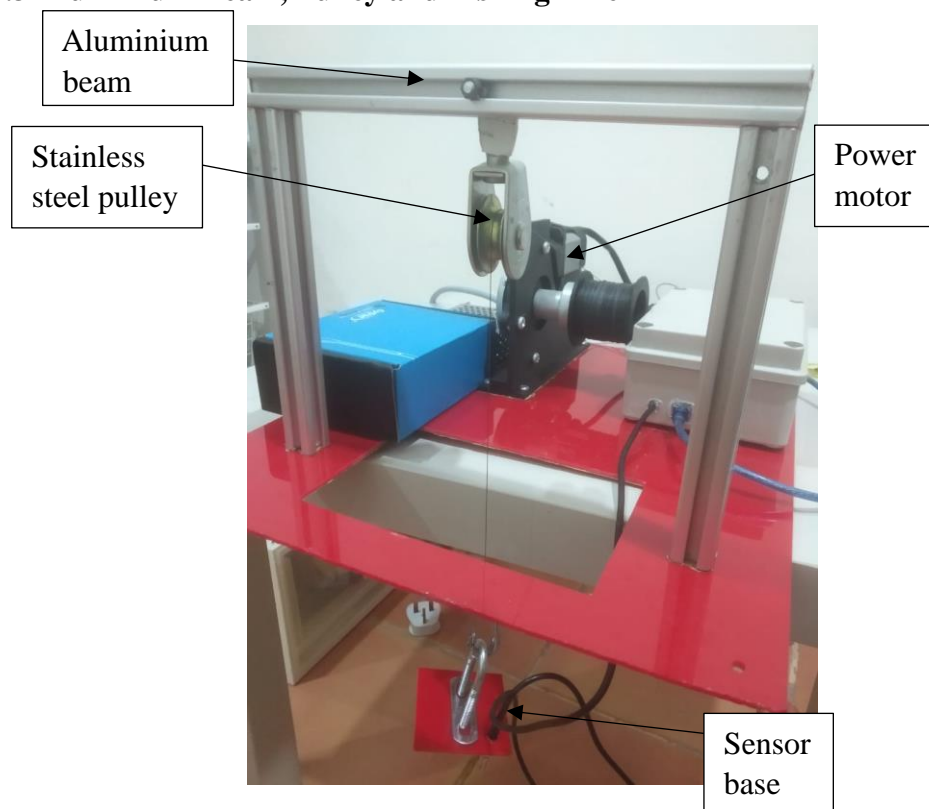


Figure 4.17: Lifting and lowering mechanism of the system

Figure 4.16 shows the lifting and lowering mechanism of the system which consist of the power motor, aluminium beam, stainless steel pulley and the sensor base. The power motor is responsible for lifting and lowering the sensor base via the stainless steel pulley. The fishing line is used to attach the sensor base to the power motor coupler. The temperature sensor is then attached to the sensor base so when it is lowered, the temperature sensor is submerged in the water.

4.3.4 Sensor Base



Figure 4.18: Top view of sensor base



Figure 4.19: Side view of sensor base

Figure 4.17 and 4.18 shows the top and side view of the sensor base. The purpose of the sensor base is to allow multiple sensors to be attached to it so that when it is lowered into

the water, the sensors can collect data. The base is made of acrylic and thus does not deteriorate over time when submerged in salt or fresh water. The sensor base has a U-bolt which is attached to the snap hook for flexible movement in case the water is turbulent.

4.3.5 Final Prototype

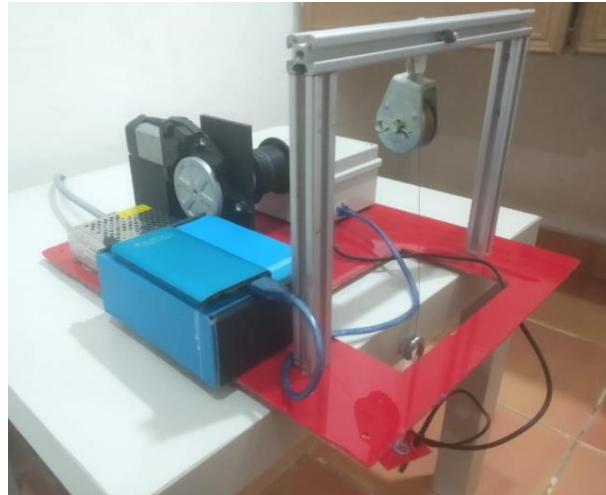


Figure 4.20: Isometric view of final prototype

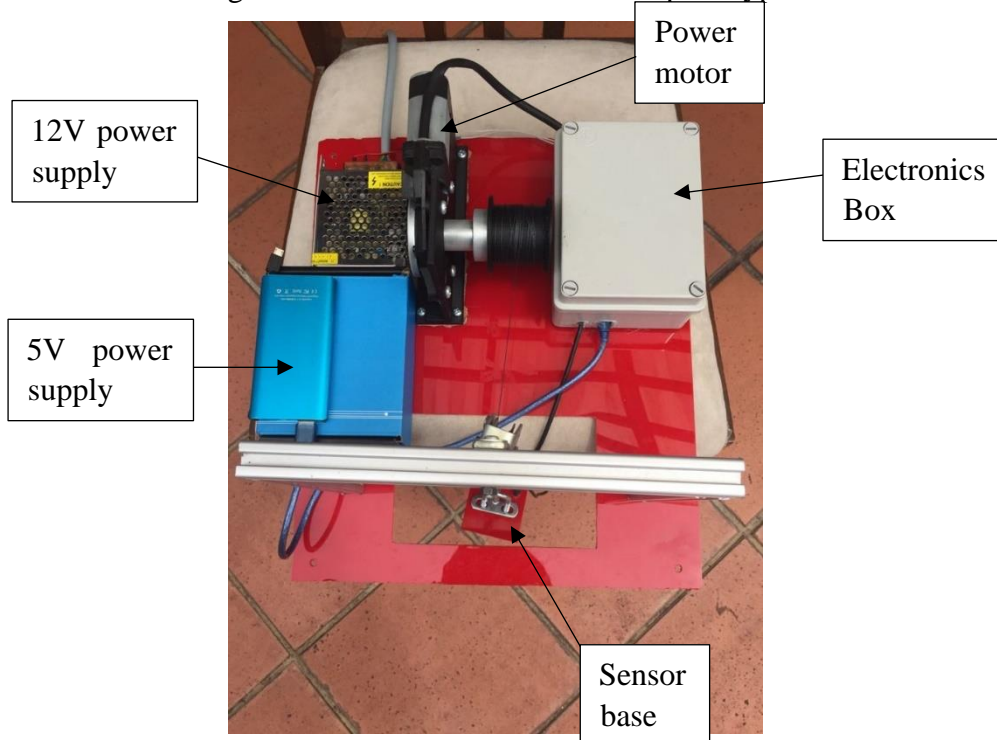


Figure 4.21: Top view of final prototype

Figure 4.19 and 4.20 shows the isometric and top view of the final prototype. The 5V power supply provides power to the microcontroller and the 12V power supply provides power to the power motor. The final prototype is portable and easily transportable which weigh about 2.6kg.

4.4 Features and Performance of Prototype

4.4.1 Data Storage

One of the features of the system is data storage. The advantage of having data storage is that data loss is prevented. Data is important for data analysis or to construct predictive models. The data is stored in a SD card which has 1 Gigabyte of memory, this is sufficient memory to store a lot of data since the temperature data is stored in a text file. If data is logged 1 hour per day, the text file typically occupies 4 kilobytes of memory. Thus, the data storage can be done for 250,000 days.

Other than that, data is also sent to the ThingSpeak cloud server where the data can be exported to a JSON, XML or CSV file. The CSV file can be opened in Microsoft Excel to plot the temperature to show the temperature data to compare with the data in the SD card to check for any data loss.

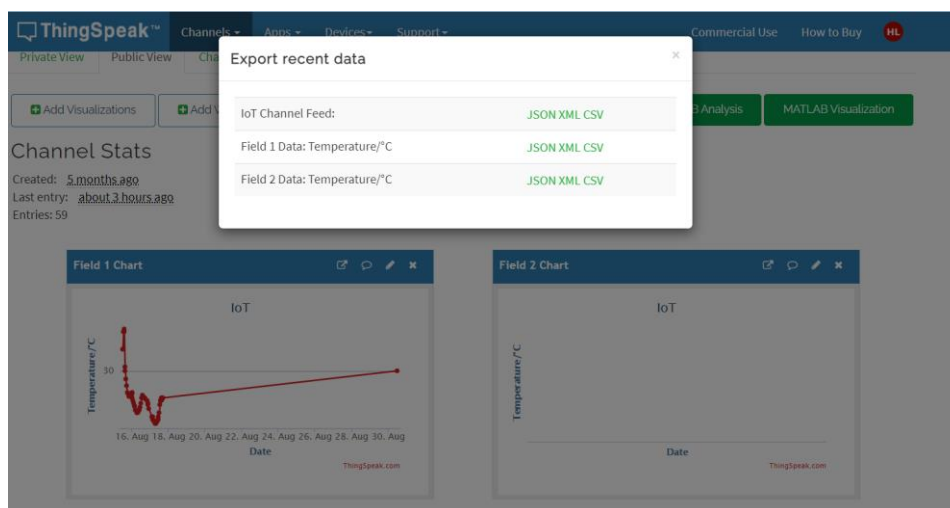


Figure 4.22: Export of data to JSON, XML or CSV file

4.4.2 Data Monitoring through cloud server and mobile

The system is integrated with IoT which means data monitoring can be done through the ThingSpeak cloud server or the ThingShow mobile application.

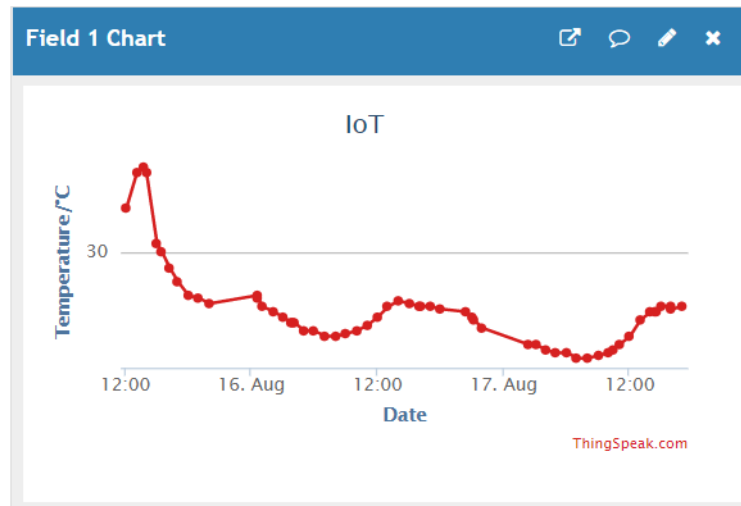


Figure 4.23: Temperature data on ThingSpeak cloud server

Figure 4.22 shows the temperature data trend on ThingSpeak cloud server. The server is used mainly for data monitoring using IoT and is free of charge. The only requirement is to register for an account and create a channel. There is also a limit for the number of data points per channel which is 8000 data points. The system collects data at an interval of 1 hour which means there is 24 data points per day. This means that for one channel, data can be collected continuously and stored for about 333 days.

Data can also be exported in JSON, XML and CSV files for data analysis. Furthermore, the upload of data to the channel is simple, by changing obtaining the API key from a new channel and updating the API key in the source code. Data can be sent to the new channel. For example, humidity data is required, a new channel can be created and the API key of that channel is copied and paste in the source code and data can be updated in the channel. There is however a limit of 8 channels which means only 8 types of water parameters can be measured.



Figure 4.24: Temperature data point

By clicking on the data points, the temperature, date and time can be obtained. The date consists of the month, day and year of when the data is collected.

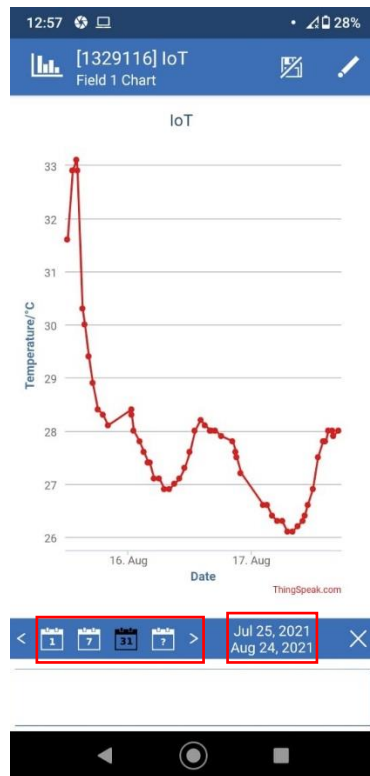


Figure 4.25: Temperature data on ThingShow mobile application

The ThingShow mobile application is used mainly to view historical data in graphical format. For example, the time of the creation of the channel is 25th July 2021 and the latest entry of data is 24th August 2021 which is shown at the bottom right of the mobile screen. The bottom left of the screen shows the options for displaying the data in daily, weekly, or monthly format.

4.5 Problems encountered and Solution

4.5.1 Power Supply

One of the issues faced in the prototype testing phase was the power consumption was too high that the system was not sustainable. The power consumption was measured to be at an average of 3.4A. The solar cell can only charge for 250mAh and the 14500 Li-Ion battery can charge at a rate of 600mAh. The system only lasted approximately 2 hours. One other problem faced was that the system turns ON and OFF occasionally due to power supply fluctuations. This causes the interrupt to trigger as a result and data is taken randomly.

Table 4.5: Power consumption of various modules

Module	Power Consumption (mA)
Arduino Mega (in sleep mode)	80
SIM900a GPRS/GSM module	500 (idle) 2000 – 3000 (transmitting data)
SD card	30 (in SPI mode)

Table 4.5 shows the power consumption of the various modules in the system. The total theoretical power consumption is estimated to be 3.11A which is similar to the measured power consumption. This suggests that the power consumption is too high. Due to time constraint, only the Arduino Mega is put to sleep mode. Whereas the SIM900a GPRS/GSM module is kept operating without entering sleep mode.

The solution lies in reducing the power consumption of the SIM900a GPRS/GSM module which is done by executing the AT+CSCLK AT command which will put the module into sleep mode. Thus, reducing the total power consumption to about 610mA. The other problem where the discharge rate is higher than the charge rate can be fixed by adding 2 more solar panels. The total charging rate would amount to 750mAh and another 14500 Li-Ion battery must be added to ensure the energy storage is enough.

4.5.2 Power Motor Stand

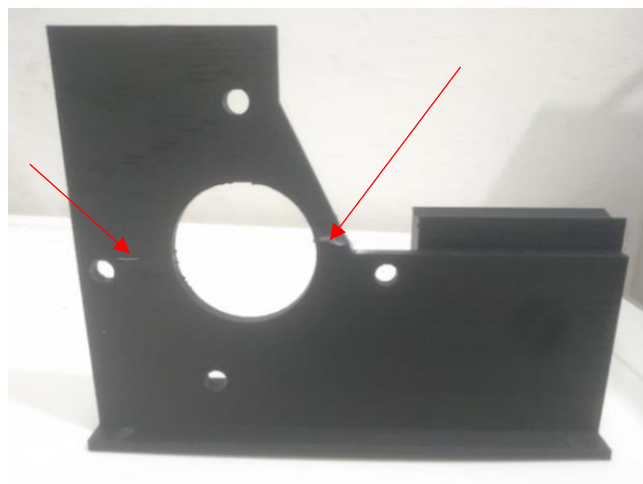


Figure 4.26: Crack marks at the power motor stand

Another problem faced was the breaking of the power motor stand due to faulty design. The thickness of stand was set to 3mm so when excessive force was applied to it, the stand broke in half indicated by the red arrows. Though during the testing phase, there was no issues. This is an important problem that needs to be addressed as the likely mechanical failure could occur at this spot.

The solution is to simply design the stand such that the thickness of the stand is increased until it does not break easily or increase the fill of the stand. The fill of the stand was set at 80% because of time constraint. If the fill is set to 100%, it may take longer to produce the stand but it will be stronger and stiffer. The thickness should also be increased to at least 5mm to prevent it from breaking.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

The development of the static water quality monitoring system and the integration of the IoT was successful. The water parameter that can be measured is the temperature of the water. Additional sensors have to be added to measure other water parameters. The system is able to monitor the temperature at a fixed interval of 1 hour and upload the data to the cloud server reliably.

The feature of the system includes data storage and data monitoring via the ThingSpeak cloud server or the ThingShow mobile application. The historical data can be observed in both the ThingSpeak cloud server and the ThingShow mobile application.

During the testing phase of this project, several problems has occurred as such high power consumption and the failure of the power motor stand. The solution is to put the SIM900a GPRS/GSM module to sleep mode to consume less power. As for the power motor stand, the thickness of the power motor stand should be increased to at least 5mm for the next design.

In conclusion, the aim and objectives of this project that is to develop an embedded system architecture that performs hourly monitoring of the water quality parameters while integrating Internet of Things (IoT) has been achieved. The data is successfully shown in the ThingSpeak cloud server and in the ThingShow mobile application.

5.2 Limitation of The Prototype

One major limitation of the prototype is that this system is static which means that it can only obtain data at a single location at a time. Mobile monitoring systems has the advantage of gathering data at multiple locations. This means that in order to gather data from multiple locations, the aquafarmer has to venture to the locations and deploy the system or the aquafarmer may need to purchase multiple static water quality monitoring systems which is expensive.

Another limitation is that the prototype is only tested using laboratory conditions which means the environment is controlled and external factors such as wind, rain or

extreme weathers are not taken into consideration. Due to Movement Control Order (MCO), the travel restrictions have prevented the system to be field tested. This means that a lot more field tests must be conducted to ensure the system can run reliably.

The last limitation of the prototype is that the prototype does not have a floatation device. The prototype will be mainly deployed in areas surrounded with water. This means that if the system is submerged it will sink to the bottom and it will not be retrievable. This is very important because if a floatation device is added, the system will still be retrievable if it is displaced from its current location.

5.3 Recommendations For Future Work

The first recommendation for future work is to improve the sustainability of the system. This is discussed in subsection 4.5.1 under Power Supply. This is important because the system is currently unsustainable and the system can only run for about 2 hours at once which is definitely not sufficient. Water quality monitoring system must be able to run for at least a week or maybe even months. The system must be fitted with more solar panels and Li-ion batteries with the correct power ratings to ensure there the charge rate is high enough and the energy storage is sufficient. More testing must also be done to test the reliability of the system over long duration of time.

The second recommendation is to install a waterproof enclosure around the whole water quality monitoring system. 4 x 5mm holes has been drilled into the corners of the base of the system. This is to ensure the enclosure can be attached using screws to the corners. The enclosure must also be waterproof, sealants must be added to the sides to prevent any water from entering the system and destroying the circuitry.

The last recommendation is related to the 12V power supply, the current configuration is using a 12V lead-acid battery. The lead-acid battery can last for a long duration of time but it will be better if the 12V power supply is sustainable. Therefore, a similar circuit to the 5V power supply can be constructed but the solar panel must have a higher power rating since the 12V power supply consumes more energy. The solar panel will directly supply solar energy to the lead-acid battery which in turn charges it.

REFERENCES

1. Troell, M., Naylor, R. & Metian, M., 2014. *Does aquaculture add resilience to the global food system?*. The State University of New Jersey, New Brunswick: National Academy of Sciences.
2. Kituyi, M. & Thomson, P., 2018. *90% of fish stocks are used up - fisheries subsidies must stop*. s.l.:World Economic Forum .
3. FAO, 2016. *The State of World Fisheries and Aquaculture 2016, Contributing to Food Security and Nutrition For All*. Rome: The State of World Fisheries and Aquaculture 2016.
4. Fondriest Environmental, I., 2013. *pH of Water*. [Online] Available at: <https://www.fondriest.com/environmental-measurements/parameters/water-quality/ph/> [Accessed 19 November 2013].
5. Boyd, C. E., 2018. *Water temperature in aquaculture*. [Online] Available at: <https://www.aquaculturealliance.org/advocate/water-temperature-in-aquaculture/>
6. Towers, L., 2015. *Water quality a priority for successful aquaculture*. [Online] Available at: <https://thefishsite.com/articles/water-quality-a-priority-for-successful-aquaculture/>
7. Rowe, D., 2002. *Lethal turbidity levels for common freshwater fish and invertebrates in Auckland streams*. Auckland: s.n.
8. Tarik, M., 2020. *Smart Water Quality Monitoring System Based On Iot*. 7 ed. s.l.:Journal of Critical Reviews.
9. N.Vijayakumar & Ramya, R., 2015. *The Real Time Monitoring of Water Quality In IoT Environment*. 4 ed. s.l.:International Journal of Science and Research (IJSR).
10. Faustine, A., 2014. *Wireless Sensor Networks for Water Quality Monitoring and Control within Lake Victoris Basin: Prototype development*. Dodoma: SciRes.

11. Geetha, S., 2017. *Internet of things enabled real time water quality monitoring system*. [Online]
Available at: <https://smartwaterjournal.springeropen.com/articles/10.1186/s40713-017-0005-y#ref-CR31>
[Accessed 27 July 2017].
12. Vinod, R. & Sushama, S., 2016. *Wireless acquisition system for water quality monitoring*. In: *Conference on advances in signal processing*. s.l.:s.n.
13. Peng, J., Hongbo, X., Zhiye, H. & Zheming, W., 2009. *Design of a Water Environment Monitoring System Based on Wireless Sensor Networks*. s.l.:s.n.
14. Hall, J. et al., 2007. *On-line water quality parameters as indicators of distribution system*. s.l.:Journal of the American Water Works Association.
15. Inc., X., 2015. *Water Quality Instrumentation*. [Online]
Available at: http://www.globalw.com/catalog_wq.html
16. Alexander, T., Michelle, V. & Evelyn, B., 2019. *A system for monitoring water quality in a large aquatic area using wireless sensor network technology*. s.l.:Sustain Environment Res 29,12.
17. M., C. Z., 2019. *Real-time water quality system in internet of things*. s.l.:IOP Conf. Ser.:Mater. Sci. Eng. 495 012021.
18. Kofi, S., A.K., F., D.A., J. & Felicia, E., 2020. *Smart River Monitoring Using Wireless Sensor Networks*. In: *Wireless Communications and Mobile Computing Volume 2020*. s.l.:s.n., p. 5.

APPENDICES

APPENDIX A: Datasheets

AtlasScientific™
Environmental Robotics
V 5.4
Revised 7/20

EZO-DO™

Embedded Dissolved Oxygen Circuit

Reads	Dissolved Oxygen
Range	0.01 – 100+ mg/L 0.1 – 400+ % saturation
Accuracy	+/- 0.05 mg/L
Response time	1 reading per sec
Supported probes	Any galvanic probe
Calibration	1 or 2 point
Temperature, salinity and pressure compensation	Yes
Data protocol	UART & I ² C
Default I ² C address	97 (0x61)
Operating voltage	3.3V – 5V
Data format	ASCII

Written by Jordan Pease
Designed by Neah Pease

PATENT PROTECTED

This is an evolving document, check back for updates.

Figure A-1: Specifications for Atlas Scientific DO kit



Part Number: SEN0161

Description: Gravity: Analog pH Sensor / Meter Kit For Arduino

SPECIFICATION

- Module Power : 5.00V
- Module Size : 43 x 32mm(1.69x1.26")
- Measuring Range :0 - 14PH
- Measuring Temperature: 0 - 60 °C
- Accuracy : ± 0.1pH (25 °C)
- Response Time : ≤ 1min
- pH Sensor with BNC Connector
- pH2.0 Interface (3 foot patch)
- Gain Adjustment Potentiometer
- Power Indicator LED

Figure A-2: Specifications for SEN0161 Gravity analog pH sensor

APPENDIX C: Coding

```

//pin numbers
//RTC SDA,SCL,SQW = 20,21,18
//SD card MISO,MOSI,SCK,CS = 50,51,52,53
//GPRS 5VR,5VT = 13,12
//Motor driver pwm,dir = 3,2
//sensor 5

#include <SoftwareSerial.h>
#include <String.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <SPI.h>
#include <SD.h>
#include <Wire.h>
#include <DS3232RTC.h>
#include <avr/sleep.h>
#include "RTClib.h"
SoftwareSerial gprsSerial(12,13);
RTC_DS1307 rtc;
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
File myFile;
File myFile1;

int count =0; //so that data is stored once
int pwm_value;
const int time_interval=1; //1 hour time interval
//for testing
//const int time_interval =2; //2 min time interval
float celsius, fahrenheit;
float temp;
boolean hasCheck = false;
boolean tempsaved =false;
boolean sent = true;

#define ONE_WIRE_BUS 5
#define interruptPin 18
#define pwm 3
#define dir 2

void setup() {
  Serial.begin(9600);
  gprsSerial.begin(9600);
  // Open serial communications and wait for port to open:
  delay(1000);

  //setup clock
  if (! rtc.begin()) {
    Serial.println("Couldn't find RTC");
    Serial.flush();
    abort();
  }

  if (! rtc.isrunning() ) {
    Serial.println("RTC is NOT running, let's set the time!");
    // When time needs to be set on a new device, or after a power loss, the
    // following line sets the RTC to the date & time this sketch was compiled
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
    // This line sets the RTC with an explicit date & time, for example to set
    // January 21, 2014 at 3am you would call:
    // rtc.adjust(DateTime(2014, 1, 21, 3, 0, 0));
  }
}

```

```

//setup for RTC
DateTime now = rtc.now();
Serial.println("Current time is:");
Serial.print(now.year());
Serial.print('/');
Serial.print(now.month());
Serial.print('/');
Serial.println(now.day());
Serial.print(now.hour());
Serial.print('-');
Serial.print(now.minute());
Serial.print('-');
Serial.println(now.second());

pinMode(LED_BUILTIN,OUTPUT);//We use the led on pin 13 to indicate when Arduino is A sleep
pinMode(interruptPin,INPUT_PULLUP);//Set pin d2 to input using the builtin pullup resistor
digitalWrite(LED_BUILTIN,HIGH);//turning LED on

//Setup for RTC alarm
// initialize the alarms to known values, clear the alarm flags, clear the alarm interrupt flags
RTC.setAlarm(ALM1_MATCH_DATE, 0, 0, 0, 1);
RTC.alarm(ALARM_1);
RTC.alarmInterrupt(ALARM_1, false);
RTC.squareWave(SQWAVE_NONE);

time_t t; //create a temporary time variable so we can set the time and read the time from the RTC
t=RTC.get();//Gets the current time of the RTC

RTC.setAlarm(ALM1_MATCH_HOURS , 0, 0, hour(t)+time_interval, 0);
//for testing
//RTC.setAlarm(ALM1_MATCH_MINUTES , 0, minute(t)+time_interval, 0, 0);
// clear the alarm flag
RTC.alarm(ALARM_1);
// configure the INT/SQW pin for "interrupt" operation (disable square wave output)
RTC.squareWave(SQWAVE_NONE);
// enable interrupt output for Alarm 1
RTC.alarmInterrupt(ALARM_1, true);

//Initializing card
Serial.print("Initializing SD card...");
if (!SD.begin(53)) {
    Serial.println("initialization failed!");
    while (1);
}
Serial.println("initialization done.\n");

//setup for motor pins
pinMode(pwm,OUTPUT);
pinMode(dir,OUTPUT);
}

```

Figure C-1: Coding for setup of main program

```

void loop() {
  sent = true;

  Going_To_Sleep();

  //lower the cable
  digitalWrite(dir,LOW); // rotate anticlockwise
  pwm_value=200;
  analogWrite(pwm,pwm_value);
  delay(1000);

  //stop the motor
  pwm_value=0;
  analogWrite(pwm,pwm_value);
  delay(1000);

  //while loop in case there is any error
  while(sent){

    Serial.print("Program Start\n-----\n");

    myFile = SD.open("templ.txt", FILE_WRITE);
    myFile.close();

    Modemsetup(); //modemsetup

    ConnectionServiceProv(); //connect to service provideer

    CheckSD(); //check SD card for data

    Serial.print("Upload done\n\n");
    Serial.print("Program End\n-----\n");

  }

  //lift the cable
  digitalWrite(dir,HIGH); // rotate clockwise
  pwm_value=200; // control speed
  analogWrite(pwm,pwm_value);
  delay(1000);

  //stop the motor
  pwm_value=0;
  analogWrite(pwm,pwm_value);
  delay(1000);
}

```

Figure C-2: Coding for loop of main program

```
void Modemsetup() {
  gprsSerial.println("AT");
  delay(1000);

  gprsSerial.println("AT+CPIN?");
  delay(1000);

  gprsSerial.println("AT+CREG?");
  delay(1000);

  gprsSerial.println("AT+CGATT?");
  delay(1000);

  gprsSerial.println("AT+CIPSHUT");
  delay(1000);

  gprsSerial.println("AT+CIPSTATUS");
  delay(1000);

  gprsSerial.println("AT+CIPMUX=0");
  delay(1000);

  Serial.print("Modem setup done!\n\n");
}

void ConnectionServiceProv() {
  gprsSerial.println("AT+CSTT=\"diginet\""); //start task and setting the APN,
  delay(3000);
  CheckError();

  gprsSerial.println("AT+CIICR"); //bring up wireless connection
  delay(3000);
  CheckError();

  gprsSerial.println("AT+CIFSR"); //get local IP adress
  delay(2000);
  CheckError();

  gprsSerial.println("AT+CIPSPRT=0");
  delay(3000);
  CheckError();

  Serial.print("Connected to service provider!\n\n");
}
```

```

void Senddata(float x){
  gprsSerial.println("AT+CIPSTART=\\"TCP\\",\\"api.thingspeak.com\\",\\"80\\""); //start up the connection
  delay(6000);
  CheckError();

  gprsSerial.println("AT+CIPSEND"); //begin send data to remote server
  delay(4000);
  CheckError();

  String str="GET https://api.thingspeak.com/update?api_key=ZEF3DQ3PMS3G7LKO&field1=" + String(x,1);
  Serial.println(str);
  gprsSerial.println(str); //begin send data to remote server
  delay(10000);
  CheckError();

  gprsSerial.println((char)26); //sending
  delay(5000); //waitting for reply, important! the time is base on the condition of internet
  gprsSerial.println();

  CheckError();

  gprsSerial.println("AT+CIPSHUT"); //close the connection
  delay(100);
  CheckError();
}

```

Figure C-3: Coding for the SIM900a GPRS/GSM module


```

void CheckSD() {

    myFile = SD.open("templ.txt");

    if (myFile) {
        String list1 = myFile.readStringUntil('\n');
        if (list1 == NULL) {
            Serial.print("SD card is empty\n");
            float TEMP = findtemp();
            Senddata(TEMP);
        }
        else {
            Serial.print("SD card has |data\n");
            float Temp = list1.toFloat();
            Senddata(Temp);

            // DATA may be lost if its not sent!!!!!!!!!

            SD.remove("templ.txt");
            Serial.print("Temperature.txt is cleared\n");
            myFile = SD.open("templ.txt");
            myFile.close();
            tempsaved = false;

        }
        myFile.close();

    }
    else {
        Serial.println("error reading temperature file\n\n");
    }

    Serial.print("SD card checked and data sent!\n");

}

```

Figure C-4: Coding for the SD card module

```

void CheckError(){
  int number_of_bytes = gprsSerial.available();
  char response[number_of_bytes];

  for(int i=0; i<number_of_bytes ; i++){
    response[i] = (char)gprsSerial.read();
  }

  myFile = SD.open("response.txt", FILE_WRITE);

  // if the file opened okay, write to it:
  if (myFile) {
    myFile.print(response);
  }
  // close the file:
  myFile.close();

  myFile = SD.open("response.txt");
  if (myFile) {
    // read from the file until there's nothing else in it:
    while (myFile.available()) {
      String list = myFile.readStringUntil('\n');
      delay(2000);
      Serial.print(list);
      Serial.print('\n');

      if (list=="ERROR\r"){
        Serial.print("ERROR!\nSaving temperature to SD card\n");
        sensors.requestTemperatures(); // Send the command to get temperature readings
        /*****/
        Serial.print("Temperature is: ");
        Serial.print(sensors.getTempCByIndex(0)); // Why "byIndex"?
        Serial.print("\n\n");

        // You can have more than one DS18B20 on the same bus.
        // 0 refers to the first IC on the wire
        delay(1000);
        temp = sensors.getTempCByIndex(0);
        saveTemp();
      }
      else if(list=="SEND OK\r"){
        Serial.print("Successful Upload!\n");
        sent = false;
      }
    }
    // close the file:
    myFile.close();

  }
  else {
    // if the file didn't open, print an error:
    Serial.println("error opening response file");
  }
  // open the file. note that only one file can be open at a time,
  // so you have to close this one before opening another.
  SD.remove("response.txt");

  delay(5000);
}

```

Figure C-5: Coding for checking error

```

float findtemp(){
    Serial.print(" Requesting temperatures...");
    sensors.requestTemperatures(); // Send the command to get temperature readings
    Serial.println("DONE");
    /*****/
    Serial.print("Temperature is: ");
    Serial.print(sensors.getTempCByIndex(0)); // Why "byIndex"?
    Serial.print("\r");
    // You can have more than one DS18B20 on the same bus.
    // 0 refers to the first IC on the wire
    delay(1000);
    float t = sensors.getTempCByIndex(0);
    /*****/
    return t;
}

void saveTemp(){

    myFile = SD.open("temp1.txt", FILE_WRITE);
    delay(3000);

    if (myFile){
        if (tempsaved ==false){
            myFile.print(temp);
            myFile.print("\n");
            myFile.close();
            tempsaved =true;
        }
    }
    else{
        Serial.println("error opening temperature file");
    }
}

```

Figure C-6: Coding for the temperature sensor

```

void Going_To_sleep(){
    sleep_enable();//Enabling sleep mode
    attachInterrupt(5, wakeUp, LOW);//attaching a interrupt to pin d18
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);//Setting the sleep mode, in our case full sleep
    digitalWrite(LED_BUILTIN,LOW);//turning LED off
    time_t t;// creates temp time variable
    t=RTC.get(); //gets current time from rtc
    Serial.println("Sleep Time: "+String(hour(t))+" "+String(minute(t))+" "+String(second(t)));//prints time stamp on serial monitor
    delay(1000); //wait a second to allow the led to be turned off before going to sleep
    sleep_cpu();//activating sleep mode

    Serial.println("just woke up!");//next line of code executed after the interrupt
    digitalWrite(LED_BUILTIN,HIGH);//turning LED on
    t=RTC.get();
    Serial.println("WakeUp Time: "+String(hour(t))+" "+String(minute(t))+" "+String(second(t)));//Prints time stamp
    //Set New Alarm
    RTC.setAlarm(ALM1_MATCH_HOURS , 0, 0, hour(t)+time_interval, 0);
    //for testing
    //RTC.setAlarm(ALM1_MATCH_MINUTES , 0, minute(t)+time_interval, 0, 0);

    // clear the alarm flag
    RTC.alarm(ALARM_1);

}

void wakeUp(){
    Serial.println("Interrupt Fired");//Print message to serial monitor
    sleep_disable();//Disable sleep mode
    detachInterrupt(5); //Removes the interrupt from pin 2;
}

```

Figure C-7: Coding for the sleep mode of the microcontroller