Department of Computer Science & Engineering
The University of Texas at Arlington

Detailed Design Specification
CSE 4317: Senior Design II
Summer 2021

The Brews
Bluetooth Hydrometer

Jesus Adrian Guerra
Jorge Avila
Calvin Mata
Doungpakanh Keomaxay-Hampf

Revision History

Revision	Date	Author(s)	Description
0.1	6.19.2021	JAG,DPKH,JA,CRM	document creation
0.2	6.27.2021	JAG,DPKH,JA,CRM	Version 1
0.3	8.8.2021	JAG,DPKH,CRM	Final draft with updated logo and scripts added to final layer

CONTENTS

List of Figures

List of Tables

1 INTRODUCTION

The Bluetooth hydrometer is a system that allows remote monitoring of temperature and specific gravity of beer during the fermentation process. Home brewers will be able to check both temperature and specific gravity(SG) at any point during the brewing process by using their smart phones running an app or a web server on a Raspberry Pi. The Bluetooth hydrometer can be sanitized and placed into a fermentation vessel. This would allow a home brewer to take the specific gravity and temperature of their brew without having to open their fermenter to take a sample which could potentially expose their beer to contamination [1]. The device would send the temperature and SG data to the brewer's phone running an app or a web server. By using this data the home brewer would then be able to monitor the fermentation process enabling them to know when they should change the temperature of their brew.

Home brewers who love to ferment and create their own variety of beers at home are the intended audience of the Bluetooth hydrometer. Experienced home brewers trying to increase the yield of their beers by removing the need of taking samples of their beer during the fermentation process, would buy this product if it were to be made available publicly or commercially. The product is designed for home brewers but can be bought by anyone wishing to try and become a home brewer.

2 SYSTEM OVERVIEW

Our system architecture is composed of a hydrometer layer, a data acquisition layer, and a data display layer. The hydrometer layer gathers specific gravity as well as temperature data from inside a brewing apparatus during the fermentation process. The device will send this data to the outside by using Bluetooth communication. The components that make up the device include a battery, microcontroller, IMU, temperature sensor, as well as the on board BLE(Bluetooth Low Energy) module. The data acquisition layer will include a communications, computation module, and history logging module. This layer is how the data will be transferred from the Hydrometer and processed with the correct calculations. The communications module will receive data from the Hydrometer layer of the system. This data will be stored in a historical logger for later distribution to the user. The computation module will be communicating with both of these modules and communicating with the Data display layer. The data display layer receives data from computation module to be used by the data provider. The data provider will distribute the data to be displayed to the user and is made up of the data provider, current data, as well as historical data components. Specific gravity and temperature data is taken from the data provider to be displayed as either historical or current data.

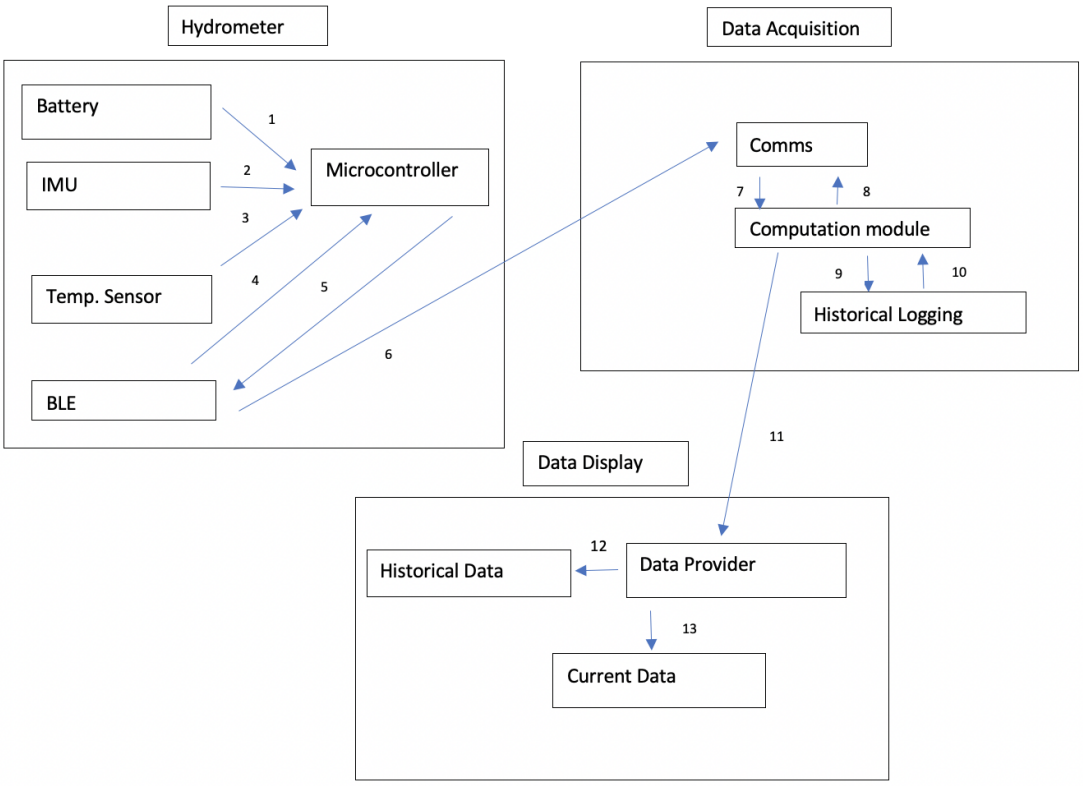

Figure 1: System architecture

3 Hydrometer Layer Subsystems

3.1 Hydrometer Layer Hardware

For our Hydrometer Layer Hardware, we have an Arduino Nano 33 BLE sense. The Arduino is our micro controller that all of our subsystems are receiving commands from for this Layer.

3.2 Hydrometer Layer Operating System

N/A

3.3 Hydrometer Layer Software Dependencies

Our micro controller relies on the following Arduino libraries and drivers:Arduino AVR boards, Arduino MBED OS boards, Adafruit BNO055, Adafruit Unified Sensor, ArduinoBLE, LM35, LM35 Sensor, and Arduino_HTS221.

3.4 Battery

The battery subsystem is the power source for the hardware that makes up the hydrometer. It is a 9 volt lithium battery that connects to the Vin and ground of the Arduino Nano 33 BLE in order to power it.

3.4.1 Battery Hardware

No hardware other than the battery terminals used to connect the 9V lithium battery to the Arduino.

3.4.2 Battery Operating System

No operating systems required by the subsystem.

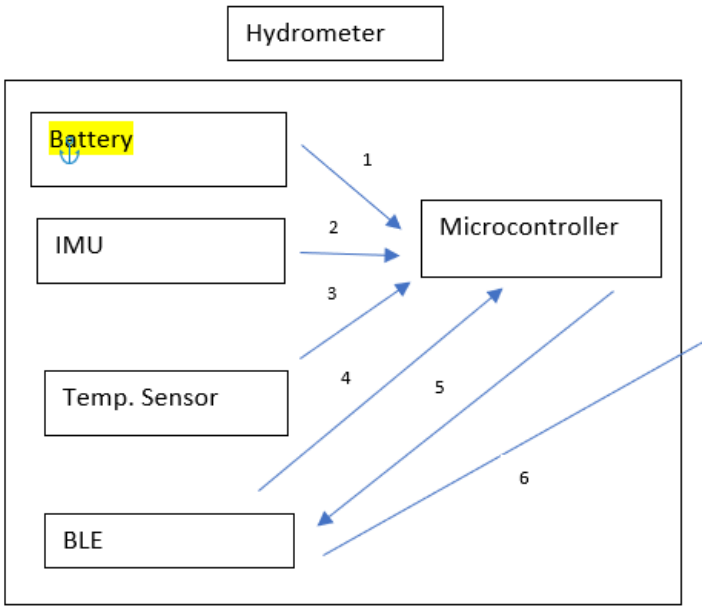

Figure 2: Battery Subsystem Diagram

3.4.3 Battery Software Dependencies

No software dependencies required by the subsystem.

3.4.4 Battery Programming Languages

No programming languages used by the subsystem.

3.4.5 Battery Data Structures

No classes or other data structures for the subsystem.

3.4.6 Battery Data Processing

No algorithms or processing strategies for the subsystem.

3.5 IMU

The Inertial Measurement Unit(IMU) subsystem is what measures the tilt of the hydrometer while inside the brew container during the fermentation process. It outputs voltage levels that can be read by the microcontroller through the use of analog to digital converters. The hardware used is the BNO055 absolute orientation sensor.

3.5.1 IMU Hardware

The IMU is comprised of the BNO055 absolute orientation sensor. The BNO055 is a system in package (SIP), integrating a triaxial 14-bit accelerometer, a triaxial 16-bit gyroscope, and a triaxial geomagnetic sensor.

3.5.2 IMU Operating System

No operating systems required by the subsystem.

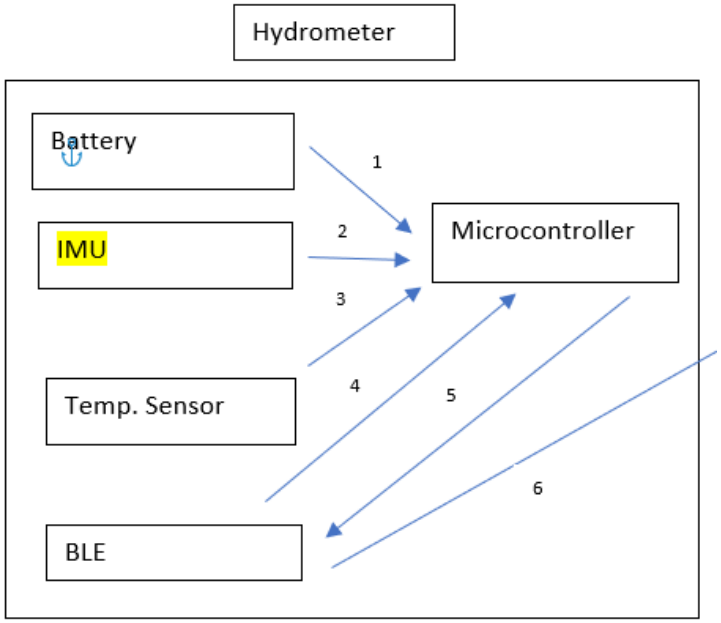

Figure 3: IMU Subsystem Diagram

3.5.3 IMU Software Dependencies

The subsystem uses the library Adafruit BNO055 version 1.4.3 which is designed specifically to work on with the Adafruit BNO055 Breakout, and is based on Adafruits Unified Sensor library.

3.5.4 IMU Programming Languages

The subsystem uses a script on Arduino IDE to get IMU data from the BNO055. The subsystem uses the programming language C++ because that is the language used in the Arduino IDE which is the main text editing program used for Arduino programming.

3.5.5 IMU Data Structures

The IMU has an SDA and and SCL port that is a UART receiver and transmitter respectively. These ports use 12C protocol. To transmit a data byte in one sequence, the sequence begins with a start signal from the master, followed by a 7 bit slave address and a write bit. The slave sends an acknowledge bit and waits for the 8 bit data byte. After the slave signals that it has received the data byte, the master terminates the writing protocol with a stop signal. To receive one or multiple data bytes, the master sends a start signal to the slave and sends the register address to be read. When the slave acknowledges the signal, the master sends another start signal with the slave address and a read bit. The master releases the slave bus and waits for the bits to be read out from the slave. An acknowledged bit from the master stops the slave from reading and allows the master to send a stop signal and terminate the transmission.

void loop(void)

/* Get a new sensor event */
sensors$_e vent_t event$;
$bno.getEvent(event)$;

```
    /* Display the floating point data */
Serial.print("X: ");
Serial.print(event.orientation.x, 4);
Serial.print(": ");
Serial.print(event.orientation.y, 4);
Serial.print(": ");
Serial.print(event.orientation.z, 4);

    /* New line for the next sample */
// read all the sensor values
Serial.println();
```

3.5.6 IMU Data Processing

The raw data being collected from the IMU will be stored and processed by the Raspberry Pi. We are receiving x,y, and z values from the orientation, rotation, linear, magnitude and acceleration of the IMU. We then need to calculate specific gravity using these values. We are still researching what algorithms/ equations we will use to calculate the specific gravity of our brew.

3.6 Temperature Sensor

The temperature sensor subsystem is what measures the temperature of the brew by outputting analog voltage. It is a piece of onboard hardware on the Arduino Nano 33 BLE.

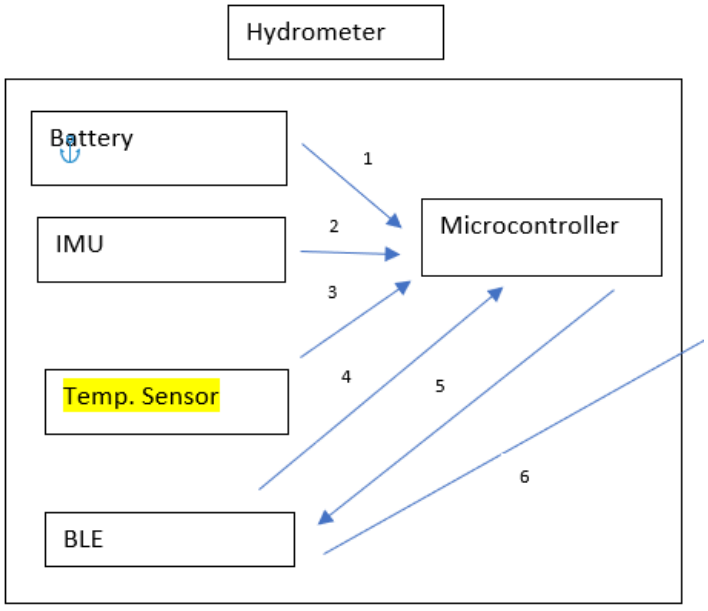

Figure 4: Temperature Sensor Subsystem Diagram

3.6.1 Temperature Sensor Hardware

The subsystem is comprised of the HTS221 which is a piece of onboard hardware on the Arduino Nano 33 BLE. The HTS221 is an ultra compact sensor for relative humidity and temperature.

3.6.2 Temperature Sensor Operating System

No operating systems required by the subsystem.

3.6.3 Temperature Sensor Software Dependencies

The subsystem uses the library Arduino HTS221 version 1.0 on the Arduino IDE which allows for reading of both temperature and humidity on the Arduino 33 BLE Sense.

3.6.4 Temperature Sensor Programming Languages

The subsystem uses a script on Arduino IDE to get temperature data from the HTS221. The subsystem uses the programming language C++ because that is the language used in the Arduino IDE which is the main text editing program used for Arduino programming.

3.6.5 Temperature Sensor Data Structures

The temperature and humidity data can be accessed through a 12C/SPI 3-wire interface for direct communication with a micro controller. The communication protocol between the temperature and the micro controller is similar to the protocol of the IMU. To begin, a START condition is created by sending a a HIGH to LOW on the Serial Data Line (SDA) where the Serial Clock Line (SCL) is HIGH. After this signal is sent by the master, the bus is now busy. The next byte of data transmitted is 7 bits that contain the address of the slave. The 8th bit tells whether the master is receiving or transmitting data.The addressed is compared to the slaves address so that it knows whether it is being communicated with by the master. The data is transmitted and read in byte format which contained 8 bits. There is no limit to the number of bytes transmitted per transfer. When the transmission/ receiving is done, the master can abort the communication by sending a STOP condition. This is done by creating a LOW to HIGH condition on the SDA while the SCL is HIGH.

```
float temperature = HTS.readTemperature();
float farhenheittemp = ((temperature*(1.8)) + 32);
float humidity = HTS.readHumidity();

    // print each of the sensor values
Serial.print("Temperature in celsius = ");
Serial.print(temperature);
Serial.println(" Â°C");

    Serial.print("Temperature in farhenheit = ");
Serial.print(farhenheittemp);
Serial.println(" Â°F");

    Serial.print("Humidity = ");
Serial.print(humidity);
Serial.println("
    // print an empty line
Serial.println();
```

Serial.println("");

3.6.6 Temperature Sensor Data Processing

The capacitive imbalance detected by the temperature sensor is converted into an analog voltage signal. An analog to digital converter is used to generate the digital output returned through a serial interface. The conversion makes it easily accessible to a micro controller that would want to use the data for its own processing. We are implementing an algorithm in our script to take the output digital voltage and convert it to Fahrenheit and Celsius temperatures.

3.7 Microcontroller

The microcontroller subsystem will be powered by current provided by the battery and will receive data from the IMU, temperature sensor, and BLE module. This data will be processed and sent back to the BLE module for processing. The microcontroller hardware is made up of the Arduino Nano 33 BLE Sense.

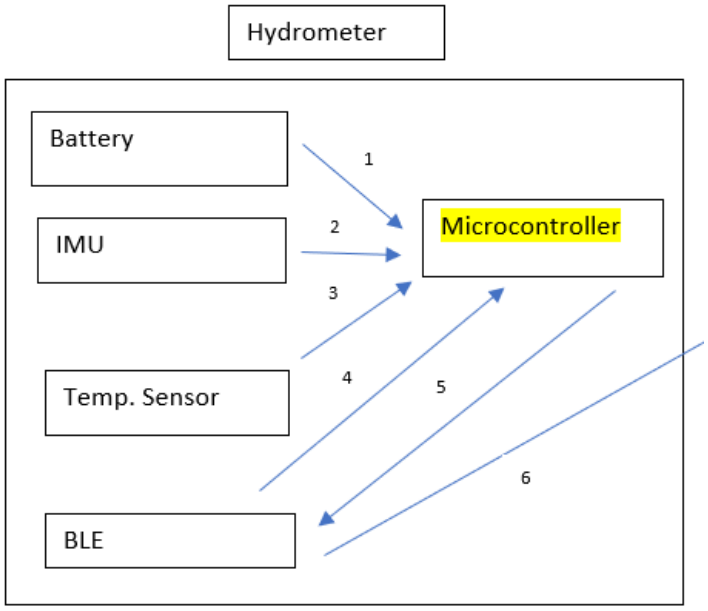

Figure 5: Microcontroller Subsystem Diagram

3.7.1 Microcontroller Hardware

The Arduino Nano 33 BLE Sense is the hardware for the subsystem. It is a cost effective solution for people seeking to have Bluetooth Low Energy connectivity in their projects and houses humidity and temperature sensors, barometric sensor, microphone, light color, and light intensity sensors.

3.7.2 Microcontroller Operating System

No operating systems required by the subsystem.

3.7.3 Microcontroller Software Dependencies

The subsystem requires the user to have the Arduino Mbed OS boards version 2.2 installed on the Arduino IDE in order to compile scripts on Arduino IDE using the Arduino Nano BLE Sense.

3.7.4 Microcontroller Programming Languages

The subsystem uses a script on Arduino IDE to receive temperature data and IMU data. The subsystem uses the programming language C++ because that is the language used in the Arduino IDE which is the main text editing program used for Arduino programming.

3.7.5 Microcontroller Data Structures

The data structures the micro controller uses to transmit/receive data to the HTS221 temperature sensor and the Adafruit BNO055 IMU is described in previous sections for the respective hardware parts.

3.7.6 Microcontroller Data Processing

There is no data processed on the micro controller.

3.8 BLE

The BLE subsystem is a piece of onboard hardware on the Arduino Nano 33 BLE sense. This subsystem requests temperature and specific gravity data from the microcontroller and processes it to be sent to the data acquisition layer.

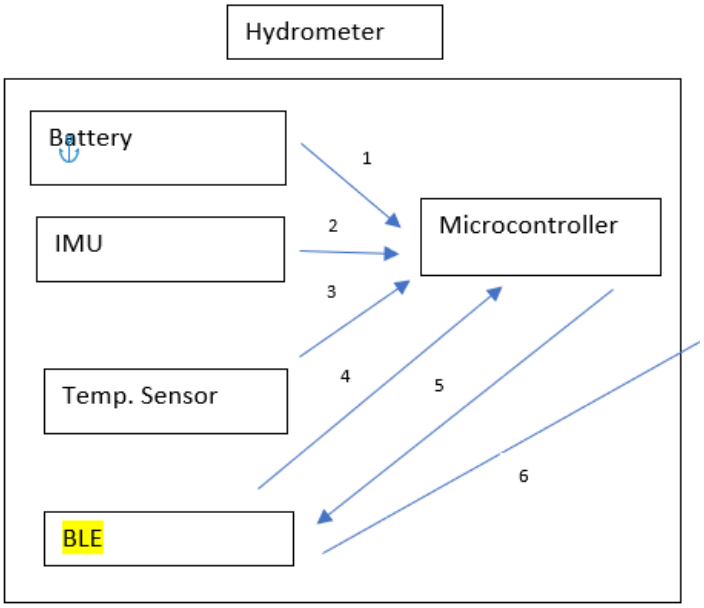

Figure 6: BLE Subsystem Diagram

3.8.1 BLE Hardware

The BLE is a communications chipset on the Arduino Nano 33 BLE Sense that can be both a BLE and Bluetooth client and host device.

3.8.2 BLE Operating System

No operating systems required by the subsystem.

3.8.3 BLE Software Dependencies

This subsystem uses the Arduino BLE library on the Arduino IDE which allows for the creation of BLE services and characteristics as well as BLE advertising.

3.8.4 BLE Programming Languages

The subsystem uses a script on Arduino IDE to request and send data from the microcontroller. The subsystem uses the programming language C++ because that is the language used in the Arduino IDE which is the main text editing program used for Arduino programming.

3.8.5 BLE Data Structures

The packet configuration of the data contains PREAMBLE, ADDRESS, S0, LENGTH, S1, PAYLOAD, and CRC fields. The PREAMBLE is sent first on the least significant bit. The byte order is always the least significant byte first for the ADDRESS and PAYLOAD fields. The ADDRESS fields is always the first least significant bit transmitted. The S0, LENGTH, S1, and PAYLOAD fields endianness can be configured via the ENDIAN in PCNF1.

```
    BLE.setLocalName("IMUandTempMonitor");
BLE.setAdvertisedService(IMUandTemp); // add the service UUID
IMUandTemp.addCharacteristic(IMUandTempdata); // add the battery level characteristic
IMUandTemp.addCharacteristic(xdata); // add the battery level characteristic
IMUandTemp.addCharacteristic(ydata); // add the battery level characteristic
IMUandTemp.addCharacteristic(zdata); // add the battery level characteristic
BLE.addService(IMUandTemp); // Add the battery service

    IMUandTempdata.writeValue(oldIMUandTemp); // set initial value for this characteristic
xdata.writeValue(oldxvalue);
ydata.writeValue(oldyvalue);
zdata.writeValue(oldzvalue);

    // start advertising
BLE.advertise();

    Serial.println("Bluetooth device active, waiting for connections...");
```

3.8.6 BLE Data Processing

There is no data processed on the BLE, only transmitted.

4 Data Acquisition Layer Subsystems

4.1 Data Acquisition Hardware

For our Data Acquisition layer hardware we will be using the Raspberry Pi 4. The Raspberry Pi 4 will receive temperature as well as IMU data from the hydrometer layer for processing and historical logging. It comes with Gigabit Ethernet, along with onboard wireless networking and Bluetooth.

4.2 Data Acquisition Operating System

We will be using the Raspberry Pi 4 OS known as Raspbian which is a Debian-based operating system for Raspberry Pi.

4.3 Data Acquisition Software Dependencies

We are currently researching Bluetooth integration between the Arduino and Raspberry Pi so this will be determined at a later date.

4.4 Comms

The comms subsystem receives temperature and IMU data from the BLE of the hydrometer layer. It takes this data for later processing in the computation module in order to compute the specific gravity of the brew.

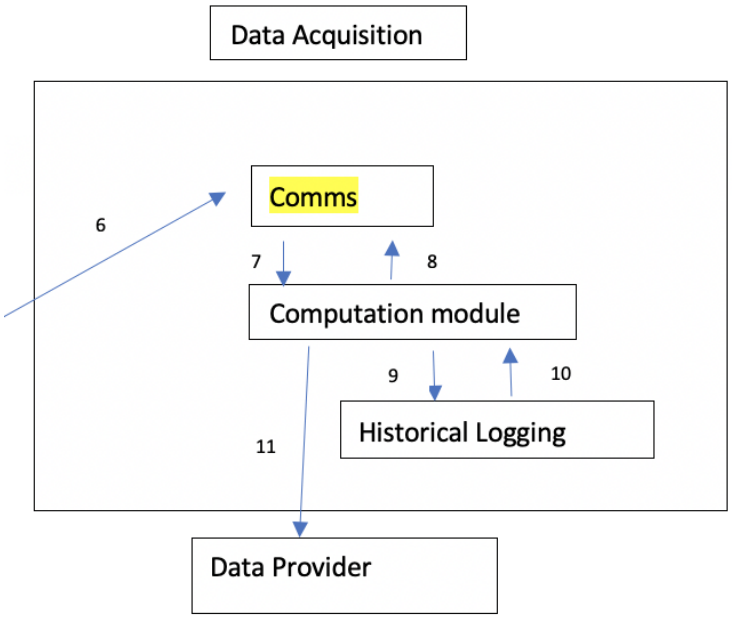

Figure 7: Comms subsystem diagram

4.4.1 Comms Hardware

The comms subsystem will be using the raspberry Pi 4's built in Bluetooth in order to receive data from the hydrometer layer.

4.4.2 Comms Operating System

The comms subsystem will be using the same operating system as the layer.

4.4.3 Comms Software Dependencies

The Bluetooth integration between the Arduino and Raspberry Pi will require the user have both Python 3 or greater installed on their Raspberry Pi 4 as well as bluepy.

4.4.4 Comms Programming Languages

This subsystem will be run using Python programming language.

4.4.5 Comms Data Structures

Bluepy's peripheral class encapsulates a connection to a BLE peripheral. We create a peripheral object by specifying its MAC address and after a connection is made, the services and characteristics being advertised by our device can be discovered for reading and/or writing. A BLE "characteristic" is a short data item that can be read or written. Rather than constructing characteristic objects directly we use the getCharacteristics() method of a connected peripheral object.

```
import sys
import time
import binascii
import codecs

    from argparse import ArgumentParser
from bluepy import btle #linux only

    def main():
args = get_ args()

    print("Connecting...")
nanoble33 = btle.Peripheral(args.mac_address)

    print("Discovering Service...")
_ = nanoble33.services
imuandtempservice = nanoble33.getServiceByUUID("1800")

    print("Discovering Characteristic...")
_ = imuandtempservice.getCharacteristics()
while True:
print("")
read_temperature(imuandtempservice)
read_sg(imuandtempservice)

    def get_args():
arg_parser = ArgumentParser(description="BLE IoT Sensor Demo")
arg_parser.add_argument('mac_address', help="MAC address of device to connect")
args = arg_parser.parse_args()
return args
```

4.4.6 Comms Data Processing

There is no data processed in the comms module, only received and transmitted.

4.5 Computation Module

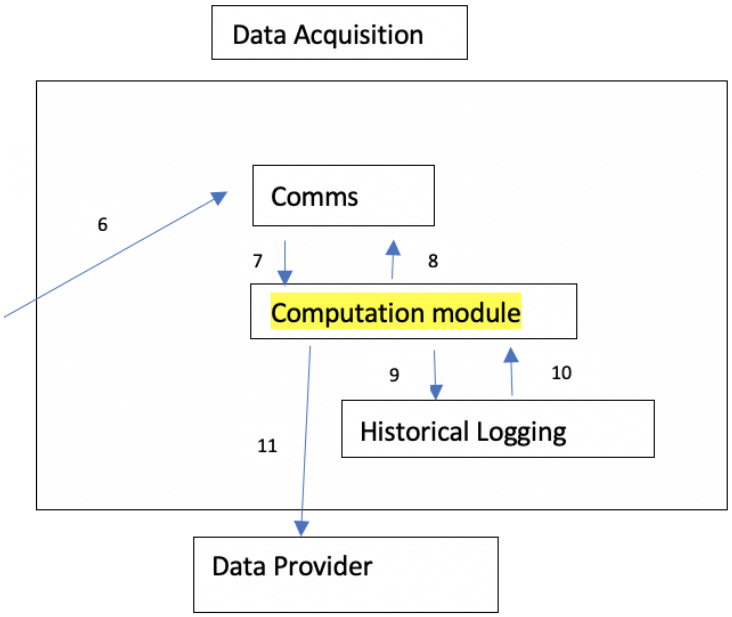

Figure 8: Computation Module subsystem diagram

4.5.1 Computation Module Hardware

The computation module subsystem will be using the same hardware as the layer.

4.5.2 Computation Module Operating System

The computation module subsystem will be using the same operating system as the layer.

4.5.3 Computation Module Software Dependencies

The Bluetooth integration between the Arduino and Raspberry Pi will require the user have both Python 3 or greater installed on their Raspberry Pi 4 as well as bluepy.

4.5.4 Computation Module Programming Languages

This subsystem will be run using Python programming language.

4.5.5 Computation Module Data Structures

The raw data sent to this module is a hexstring of int values, as a series of bytes, in little endian byte order. This data is taken as advertised by our device for conversion.

```
def byte_array_to_int(value):
value = bytearray(value)
value = int.from_bytes(value, byteorder="little")
return value

    def byte_array_to_char(value):
value = value.decode("utf-8")
```

```
return value

    def decimal_exponent_two(value):
return value / 100 # e.g., 2350 -> 23.5

    def celsius_to_fahrenheit(value):
return (value * 100 * 1.8) + 32 # return value*1.0

    def read_temperature(service):
temperature_char = service.getCharacteristics("2A6E")[0]
temperature = temperature_char.read()
temperature = byte_array_to_int(temperature)
temperature = decimal_exponent_two(temperature)
temperature = celsius_to_fahrenheit(temperature)
print(f"Temperature: round(temperature, 2)F")
with open("/home/pi/PycharmProjects/imuandtempfile.csv", "a") as log:
log.write("0,".format(str(temperature)))

    def read_sg(service):
sg_char = service.getCharacteristics("2B0D")[0]
sg = sg_char.read()
sg = byte_array_to_int(sg)
sg = decimal_exponent_two(sg)
print(f"SG: round(sg, 2)")
with open("/home/pi/PycharmProjects/imuandtempfile.csv", "a") as log:
log.write("0,".format(str(sg)))
```

4.5.6 Computation Module Data Processing

The raw data of temperature and specific gravity values are converted from bytes to bytearray to int. For legibility, these integer values are then given two decimal places. As an example 2350 turns into 23.5. As an added measure the temperature data is converted from Celsius to Fahrenheit.

4.6 Historical Logging

4.6.1 Historical Logging Hardware

The Historical Logger will use the Raspberry Pi to store and communicate data to different layers/ sub modules.

4.6.2 Historical Logging Operating System

N/A

4.6.3 Historical Logging Software Dependencies

The Historical Logger will use Firebase database and Angular JS.

4.6.4 Historical Logging Programming Languages

AngularJS and Firebase use Html, Typescript, and CSS as built in tool/programming languages to transfer data from server to database and vice versa.

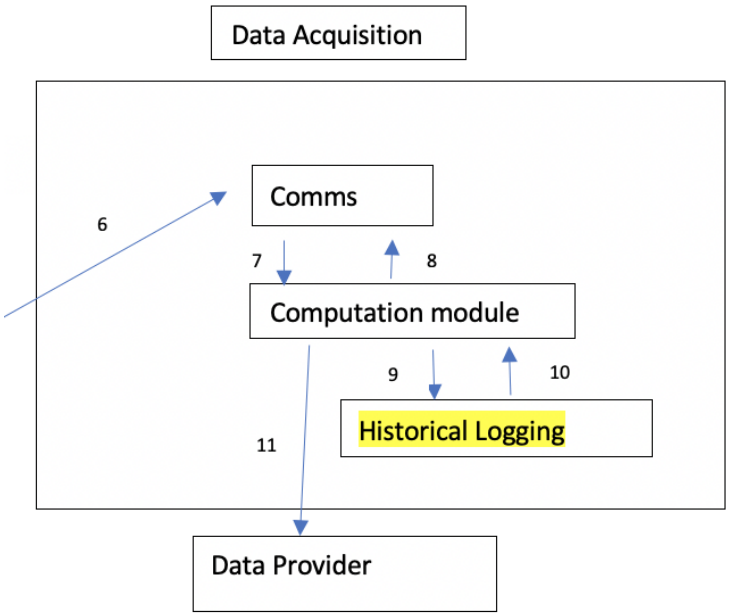

Figure 9: Historical Logging subsystem diagram

4.6.5 Historical Logging Data Structures

The Firebase database will parse a comma separated file of raw data and store it in their respective tables in the database. Angular JS will act as an API to connect the data base with the web app. Angular JS will create an HTML file that will define the structure of our template being transmitted. Defining a structure using Angular JS will let us effectively communicate different components of our data that we are displaying to the web app.

4.6.6 Historical Logging Data Processing

There is no data processed in this sub module.

5 Data Display Layer Subsystems

The Data Display is the layer that will define how things are displayed from the Logger/Database back onto the web page using web services and tools such as API's that will quickly show data to the user. This layer is the most abstract to the user that aids in hiding the back end nuisance.

5.1 Data Display Hardware

The layer hardware that will be involved in the data display will be the bluetooth component that will send raw data to the pi, the pi will process the data and send that processed data to the MySQL database and that data will be displayed on the website that will be connected to the server through AngularJS.

5.2 Data Display Operating System

The data will be displayed from the raw data given from the Pi. So the OS used will be the Raspbian OS, but can also be displayed using Ubuntu Mate, Snappy Ubuntu Core, the Kodi-based mediacenters OSMC and LibreElec, the non-Linux based Risc OS, or the Windows 10 IoT Core.

5.3 Data Display Software Dependencies

We will use Angular to display the software onto a monitor for the users to read given data.

5.4 Data Provider(API)

This section is illustrating more in depth on how the Data Provider is constructed. This diagram shows how we inquire data from the Data Acquisition layer, then brought over to the Data Display layer to leverage on behalf of the user. We want to display that data to the user in a form of a web service to the data provider

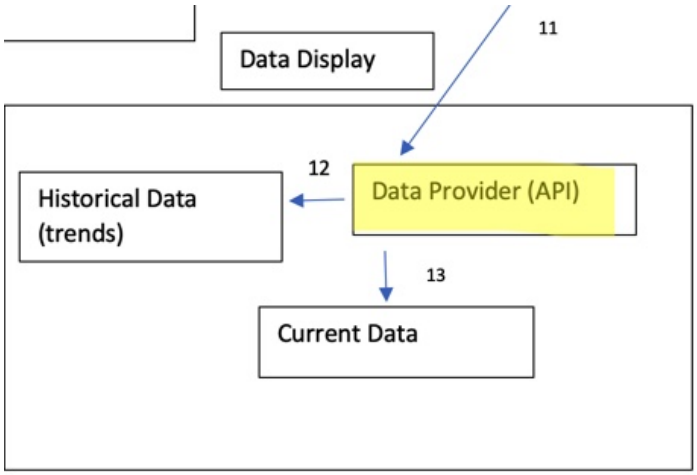

Figure 10: Data Provider(API) subsystem diagram

5.4.1 Data Provider(API) Hardware

The user's phone will be required to run the web app.

5.4.2 Data Provider(API) Operating System

The Pi can run the official Raspbian OS, Ubuntu Mate, Snappy Ubuntu Core, the Kodi-based media centers OSMC and LibreElec, the non-Linux based Risc OS (one for fans of 1990s Acorn computers).

It can also run Windows 10 IoT Core, which is very different to the desktop version of Windows, as mentioned below.

5.4.3 Data Provider(API) Software Dependencies

We as a team will be using Angular, which is a web based framework and that uses typescript language to display data that we will be leveraging to our use.

5.4.4 Data Provider(API) Programming Languages

AngularJS and Arduino based C language.

5.4.5 Data Provider(API) Data Structures

Data being transmitted from the Bluetooth module will be received in a "raw" state, we then have to find the equation that best suits our circuit board by trail and error.

5.4.6 Data Provider(API) Data Processing

We have to implement an algorithm for calculating the specific gravity of the liquid for which our device will be submerged. In doing so, this algorithm will be what gives us our "raw" data to be used for the rest of the data processing.

5.5 Current Data

This section illustrates how Current data will be displayed on it's own web page to the user viewing the data. Current data is the data that is currently being displayed continuously through a module set up on a web page that is brought in through data provider. An example of this is through the fermentation process, the specific gravity of the liquid at the exact moment.

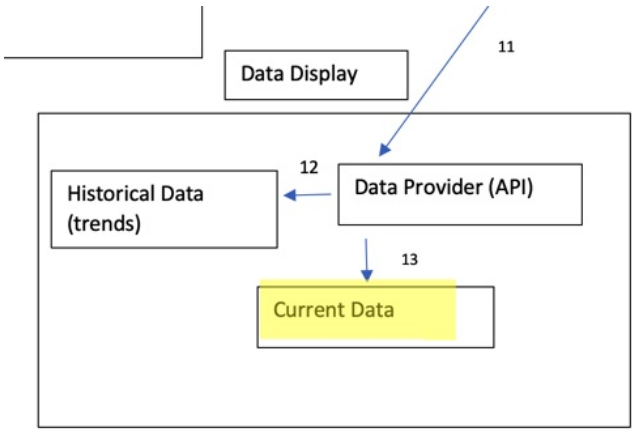

Figure 11: Current Data subsystem diagram

5.5.1 Current Data Hardware

Monitor that will connect to the PI to read data.

5.5.2 Current Data Operating System

A monitor that will connected to the PI.

5.5.3 Current Data Software Dependencies

We as a team will be using Angular, which is a web based framework and that uses typescript language to display data that we will be leveraging to our use.

5.5.4 Current Data Programming Languages

AngularJS and Arduino based C language.

5.5.5 Current Data Data Structures

Data being transmitted from the Bluetooth module will be received in a "raw" state, we then have to find the equation that best suits our circuit board by trail and error.

Our HTML code reads as follows:

```
<div class="container-fluid">
<div style="color: blue;">
<h1>title</h1>
<h3>description</h3>
</div>
<div style="width: 300px;">
<form (ngSubmit)="onSubmit()">
<div class="form-group">
<label for="item">item</label>
<input type="file" class = "form-control" fileImportInput name="File Upload" id="csvFileUpload" required [(ngModel)] = "itemValue" (change)="fileChangeListener(event)"accept = ".csv"/ >< /div >
<div class="btn-group">
<button type="submit" class="btn btn-success">submit</button>
</div>
</form>
</div>
<h2>Content From Firebase</h2>
<div class="col-md-3">

</div>
</div>

<div class="csv-file-chooser-section">

</div>

<div class="csv-result-table">

<table>
<thead>
<tr>
<th></th>
<th>Temperature</th>
<th>Specific Gravity</th>
</tr>
</thead>
<tbody>
```

```
<tr *ngFor="let csvData of csvRecords;let i = index;">

    <td>i+1</td>
<td>
<span>csvData.Temperature</span>
</td>
<td>
<span>csvData.Specific_Gravity</span>

    </tr>
</tbody>
</table>

    </div>
```

5.5.6 Current Data Data Processing

We have to implement an algorithm for calculating the specific gravity of the liquid for which our device will be submerged. In doing so, this algorithm will be what gives us our "raw" data to be used for the rest of the data processing.

5.6 Historical Data

This section is illustrating more the Historical Data layer is used for all trending data that the IMU sends back to user ona web page for their consumption of data analysis.

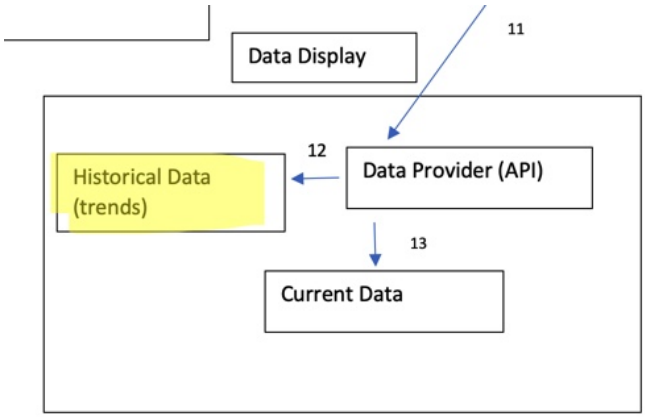

Figure 12: Historical Data subsystem diagram

5.6.1 Historical Data Hardware

Monitor that will connect to the PI to read data.

5.6.2 Historical Data Operating System

The Pi can run the official Raspbian OS, Ubuntu Mate, Snappy Ubuntu Core, the Kodi-based media centers OSMC and LibreElec, the non-Linux based Risc OS (one for fans of 1990s Acorn computers).

It can also run Windows 10 IoT Core, which is very different to the desktop version of Windows, as mentioned below.

5.6.3 Historical Data Software Dependencies

We as a team will be using Angular, which is a web based framework and that uses typescript language to display data that we will be leveraging to our use.

5.6.4 Historical Data Programming Languages

AngularJS and Arduino based C language.

5.6.5 Historical Data Data Structures

Data being transmitted from the Bluetooth module will be received in a "raw" state, we then have to find the equation that best suits our circuit board by trail and error.

5.6.6 Historical Data Data Processing

We have to implement an algorithm for calculating the specific gravity of the liquid for which our device will be submerged. In doing so, this algorithm will be what gives us our "raw" data to be used for the rest of the data processing.

6 Appendix A

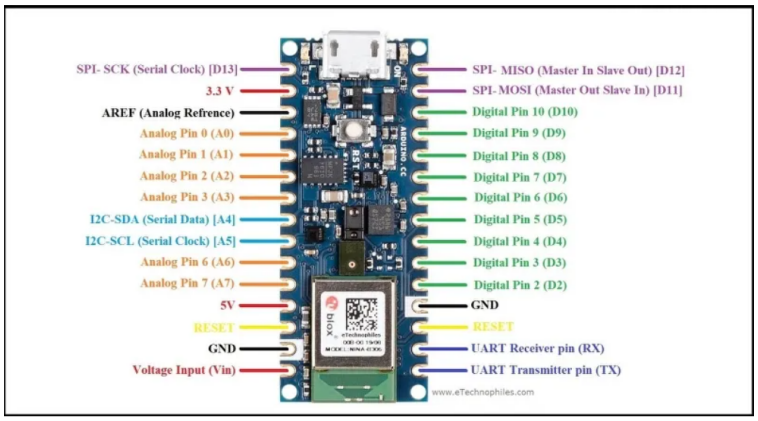

Figure 13: Arduino Nano 33 BLE Sense Pinout

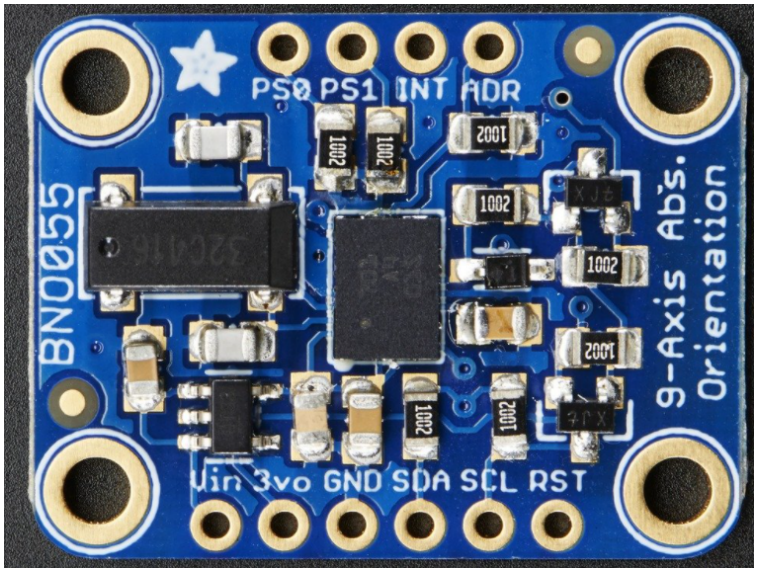

Figure 14: BNO055 Pinout

References

[1] Bill Downs. Meet the Microbes Who Cause Homebrew Contamination, 2018.