Mobile Water Purity Tester (MWPT)

Adrian Sucahyo, Vivek Anandh, Ruiyu Zhang, Nickolas Zhao

6.4.2021

SKYLINE HIGH SCHOOL

UTAH

Project Overview and Goals

Abstract

In order to address water quality access issues, we created an Arduino-based water quality tester that measures pH, electrical conductivity, temperature, and turbidity. Because the prototype is small, portable, and inexpensive, our project can expand access to water quality testing. In this current iteration, the device is able to record data from the probes onto an SD card, present data through an interface, and connect to a companion app to show relative levels of safety.

Problem Statement

One of the major problems that Utah has been facing in recent years has been water contamination. The most common culprits that lead to contamination have terrible side effects and are invisible to the naked eye such as Heavy Metals, Chlorine, and MTBE. Poor water quality leads to numerous health problems that in some cases, are life-threatening. The United States Environmental Protection Agency states that:

> "Chemical exposure through drinking water can lead to a variety of short- and long-term health effects. Exposure to high doses of chemicals can lead to skin discoloration or more severe problems such as nervous system or organ damage and developmental or reproductive effects. Exposure to lower doses over long periods of time can lead to chronic, longer-term conditions such as cancer."

The most concerning issue is that the people who are most affected by this come from marginalized or traditionally colored communities and with Utah's already scarce water resources and rapidly increasing population, this issue continues to exacerbate.

A prominent point that comes up frequently in the discussion of water quality is inequity in water purity awareness. In its current state, the process of testing water quality is seen by most as complicated and tiresome. The solution is to create a device that is not only fast and simple to use, but also one that is affordable and can be purchased by everyone regardless of economic status. This solution would have the primary benefit of providing quality information about water purity to individuals but also the additional benefits of identifying widespread water contamination issues and being safe for the environment due to its reusability.

Research

To get a comprehensive sense of traditional water purity testing, we started by talking to one of our clients who owns multiple homes and apartment complexes. He informed us that checking water quality in homes is usually a laborious process. We discussed the two primary ways of checking water quality and their respective advantages and disadvantages. The first option is to hire a professional to properly test the water quality of a home, however, it is very expensive due to labor costs and it is also not accessible to many people. The second option is for those who can't hire professionals; They can either buy an expensive high-quality single-use kit that takes two weeks to deliver data from a lab or, they can buy a cheap sensor that isn't accurate and doesn't present data in an intelligible way. Our goal was to take the advantages from both sides and create the perfect medium.

Through our several meetings with potential clients during the design process, we noticed that there were three main features/characteristics that were most requested by our clients.

1. **Affordability** - The biggest concern brought up by our clients was in terms of cost. They wanted the device to be below $50 while still maintaining all the key components of a high-quality water purity tester. In the current market, high-quality water purity testers are expensive around $100, and often require you to send them to a lab. The other end of the spectrum comprises of cheap sensors that lack measurement characteristics and understandable data.
2. **Reusability and Logging Capability -** Our clients wanted the ability to check their water quality on a regular basis and track any changes over time. The devices they had previously used were described as inconvenient because they only measured one variable and the owner had to manually record data to analyze changes.
3. **Ease of Use** - Another major concern for our clients was understandability and convenience. They wanted the data presented in a model that allowed them to see exactly what was being measured by the sensors and on a scale that shows what the safe amount should be. The product also had to be small and easy to set up so that customers can check on a daily basis without much hassle.

Design Process

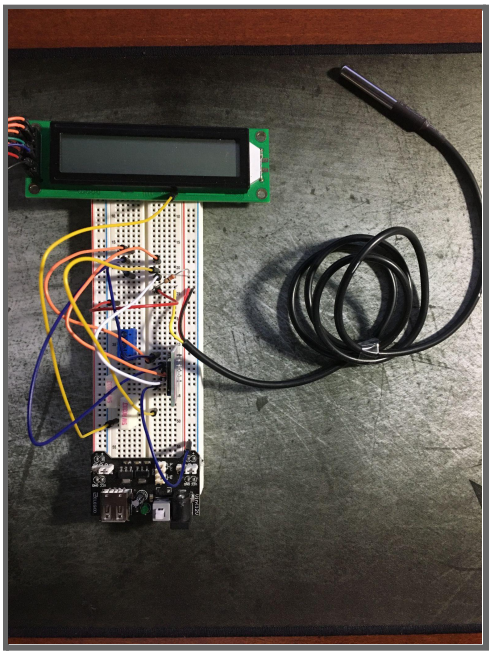
Prototypes

Prototyping was integral to the development of the project, especially for making successive improvements with each revision. The goal of the prototypes was to learn what needs to be changed in the successive revisions and discover ways to improve project integrity as well as usability of the device. Testing and feedback from the user was a major factor that determined the path we took, along with our own decisions to

make the project better as a whole.

First Prototype

The initial prototype was mostly used as a basic interface to learn about the sensors. A 2 x 16 LCD was used as a basic display for the values collected from the sensors, and the Arduino MEGA was chosen in order to allow for easy prototyping. The entire construction was built on a breadboard for easy accessibility to all the ports and pins of the Arduino module and the sensors. The sensors that were tested were the turbidity probe, the conductivity probe, the pH probe, and the temperature probe. Since this was the first prototype, the main goal was to learn the operations of these modules and make sure that they operate as expected. Calibration of these modules was also taken into account because the ability to alter hardware configuration is easiest in the first stage.



Pictured above is the initial prototype with a single sensor connected to the device. Each sensor was attached to the device individually in order to simplify the process, but they were eventually consolidated into a single build at the beginning of the successive

prototype.

Major challenges that were encountered were mostly due to the sensor calibration. The provided manufacturing data and user manuals for the modules were insufficient and during testing, there were some errors with the turbidity, pH, and conductivity sensor. This was suspected to be caused by some of the manufacturing tolerances of the devices. This issue was apparent when tested with distilled water, and the turbidity, pH, and conductivity sensors gave unexpected results. Distilled water was used to calibrate all of the sensors, as it is representative of pure H2O. Knowing this, the conductivity and turbidity sensors should have given values of zero and a pH of 7, yet they returned nonzero or incorrect values. To resolve this issue, a programmed offset was added to account for this error. This was able to account for this error, but a more flexible solution was used in future iterations to solve this.

Another smaller issue we ran into was improper communication between the Arduino and the temperature sensor. The unique protocol of the temperature sensor was not being detected by the Arduino and the integrated library. The communication protocol that was used is known as OneWire, where all the data is transmitted through a single wire. This error was due to a hardware issue with the data wire in the probe, requiring a pullup resistor to be properly connected to the Arduino, as the temperature probe was unable to pull the line to the supply voltage or a “high” digital state, interfering with data transmission. This issue was easily addressed due to the fact that it was built on a breadboard at this stage.

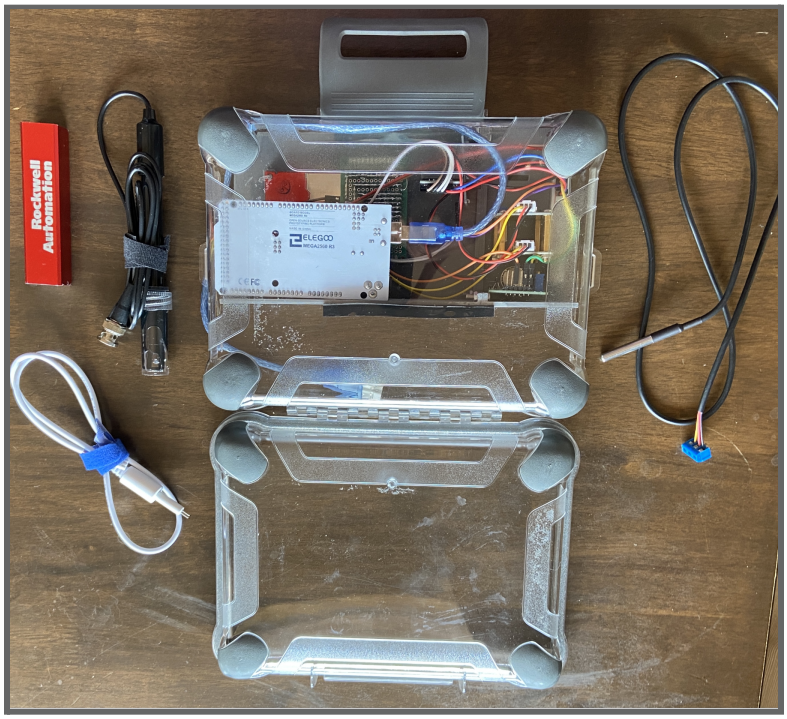
Some feedback that we got from our clients was to make it easier to understand the data that was being displayed on the LCD. As the data in this stage was displayed in

numerical form, it was difficult for the user to understand what was being given, as they had no context for the values. Our idea was to implement a system that would visually display data for the user. This was kept in mind for future improvements.

Some other improvements addressed the unreliability of the protoboard and the inconvenience of the device usage. The loose wires were an annoyance during development and for the user because the connections were not held together well. To cause further problems, the power to the device was through a direct wall connection. These were all priorities for the next iteration, along with the feedback from our clients.

Second Prototype

The second prototype was an improvement on the first with the addition of the portability factor. This was achieved by soldering the modules to perfboards (PCBs) and putting the entire construction within a waterproof box. This way, the device was now portable and could be safely placed in wet environments. The probes themselves could also be stored within the waterproof container, thus allowing everything to be contained within the enclosure. Furthermore, there was an integration of a power bank in order to provide a stable portable power source.



Pictured above is the second prototype and within the enclosure, it is possible to see the internal wiring. As some of the modules are already designed with breakout pins, they could not be soldered onto the PCB. This prototype was useful in determining an appropriate size for the device, and to give our clients an idea of how large the device would end up being.

Another major addition to this prototype was the integration of an SD card module in order to have data logging capabilities. This was integrated into the design through an SPI interface. This interface is advantageous since it is able to quickly transmit data while only using a few wires. Furthermore, the module itself is able to convert the 5V supply voltage to the 3.3V logic levels required to interface with the SD card. In future iterations where the modules can be consolidated to a single PCB, a 3.3V logic level converter chip will be required. However, this is a consideration for the future and is not

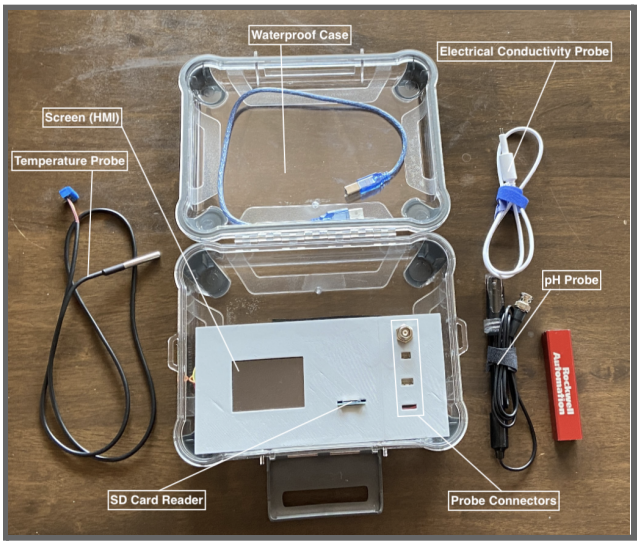
a concern on this page.

One issue that we faced was a software-related issue. The SD card module caused lots of issues as the device would randomly disconnect during operation and interrupt the data collection process. This issue was resolved by lowering the communication speed from the default 40 MHz frequency to a lower 30 MHz clock. This was due to a hardware issue where the wires that connected the Arduino to the SD module were not able to handle the higher transmission rates, as the wire runs were long and the wires were not electrically insulated from external noise.

The strength of this prototype was most definitely in the ability to log data. The data that was collected from the sensors are stored in .csv files, which are compatible with common spreadsheet applications. This makes it easier for users to perform further data analysis on the many different water samples that were tested. To further improve the logging system, the files that are created should be named with the time and date that they were started, and each line added to the .csv file will also have a timestamp for longer-term data logging.

Third Prototype

The third iteration had the addition of an RTC module, a higher resolution interface, and the integration of a custom 3D printed bracket to hold the modules together. The 3D printed bracket was used in order to build the device in the waterproof case. This allowed for the probes to be easily accessed, each of the probe modules to be properly mounted together, and wiring to be enclosed within the case.



The LCD module was chosen to help with the user's ability to interpret and visualize the data. A 2.8" ILI9341 TFT LCD (Thin Film Transistor Liquid Crystal Display) with an SPI interface was chosen, as the SPI communication protocol was already in use with the SD card module. This caused a challenge to our design as there was an incompatibility between the two of the libraries. The SD card and the TFT LCD library were interfering with each other, as a modified TFT LCD library was in use. The standard library did not have the speed required for quickly displaying the data that was gathered, so a modified and optimized open-source library was used. The issue arose because the two SPI devices shared the same MISO, MOSI, SCK data lines, and when the libraries would pull the CS data lines for each device at the wrong times, it would cause interference and failure of both devices. The solution to this problem was the integration of a "software SPI" where certain pins act as the hardware SPI bus of the Arduino. This solution was appropriate and the interference was no longer an issue.

The RTC (real-time clock) module was integrated easily as it did not require the SPI bus, but rather an I2C bus. Since this was a single device, there was no interference and

the operations of it were normal. The data that was pulled from the device were the date and the time, and they were integrated into the data logging functionality of the SD card.

In this stage, many of the major hardware components were finalized, and the further steps were mostly software-based. Some key problems that had yet to be addressed were the lack of a more adaptable version of the sensor calibration, along with the improvements to user accessibility. These are addressed in the next section.

Fourth Prototype

The current prototype is in its fourth iteration, with all the features presented before, along with some major improvements to the software. The hardware changes were solely an improvement to the battery capacity from a 500mAh commercial power bank to an 8.4V 5000 mAh NiMH battery. Another improvement was the integration of the TFT LCD's touch screen capabilities to help the user's experience with the device.

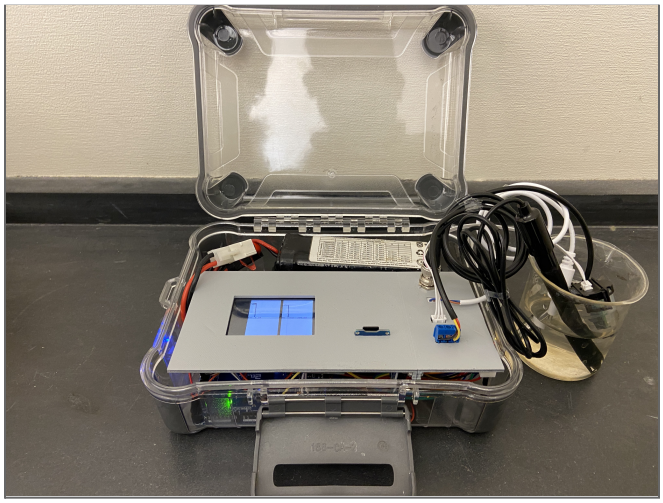
The probe calibration issue from the initial prototype was resolved through a software modification. Three of the four probes use an analog interface with the Arduino module, and the resulting voltages that are fed back to the Arduino are a "fraction" between the 0V ground reference and the 5V supply voltage. In certain cases where the 5V supply voltage is not 5.0 V exactly, the returned analog voltages from the probes will not be accurate. In order to counteract this, an offset constant needed to be added to the recorded voltage. This was determined by using a known constant voltage reference, and seeing where it fell when recorded by the Arduino. Knowing this, it was possible to determine a constant that would be used to adjust the readings of the probes by the Arduino. The voltage that was chosen was 3.3V, as the Arduino's onboard regulator remained isolated from the fluctuating supply voltage. This proved effective in

correcting the recorded values from the probes on the analog inputs.

To further counteract the voltage offset and drift, the buck converter from the 8.2 V battery was adjusted to provide as accurate voltage as possible to the required 5.0 V. The device used is known as a buck converter, which can efficiently step down the voltage accurately. The only issue with this is that there are unexpected voltage drops when the buck converter is heavily loaded.

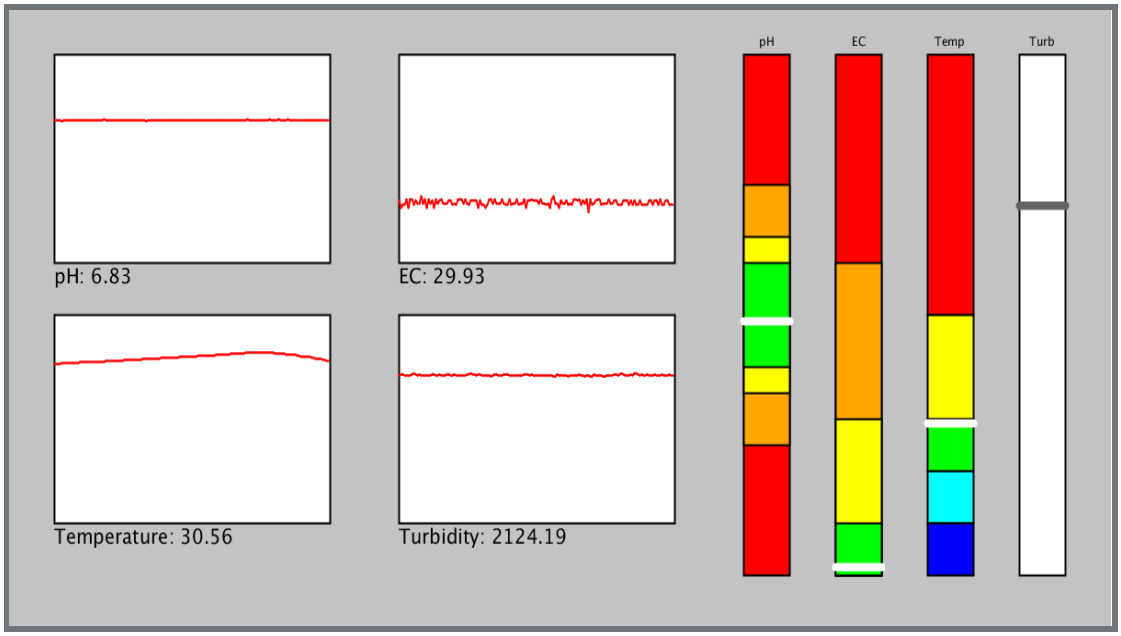
The touchscreen implementation was used to initialize the recordings on the SD card, along with initialization of the SD card slot in the case that the card was to disconnect unexpectedly.

This finalizes the hardware elements, and thus the block diagram includes all of the major components, along with how they are connected together. The block diagram is present in Appendix A. A picture of the current prototype is included below, and it is possible to see all of the connected probes, the touch display, and the upgraded battery. However, internally it is not possible to see the new buck converter.



The software implementation introduced a desktop companion application. The

device can communicate directly with a computer through a serial interface (USB), and the data is transmitted every second. This data is then gradually added to the graphs that are present in the application's window. One of the major features of this companion application is the ability to more easily display the quality or the risk of the water. This is done by the use of multicolored bar gauges, with colors indicating the corresponding risk level. The application window is shown below.



The feedback we received supported this addition as it helped to provide some context to the actual values being recorded by each of the sensors and allowed less informed clients to understand the data being presented.

All of the current code and software is provided in Appendix A for both the companion application and the Arduino hardware. The organization of the code for the Arduino is organized into separate sections in order to aid readability. All of the separate sections/files were compiled to a single binary file to be flashed to the Arduino microcontroller.

Testing Procedures

Data Collection

The data was collected by testing our probes against a commercially available probe. This was done in order to verify that the methods used in the program functioned correctly.

Data Analysis

Statistical Test:

We are attempting to determine the accuracy of our prototype compared to traditional ph sensors which give the readings on a connected interface.

We can perform a two-tailed t-test for a significant difference between the mean ph values given by separate samples of ph values from both the independent ph sensor and our Arduino prototype. This is to determine if our programming and prototype are functional.

Procedure:

- A random concentration of Sodium hydroxide (NaOH) or hydrochloric acid (HCl) will be placed in one beaker and the independent sensor will read the solution's pH value.
- The same concentration of acid/base will be added to a solution and its pH will be measured by the Arduino.
- This process will be repeated 31 times, yielding 31 pairs of beakers with the same concentration of either an acid or base. Because one of each pair is assigned a sensor (Arduino/indep.), each beaker can be considered an experimental unit.

- The readings will be recorded for each pair, then the average pH of the ones assigned for the independent sensor and the Arduino will be calculated. The p-value for the test will be calculated and a conclusion will be drawn.

Conditions/Statement of Hypotheses:

$H_0 = \mu_1 - \mu_2 = 0$

$H_A = \mu_1 - \mu_2 \neq 0$

$\alpha = .05$

- Normal/Large Counts: Because the amount of beakers assigned to the arduino is 31, and the same is true for the indep. sensor, we can say that the large counts condition is met.
- Random: We randomly assigned beakers to either get a reading from the indep. sensor or the Arduino.

Calculations (refer to Appendix B. for raw data):

To find t-val:

$\bar{x}_1 - \bar{x}_2 = 8.550053238 - 8.204836615 = .345216623$

$\sqrt{\frac{s_1^2}{n} + \frac{s_2^2}{n}} = 1.06067282$

.345216623/1.06067282 = 0.32546947229 = t

P val/2: tcdf(0.32546947229, 1e99, 30) = .3735424929

Pval = .7471

Results:

We fail to reject the null hypothesis because there is no difference between the

accuracy of the indep. sensor and the Arduino at the .05 level, since p is > than .05. This means that the Arduino programming and our prototype are functional.

Further Steps

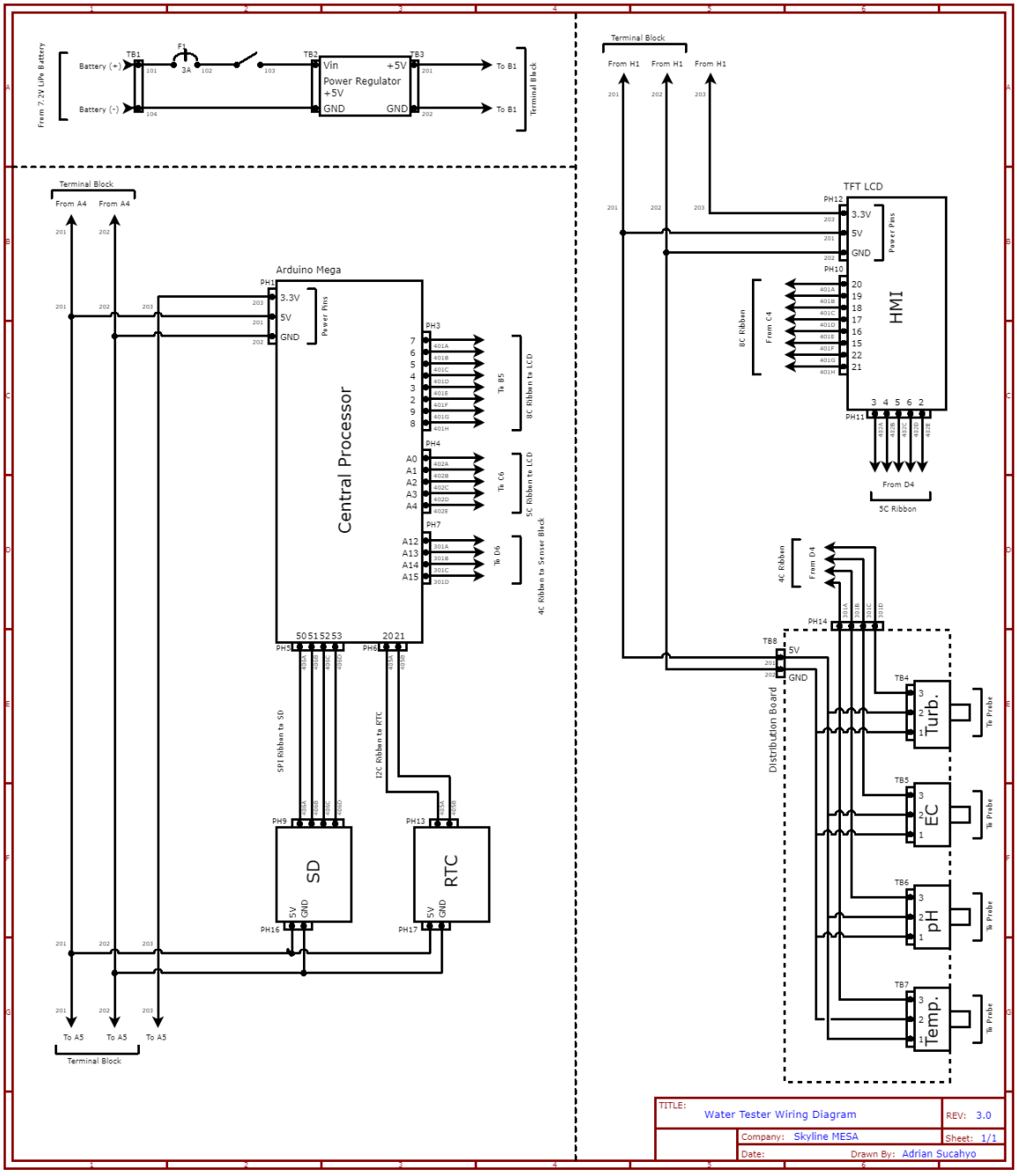
As the device has been shown to be accurate through testing, it is appropriate to conclude that we can advance to further developed designs, such as the integration of the device with a fully designed PCB that can be quickly manufactured at a low cost, especially at higher volumes. The simplicity of the current design is further improved by the fact that the sensors used have known working subsystems that can be easily implemented through the direct integration of the chips used.

From the user after the latest revision, one feature that they suggested was a way to add a wireless communication device to interface with the device. A mobile application has been developed to do this, although it is not possible to communicate in the current state since we haven't been able to acquire a Bluetooth module. However, this modification would not be difficult to implement, as there are many points to connect to the Arduino, along with the structure in the code to easily implement the device.

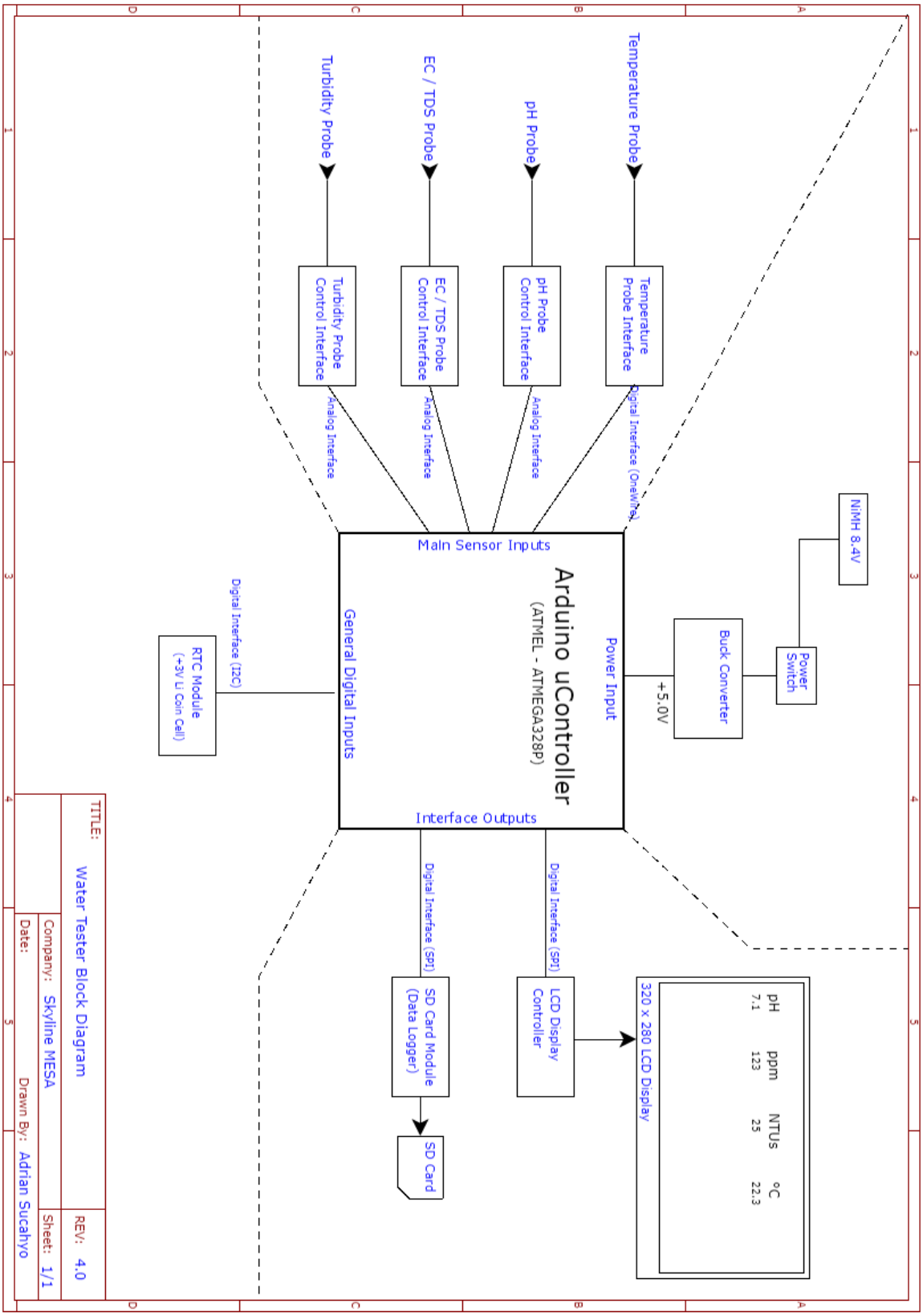
A secondary design also arose to build off of the current revision. Although the device is freestanding, it can also be integrated into more permanent systems. The system in mind is to integrate it with a water filtration system, where the data is logged onto the SD card module or streamed to a companion application. The app will allow the user to monitor their water quality at all times and be notified if safety level thresholds are exceeded.

Appendix A

Schematics



Block Diagrams



Software

Main

```
/*-----------------------------------------------------------------------
------------------------
   Arduino MESA Project 2020-2021

   Water Monitor Project
   - Monitors Various Conditions in Water for determination of:
    - Safety
    - Quality
    - Conditions

   Integrated Modules:
   - SD Card Mod.
   - RTC Mod.
   - pH Probe
   - EC Probe
   - Temperature Probe
   - Turbidity Probe


-----------------------------------------------------------------------
----------------------
*/

#include <arduino-timer.h>

auto timer = timer_create_default();

int MenuState = 0;
#define bootPage 0
#define metersPage 1

const int arrLengths = 60;
float pHVal, ECVal, TempVal, TurbVal;
float pHVals[arrLengths];
float ECVals[arrLengths];
float TempVals[arrLengths];
float TurbVals[arrLengths];

float maxpH, maxEC, maxTemp, maxTurb = 1;

float analogConstant;

bool recording = false;
```

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

  setupEC(); //Setup all devices
  setuppH();
  setupTemp();
  setupTurb();

  setupRTC();
  setupLCD();

  delay(500);
  setupSD();

  TFTdisplay(); //Boot screen to allow delay to occur
  delay(1000);

  timer.every(500, readSensors);
  timer.every(1000, TFTdisplay);
  timer.every(50, readTouch);

  delay(2000);
  MenuState = 1;
  clearDisplay();
  TFTdisplay();
}

void loop() {
  timer.tick(); //Internal Timer Setup
  switch (MenuState) {
    case bootPage: //Introductory Page
      break;
    case metersPage: //Main Meters Page
      break;
  }
}

bool readSensors() { //Read all sensor data
  //Gather Data
  analogConstant = (3.37 / 5.0) / (float(analogRead(A15)) /
float(analogRead(A14)));
  pHVal = calculatepH();
  ECVal = generateTDS();
  TempVal = generateTemperatureC();
  TurbVal = calculateTurbidity();

  //Determine Max Values
```

```
if (pHVal > maxpH) {
  maxpH = 5 * ceil(pHVal / 5);
}
if (ECVal > maxEC) {
  maxEC = 100 * ceil(ECVal / 100);
}
if (TempVal > maxTemp) {
  maxTemp = 10 * ceil(TempVal / 10);
}
if (TurbVal > maxTurb) {
  maxTurb = 100 * ceil(TurbVal / 100);
}

//Limit the readings values
constrain(pHVal, 0, maxpH);
constrain(ECVal, 0, maxEC);
constrain(TempVal, 0, maxTemp);
constrain(TurbVal, 0, maxTurb);

//Shift data array
for (int i = arrLengths - 1; i > 0; i--) {
  pHVals[i] = pHVals[i - 1];
  ECVals[i] = ECVals[i - 1];
  TempVals[i] = TempVals[i - 1];
  TurbVals[i] = TurbVals[i - 1];
}
pHVals[0] = pHVal;
ECVals[0] = ECVal;
TempVals[0] = TempVal;
TurbVals[0] = TurbVal;

//Check for new high reading
for (int i = 0; i < arrLengths; i++) {
  float maxpHTemp;
  float maxECTemp;
  float maxTempTemp;
  float maxTurbTemp;

  if (pHVals[i] > maxpHTemp) {
    maxpHTemp = pHVals[i];
  }
  if (ECVals[i] > maxECTemp) {
    maxECTemp = ECVals[i];
  }
  if (TempVals[i] > maxTempTemp) {
    maxTempTemp = TempVals[i];
  }
```

```
    if (TurbVals[i] > maxTurbTemp) {
      maxTurbTemp = TurbVals[i];
    }

    maxpH = constrain(5 * ceil(maxpHTemp / 5), 0, 15);
    maxEC = 100 * ceil(maxECTemp / 100);
    maxTemp = 10 * ceil(maxTempTemp / 10);
    maxTurb = 100 * ceil(maxTurbTemp / 100);
  }

  //Log Data if applicable
  if (recording) {
    logData();
  }

  //Send though serial port if applicable
  if (Serial) {
    Serial.print(logTime() + ",");
    Serial.print(String(pHVal) + ",");
    Serial.print(String(ECVal) + ",");
    Serial.print(String(TempVal) + ",");
    Serial.print(String(TurbVal) + '\n');
  }
  return true;
}

String dataString() {
  String data = logTime() + "," + String(pHVal) + "," +
String(ECVal) + "," + String(TempVal) + "," + String(TurbVal);
  return data;
}
```

EC Probe

```
/*-------------------------------------------------------------------
-----------------------
  Manager for the Electrical Conductivity Probe

  Sensor Connected to Pin A1 of Arduino


-------------------------------------------------------------------
-----------------------*/
#define SensorPin A1

float offset = 0;
```

```
void setupEC() {
}

float readECRaw() {
  float rawValue;
  rawValue = float(analogRead(SensorPin)) * analogConstant;
  return rawValue;
}

float generateEC() {
  float voltage;
  voltage = readECRaw() * analogConstant * (5.0 / 1024.0);
  return voltage;
}

float generateTDS() {
  float TDS;
  float voltage = generateEC();
  TDS = ((133.42 * (pow(voltage, 3)) - (255.86 * sq(voltage)) +
(857.39 * voltage)) * 0.5) - offset;
  return TDS;//constrain(TDS, 0.0, 5000.0);
}
```

pH Probe

```
/*-------------------------------------------------------------------
------------------------
  Manager for the pH Probe

  Sensor Connected to Pin A2 of Arduino


-------------------------------------------------------------------
-----------------------*/
#define SensorPin A2
#define Offset 0.00
#define samplingInterval 20
#define numReadings 10

float pHArray[numReadings];

float calibration_value = 21.34;
int phval = 0;
unsigned long int avgval;
int buffer_arr[10], temp;
```

```
float ph_act;


void setuppH() {
}

float averageArr() {
  float avgVal = 0;
  for (int i = 0; i < numReadings; i++) {
    avgVal += pHArray[i];
  }
  avgVal = avgVal / numReadings;
  return avgVal;
}

float readpH() {
  float voltage;
  for (int i = 0; i < numReadings; i++) {
    pHArray[i] = float(analogRead(SensorPin)) * analogConstant;
    delay(samplingInterval);
  }
  voltage = averageArr() * (5.0 / 1024);
  return voltage;
}

float calculatepH() {
  //  float pHValue;
  //  pHValue = (3.5 * readpH()) + Offset;
  //  return pHValue;

  for (int i = 0; i < 10; i++)
  {
    buffer_arr[i] = float(analogRead(SensorPin)) * analogConstant;
    delay(5);
  }
  for (int i = 0; i < 9; i++)
  {
    for (int j = i + 1; j < 10; j++)
    {
      if (buffer_arr[i] > buffer_arr[j])
      {
        temp = buffer_arr[i];
        buffer_arr[i] = buffer_arr[j];
        buffer_arr[j] = temp;
      }
    }
  }
```

```
  avgval = 0;
  for (int i = 2; i < 8; i++)
    avgval += buffer_arr[i];
  float volt = (float)avgval * 5.0 / 1024 / 6;
  volt -= 0.22;
  //Serial.println(volt);
  ph_act = -5.70 * volt + calibration_value;
  //Serial.println(ph_act);
  return ph_act;
}
```

RTC Module

```
/*-------------------------------------------------------------------
------------------------
  Manager for the Real Time Clock Module (DS3231)

  Module connected to dedicated SCL and SDA pins


-------------------------------------------------------------------
------------------------*/

#include <RTClib.h>

bool RTCPresent = false;
bool powerLoss = false;

RTC_DS3231 rtc;


void setupRTC() {
  if (!rtc.begin()) {
    RTCPresent = false;
    //Serial.println("Couldn't find RTC");
  } else {
    RTCPresent = true;
  }

  if (!rtc.lostPower()) {
    powerLoss = false;
  } else {
    powerLoss = true;
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  }
}
```

```
void setRTC(int yyyy, int mo, int d, int h, int mi) {
  rtc.adjust(DateTime(yyyy, mo, d, h, mi , 0));
}

String logTime() {
  DateTime now = rtc.now();
  String logPhrase = "";
  logPhrase = (String(now.month()) + "/") + (String(now.day()) +
"/") + (String(now.year()) + "  ") + (String(now.hour()) + ":") +
(String(now.minute()) + ":") + String(now.second());
  return logPhrase;
}

String fileTimestamp() {
  DateTime now = rtc.now();
  String Timestamp = "";
  Timestamp = String(now.month()) + "." + String(now.day()) + "." +
String(now.year()) + "-" + String(now.hour()) + "." +
String(now.minute());
  return Timestamp;
}
```

SD Module

```
/*-----------------------------------------------------------------
------------------------
  Manager for the SD Module

  Module Connected to SPI Pin Header


-----------------------------------------------------------------
-----------------------*/
#include <SdFat.h>

const uint8_t SOFT_MISO_PIN = 28;
const uint8_t SOFT_MOSI_PIN = 30;
const uint8_t SOFT_SCK_PIN  = 32;
const uint8_t SD_CHIP_SELECT_PIN = 34;

SdFatSoftSpi<SOFT_MISO_PIN, SOFT_MOSI_PIN, SOFT_SCK_PIN> sd;

SdFile loggingFile;

bool SDCardPresent = false;
String currentFileName = "";
```

```
void setupSD() {
  pinMode(SOFT_MISO_PIN, INPUT_PULLUP);
  if (!sd.begin(SD_CHIP_SELECT_PIN)) {
    SDCardPresent = false;
    //sd.initErrorHalt();
    //Serial.println("NoCard");
  } else {
    SDCardPresent = true;
    //Serial.println("Card");
  }
}

void logData() {
  if (loggingFile.open(currentFileName.c_str(), O_RDWR | O_CREAT))
{
    if (recording) {
      loggingFile.print(dataString());
    }
    loggingFile.println();
  } else {
    recording = false;
    SDCardPresent = false;
    //Serial.println("errorSD");
  }
  loggingFile.close();
}

void startLogFile() {
  currentFileName = fileTimestamp() + ".csv";
  createFile();
}

void closeLogFile() {
  loggingFile.close();
  //Serial.println("sd close");
}

void createFile() {
  if (loggingFile.open(currentFileName.c_str(), O_RDWR | O_CREAT))
{
    loggingFile.println(F("Time,pH,Electrical
Conductivity,Temperature,Turbidity"));
    recording = true;
  } else {
        recording = false;
        SDCardPresent = false;
  }
```

```
  loggingFile.close();
}
```

TFT Display Manager

```
/*-----------------------------------------------------------------
------------------------
  Manager for the TFT Display and Touch Functionality

  Display Connected to Pins defined in the Main File


-----------------------------------------------------------------
-----------------------*/
//TFT
#include "TFT_ILI9341.h"
#include "TFT_Touch.h"
#include <SPI.h>

#define DOUT 44
#define DIN  45
#define DCS  46
#define DCLK 47
//CS   -  42
//RST  -  48
//DC   -  49
//MOSI -  51
//SCK  -  52

TFT_ILI9341 tft = TFT_ILI9341();
TFT_Touch touch = TFT_Touch(DCS, DCLK, DIN, DOUT);

int color = TFT_WHITE;
#define TFT_GREY 0x5AEB
unsigned int colors[10] = {TFT_RED, TFT_GREEN, TFT_BLUE, TFT_BLACK,
TFT_CYAN, TFT_YELLOW, TFT_WHITE, TFT_MAGENTA, TFT_BLACK,
TFT_BLACK};

bool fileSplash = false;
int splashcount = 0;

void setupLCD() {
  tft.init();
  touch.setCal(3611, 671, 3397, 698, 320, 240, 1);

  tft.setTextColor(TFT_BLACK);
  tft.setRotation(1);
```

```
  touch.setRotation(1);
  tft.setTextSize(1);
  tft.fillScreen(TFT_BLACK);

  //tft.setTextColor(TFT_GREEN);
}

bool readTouch() {
  int X_RawData;
  int Y_RawData;
  int X_Coord;
  int Y_Coord;

  if (touch.Pressed()) {
    X_Coord = touch.X();
    Y_Coord = touch.Y();
    //Serial.println("XY");
    //Serial.println(X_Coord);
    //Serial.println(Y_Coord);
    switch (MenuState) {
      case bootPage:
        break;
      case metersPage:
        if (X_Coord >= 0 && X_Coord <= 100 && Y_Coord >= 0 &&
Y_Coord <= 150) {
          if (recording) {
            recording = false;
            closeLogFile();
          } else if (!recording) {
            if (SDCardPresent) {
              startLogFile();
            } else if (!SDCardPresent) {

            }
            fileSplash = true;
            splashcount = 3;
          }
        }
        if (X_Coord >= 0 && X_Coord <= 100 && Y_Coord >= 150 &&
Y_Coord <= 220) {
          if (SDCardPresent) {
            if (recording) {
              closeLogFile();
              fileSplash = true;
              splashcount = 3;
            }
          } else {
```

```
            setupSD();
          }
        }
        delay(250);
        break;
    }
  }
  return true;
}

void clearDisplay() {
  tft.fillScreen(TFT_BLACK);
}

bool TFTdisplay() {

  switch (MenuState) {
    case bootPage:
      tft.fillRect(10, 10, 300, 60, TFT_WHITE);
      tft.fillRect(20, 50, 250, 4, TFT_BLUE);
      tft.fillRect(10, 90, 250, 40, TFT_WHITE);
      tft.setTextSize(2);
      tft.setCursor(15, 15);
      tft.print("ARDUINO WATER TESTER");
      tft.setTextSize(2);
      tft.setCursor(15, 95);
      tft.print(logTime());
      break;
    case metersPage:
      if (recording == true) {
        tft.fillRect(10, 10, 20, 150, TFT_RED);
        tft.setTextSize(1);
        tft.setRotation(2);
        tft.setCursor(20, 295);
        tft.print("RECORDING");
        tft.setCursor(21, 295);
        tft.print("RECORDING");
        tft.setRotation(1);
      } else {
        tft.fillRect(10, 10, 20, 150, TFT_GREEN);
        tft.setTextSize(1);
        tft.setRotation(2);
        tft.setCursor(20, 295);
        tft.print("Press to Begin Log");
        tft.setCursor(21, 295);
        tft.print("Press to Begin Log");
        tft.setRotation(1);
```

```
    }

    if (SDCardPresent == true) {
      tft.fillRect(10, 170, 20, 50, TFT_GREEN);
      tft.setTextSize(1);
      tft.setRotation(2);
      tft.setCursor(175, 295);
      tft.print("SD GOOD");
      tft.setCursor(176, 295);
      tft.print("SD GOOD");
      tft.setRotation(1);
    } else {
      tft.fillRect(10, 170, 20, 50, TFT_RED);
      tft.setTextSize(1);
      tft.setRotation(2);
      tft.setCursor(175, 295);
      tft.print("NO CARD");
      tft.setCursor(176, 295);
      tft.print("NO CARD");
      tft.setRotation(1);
    }

    if (!fileSplash) {
      tft.fillRect(60, 0, 120, 110, TFT_WHITE);
      tft.fillRect(190, 0, 120, 110, TFT_WHITE);
      tft.fillRect(60, 110, 120, 110, TFT_WHITE);
      tft.fillRect(190, 110, 120, 110, TFT_WHITE);

      tft.drawRect(60, 100, 120, 10, TFT_BLACK);
      tft.drawRect(190, 100, 120, 10, TFT_BLACK);
      tft.drawRect(60, 210, 120, 10, TFT_BLACK);
      tft.drawRect(190, 210, 120, 10, TFT_BLACK);

      tft.setTextSize(1);
      tft.setCursor(62, 101);
      tft.println("pH: " + String(pHVal, 2));
      tft.setCursor(192, 101);
      tft.println("EC (ppm): " + String(ECVal, 2));
      tft.setCursor(62, 211);
      tft.println("Temp (C): " + String(TempVal, 1));
      tft.setCursor(192, 211);
      tft.println("Turb. (NTU): " + String(TurbVal, 0));

      for (int i = 0; i < arrLengths - 1; i++) {
        tft.drawLine(60 + i * 2, ((100 / maxpH) * (maxpH -
pHVals[i])), 60 + (1 + i) * 2, ((100 / maxpH) * (maxpH - pHVals[i +
1])), TFT_BLACK);
```

```
          tft.drawLine(61 + i * 2, ((100 / maxpH) * (maxpH -
pHVals[i])), 61 + (1 + i) * 2, ((100 / maxpH) * (maxpH - pHVals[i +
1])), TFT_GREY);
          tft.drawLine(59 + i * 2, ((100 / maxpH) * (maxpH -
pHVals[i])), 59 + (1 + i) * 2, ((100 / maxpH) * (maxpH - pHVals[i +
1])), TFT_GREY);
        }
        for (int i = 0; i < arrLengths - 1; i++) {
          tft.drawLine(190 + i * 2, ((100 / maxEC) * (maxEC -
ECVals[i])), 190 + (1 + i) * 2, ((100 / maxEC) * (maxEC - ECVals[i
+ 1])), TFT_BLACK);
          tft.drawLine(191 + i * 2, ((100 / maxEC) * (maxEC -
ECVals[i])), 191 + (1 + i) * 2, ((100 / maxEC) * (maxEC - ECVals[i
+ 1])), TFT_GREY);
          tft.drawLine(189 + i * 2, ((100 / maxEC) * (maxEC -
ECVals[i])), 189 + (1 + i) * 2, ((100 / maxEC) * (maxEC - ECVals[i
+ 1])), TFT_GREY);
        }
        for (int i = 0; i < arrLengths - 1; i++) {
          tft.drawLine(60 + i * 2, 110 + ((100 / maxTemp) *
(maxTemp - TempVals[i])), 60 + (1 + i) * 2, 110 + ((100 / maxTemp)
* (maxTemp - TempVals[i + 1])), TFT_BLACK);
          tft.drawLine(61 + i * 2, 110 + ((100 / maxTemp) *
(maxTemp - TempVals[i])), 61 + (1 + i) * 2, 110 + ((100 / maxTemp)
* (maxTemp - TempVals[i + 1])), TFT_GREY);
          tft.drawLine(59 + i * 2, 110 + ((100 / maxTemp) *
(maxTemp - TempVals[i])), 59 + (1 + i) * 2, 110 + ((100 / maxTemp)
* (maxTemp - TempVals[i + 1])), TFT_GREY);
        }
        for (int i = 0; i < arrLengths - 1; i++) {
          tft.drawLine(190 + i * 2, 110 + ((100 / maxTurb) *
(maxTurb - TurbVals[i])), 190 + (1 + i) * 2, 110 + ((100 / maxTurb)
* (maxTurb - TurbVals[i + 1])), TFT_BLACK);
          tft.drawLine(191 + i * 2, 110 + ((100 / maxTurb) *
(maxTurb - TurbVals[i])), 191 + (1 + i) * 2, 110 + ((100 / maxTurb)
* (maxTurb - TurbVals[i + 1])), TFT_GREY);
          tft.drawLine(189 + i * 2, 110 + ((100 / maxTurb) *
(maxTurb - TurbVals[i])), 189 + (1 + i) * 2, 110 + ((100 / maxTurb)
* (maxTurb - TurbVals[i + 1])), TFT_GREY);
        }
      }

      if (fileSplash == true) {
        if (splashcount == 0) {
          fileSplash = false;
          clearDisplay();
        } else {
```

```
          splashcount--;
        }
      }

      if (fileSplash == true) {
        tft.fillRect(80, 40, 200, 100, TFT_WHITE);
        tft.drawRect(80, 40, 200, 100, TFT_BLACK);
        if (SDCardPresent) {
          if (recording) {
            tft.setCursor(85, 45);
            tft.println("File Created. Called:");
            tft.setCursor(85, 60);
            tft.println(currentFileName);
          } else {
            tft.setCursor(85, 45);
            tft.println("Error");
          }
        } else if (!SDCardPresent) {
          tft.setCursor(85, 45);
          tft.println("NO SD CARD INSERTED");
        }
      }
      break;
  }
  return true;

}
```

Temperature Probe

```
/*-----------------------------------------------------------------------
-----------------------
  Manager for the Temperature Probe

  Sensor Connected to Pin 8 of Arduino


-----------------------------------------------------------------------
-----------------------*/
#include <OneWire.h>
#include <DallasTemperature.h>

#define TemperatureBusPin 8
int counter = 0;

OneWire oneWire(TemperatureBusPin);
DallasTemperature sensors(&oneWire);
```

```
void setupTemp() {
  sensors.begin();
  sensors.setWaitForConversion(false);
}

void readTemperatures() {
  if (counter == 0) {
    sensors.requestTemperatures();
  } else if (counter == 2) {
    counter = 0;
  }
  counter++;
}

float generateTemperatureC() {
  if (counter == 0) {
    sensors.requestTemperatures();
    //Serial.println(sensors.getTempCByIndex(0));
    //counter++;
    return sensors.getTempCByIndex(0);
  } else if (counter == 2) {
    counter = 0;
  } else {
    counter++;
    //Serial.println(counter);
    return TempVal;
  }
}

float generateTemperatureF() {
  if (counter == 0) {
    sensors.requestTemperatures();
  } else if (counter == 2) {
    counter = 0;
  }
  counter++;
  return ((sensors.getTempCByIndex(0)) * 9.0 / 5.0) + 32.0;
}
```

Turbidity Probe

```
/*-----------------------------------------------------------------
-----------------------
  Manager for the Turbidity Probe

  Sensor Connected to Pin A0 of Arduino
```

```
------------------------------------------------------------------------
----------------------*/
#define SensorPin A0

float voltage = 0;
float sensorValue = 0;

float NTUs[5];
int counts = 0;

void setupTurb() {
}

float readTurbidity() {
  sensorValue = float(analogRead(SensorPin)) * analogConstant;
  return sensorValue;
}

float calculateTurbidity() {
  float NTU;
  voltage = readTurbidity() * (5.0 / 1024.0);
  //Serial.println(voltage);
  if (voltage < 2.5) {
    NTU = 3000;
  } else {
    NTU = -1120.4 * square(voltage) + 5742.3 * voltage - 4353.8;
  }
  if (counts < 5) {
    counts++;
    for (int i = 0; i < counts - 1; i++) {
      NTUs[i + 1] = NTUs[i];
    }
    NTUs[0] = NTU;
    float total = 0;
    for (int i = 0; i < counts; i++) {
      total += NTUs[i];
    }
    NTU = total / counts;
  } else {
    for (int i = 0; i < 4; i++) {
      NTUs[i + 1] = NTUs[i];
    }
    NTUs[0] = NTU;
    float total = 0;
    for (int i = 0; i < 5; i++) {
```

```
      total += NTUs[i];
    }
    NTU = total / 5;
  }
  return constrain(NTU, 0.0, 3000.0);
}
```

Desktop Companion Application

```
import processing.serial.*;

Serial myPort;  // Create object from Serial class
String input = ",,,,";      // Data received from the serial port
String[] splitInputs = {"", "", "", "", ""};

int arrLengths = 150;
float pHVal, ECVal, TempVal, TurbVal;
float[] pHVals = new float[arrLengths];
float[] ECVals = new float[arrLengths];
float[] TempVals = new float[arrLengths];
float[] TurbVals = new float[arrLengths];

float maxpH, maxEC, maxTemp, maxTurb;


void setup()
{
  size(1200, 600);
  // Initialize the Serial Port
  try {
    myPort = new Serial(this, Serial.list()[1], 9600);
    myPort.bufferUntil('\n');
  }
  catch (Exception e) {
    stop();
  }
}

void draw()
{
  // Draw all data
  background(195);
  drawGraphs();
}
```

```
void drawGraphs() {
  // Draw all line graphs
  stroke(0);
  strokeWeight(2);
  fill(255);
  rect(50, 50, 300, 200);
  rect(50, 300, 300, 200);
  rect(425, 50, 300, 200);
  rect(425, 300, 300, 200);
  stroke(255, 0, 0);
  strokeWeight(2);

  for (int i = 0; i < pHVals.length - 1; i++) {
    line(50 + i * 2, 50 + ((200 / maxpH) * (maxpH - pHVals[i])), 50
+ (1 + i) * 2, 50 + ((200 / maxpH) * (maxpH - pHVals[i + 1])));
  }
  for (int i = 0; i < ECVals.length - 1; i++) {
    line(425 + i * 2, 50 + ((200 / maxEC) * (maxEC - ECVals[i])),
425 + (1 + i) * 2, 50 + ((200 / maxEC) * (maxEC - ECVals[i + 1])));
  }
  for (int i = 0; i < TempVals.length - 1; i++) {
    line(50 + i * 2, 300 + ((200 / maxTemp) * (maxTemp -
TempVals[i])), 50 + (1 + i) * 2, 300 + ((200 / maxTemp) * (maxTemp
- TempVals[i + 1])));
  }
  for (int i = 0; i < TurbVals.length - 1; i++) {
    line(425 + i * 2, 300 + ((200 / maxTurb) * (maxTurb -
TurbVals[i])), 425 + (1 + i) * 2, 300 + ((200 / maxTurb) * (maxTurb
- TurbVals[i + 1])));
  }

// Draw rating graphs

  fill(0);
  stroke(255);
  textSize(20);
  textAlign(LEFT, BOTTOM);
  text("pH: " + pHVal, 50, 275);
  text("EC: " + ECVal, 425, 275);
  text("Temperature: " + TempVal, 50, 525);
  text("Turbidity: " + TurbVal, 425, 525);

  fill(255, 0, 0);
  stroke(0);
  rect(800, 50, 50, 500);
  fill(#FFA500);
```

```
  rect(800, 175, 50, 250);
  fill(#FFFF00);
  rect(800, 225, 50, 150);
  fill(0, 255, 0);
  rect(800, 250, 50, 100);
  fill(255);

  fill(255, 0, 0);
  stroke(0);
  rect(900, 50, 50, 500);
  fill(#FFA500);
  rect(900, 250, 50, 250);
  fill(#FFFF00);
  rect(900, 400, 50, 150);
  fill(0, 255, 0);
  rect(900, 500, 50, 50);

  fill(255, 0, 0);
  stroke(0);
  rect(1000, 50, 50, 250);
  fill(255, 255, 0);
  rect(1000, 300, 50, 100);
  fill(0, 255, 0);
  rect(1000, 400, 50, 50);
  fill(0, 255, 255);
  rect(1000, 450, 50, 50);
  fill(0, 0, 255);
  rect(1000, 500, 50, 50);

  fill(255);
  rect(1100, 50, 50, 500);

  fill(0);
  stroke(255);
  strokeWeight(8);
  textSize(12);
  textAlign(CENTER, BOTTOM);
  text("pH", 825, 45);
  text("EC", 925, 45);
  text("Temp", 1025, 45);
  text("Turb", 1125, 45);

  line(800, 550 - (500 * (pHVal / 14)), 850, 550 - (500 * (pHVal /
14)));
  line(900, 550 - constrain((500 * (ECVal / 2000)), 0, 500), 950,
550 - constrain((500 * (ECVal / 2000)), 0, 500));
  line(1000, 550 - (500 * (TempVal / 105)), 1050, 550 - (500 *
```

```
(TempVal / 105)));
  stroke(100);

  line(1100, 550 - (500 * (TurbVal / 3000)), 1150, 550 - (500 *
(TurbVal / 3000)));

/*
  strokeWeight(2);
  stroke(0);
  fill(255);
  fill(255, 0, 0);
  rect(100, 525, 120, 50);
  fill(#0FA8F7);
  rect(300, 525, 120, 50);
  fill(0);
  textSize(20);
  text("STOP", 160, 560);
  text("LOG", 360, 560);
  */
}

void manageValues() {

  // Manage the input string
  pHVal = float(splitInputs[1]);
  ECVal = float(splitInputs[2]);
  TempVal = float(splitInputs[3]);
  TurbVal = float(splitInputs[4]);

  //Determine Max Values
  if (pHVal > maxpH) {
    maxpH = 5 * ceil(pHVal / 5);
  }
  if (ECVal > maxEC) {
    maxEC = 100 * ceil(ECVal / 100);
  }
  if (TempVal > maxTemp) {
    maxTemp = 10 * ceil(TempVal / 10);
  }
  if (TurbVal > maxTurb) {
    maxTurb = 100 * ceil(TurbVal / 100);
  }

  //Limit the readings values
  constrain(pHVal, 0, maxpH);
  constrain(ECVal, 0, maxEC);
  constrain(TempVal, 0, maxTemp);
```

```
  constrain(TurbVal, 0, maxTurb);

  //Shift data array
  for (int i = pHVals.length - 1; i > 0; i--) {
    pHVals[i] = pHVals[i - 1];
    ECVals[i] = ECVals[i - 1];
    TempVals[i] = TempVals[i - 1];
    TurbVals[i] = TurbVals[i - 1];
  }
  pHVals[0] = pHVal;
  ECVals[0] = ECVal;
  TempVals[0] = TempVal;
  TurbVals[0] = TurbVal;

  //Check for new high reading
  for (int i = 0; i < pHVals.length; i++) {
    float maxpHTemp = maxpH;
    float maxECTemp = maxEC;
    float maxTempTemp = maxTemp;
    float maxTurbTemp = maxTurb;

    if (pHVals[i] > maxpHTemp) {
      maxpHTemp = pHVals[i];
    }
    if (ECVals[i] > maxECTemp) {
      maxECTemp = ECVals[i];
    }
    if (TempVals[i] > maxTempTemp) {
      maxTempTemp = TempVals[i];
    }
    if (TurbVals[i] > maxTurbTemp) {
      maxTurbTemp = TurbVals[i];
    }

    maxpH = constrain(5 * ceil(maxpHTemp / 5), 0, 15);
    maxEC = 100 * ceil(maxECTemp / 100);
    maxTemp = 10 * ceil(maxTempTemp / 10);
    maxTurb = 100 * ceil(maxTurbTemp / 100);
  }

  println(pHVal, ECVal, TempVal, TurbVal);
}

void serialEvent(Serial myPort) {
  input = myPort.readString();         // read it and store it in
val
  input = trim(input);
```

```
  println(input);
  String[] prevVals = splitInputs;
  splitInputs = split(input, ',');
  if (splitInputs.length < 5) {
    splitInputs = prevVals;
  }
  manageValues();
}
```

Appendix B

Raw data for statistical test:

pH vals	pH Vals
5.178325329	5.36712314
7.124155533	7.002434141
10.82115964	10.63511212
8.308391304	8.5365789
7.385947996	7.341236821
8.00903523	8.009981231
10.54986496	9.301325674
11.05692496	9.120112989
8.976573764	8.95341386
11.49127775	11.01235343
9.462092507	7.42724387
8.626541112	8.1465465
9.798680479	8.04125763
7.152930736	5.832473937
10.09576487	9.12312034
8.21643617	8.123178573
11.10263968	9.01237389
10.68995988	9.45645373
11.12120566	9.234556364
7.873734081	7.35435435
8.660394847	7.117586681

11.25370936	8.054453524
8.806390589	8.042134474
9.823309556	9.0096413
7.681698759	6.012638439
11.66198371	7.32583935
10.48705475	10.412145
11.74168472	7.7844143
13.43243431	11.00465454
12.4822348	12.13543489
11.2313479	11.12147639
St Dev	St Dev
4.280133306	4.068942204
Mean	Mean
8.204836615	8.550053238

Bill of Materials

Component	**Price**
Arduino Mega	$3.10
TDS Sensor	$2.80
Turbidity Probe	$2.20
Temperature Sensor	$1.20
pH probe	$4.20
TFT LCD (ILI9341)	$4.57
5V Buck Converter	$0.68
DS3231 RTC Module	$0.85
SD Card Module	$0.59
General Discrete Components	$0.10
Waterproof Case	$5.50
Battery	$8.95

Total:	$34.74

REFERENCES

1. https://www.google.com/url?q=https://www.pnas.org/content/115/9/2078&sa=D&source=editors&ust=1622795345366000&usg=AOvVaw0lt0jJmSB3rK8d7qL8e-xK
2. https://www.theguardian.com/us-news/2021/mar/31/americas-tap-water-samples-forever-chemicals&sa=D&source=editors&ust=1622795345359000&usg=AOvVaw3xZZbK-UoFuF-7pi5KcMU5
3. https://www.usatoday.com/story/news/2017/08/14/63-million-americans-exposed-unsafe-drinking-water/564278001/
4. https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.dailymail.co.uk%2Fhealth%2Farticle-4488948%2FNearly-30-million-Americans-drink-contaminated-tap-water.html&psig=AOvVaw1oqqUa6s4w_k2eI7QCgirb&ust=1620555787680000&source=images&cd=vfe&ved=0CA0QjhxqFwoTCKiW1_LuufACFQAAAAAdAAAAABAI
5. https://www.pnas.org/content/115/9/2078
6. https://www.greatlakesnow.org/2020/09/drinking-unsafe-water-contaminants-solutions/
7. https://www.motherjones.com/environment/2017/05/millions-drinking-contaminated-water/
8. https://theconversation.com/nearly-60-million-americans-dont-drink-their-tap-water-research-suggests-heres-why-thats-a-public-health-problem-158483

9. https://www.epa.gov/report-environment/drinking-water