CSCE 462 Design Document

Jared Jones
Charlemagne Wong
Austin Collier
April 10, 2020

Smart FishTank Monitor

Overview

Fish are very popular because they are generally seen as low maintenance while still having the joys of a traditional pet. However, many novice owners may do things that will harm their new fish and their tank ecosystem. For this reason, we decided to make a system that could mitigate these issues by alerting the owner about the conditions of the tank and having a system that maintains their fish tank to optimal levels.

Goals

The goal of our project is to design a smart fish tank monitoring system that would display the water conditions of the tank and have a system that can maintain those conditions. This system will monitor water temperature, pH level, and water clarity and will maintain proper tank conditions with an automated feeder and a water temperature regulator. The values of those conditions are displayed on an LCD monitor that is controlled by an Arduino. These features will help fish owners manage their pets without worrying about the constant maintenance that fishes require to stay healthy. Additionally, this could help novices stay on top of managing their new pets with day to day fish care such as feeding. This project is inspired by one of our group member's pet fish, who lives in a 1-gallon tank.

Table of Contents

1. System Design

1.1 3D Printing

In order to properly design the various custom parts that were needed for the completion of this project, CAD software and 3D printing was used. This was used to custom create housings for the electronics along with the fish tank lid with custom holes for all of the sensors and that held the auto-feeding system.

1.2 Water Temperature

To maintain water temperature, we will use a relay that is attached to a heating module. The model we will be using is a fish-safe heading module with no temperature control other than an on/off mode. This heating module raises the water temperature to safe levels, but various species of fish have different temperature preferences. In order to stay healthy and maximize the comfort of the fish, this feature will allow owners to set and maintain specific water temperature. The heater will keep the tank at a fixed temperature within ±1ºC by having the Arduino constantly monitor the temperature via two temperature sensors (one for redundancy) and then either activating or deactivating the relay so the water temperature stays within the set range. If something causes the tank to heat up or cool down too quickly where the heater could not manage, the alarm would sound.

1.3 pH Sensor

To maintain the pH, we will be using a pH sensor that will constantly monitor the pH of the water. Different types of fish need various levels of pH so we would need to ensure that the water stays within the acceptable range for the specific fish(es). The user would have the option of choosing what type of fish are living in the tank, being: freshwater, tropical water, marine water, and reef and have the sensor notify the user via the screen, and in extreme cases, the alarm if the tank reaches the extremes.

1.4 Feeder

Many new fish owners tend to be unsure of how much to feed their fish and may potentially overfeed or forget to feed their pets. Our goal to resolve this issue is to make an automated feeder for the tank. This feeder will come from a custom 3D printed model as shown below that will consist of 2 parts: a stationary base that has an open area with a cutout for the feeder, an indention to keep the feeder in place, and a rotational top. This system works by having a rotational top that is divided into 8 different sections, one for each day of the week and an empty slot. The bottom stationary base will have a cutout in the same shape as the empty slot. The idea is that the owner could put the amount of food they want to feed the fish in the 7 non-empty slots for each day of the week. This will then allow the Arduino to control the stepper motor, which will turn the top around its enclosure. The stepper motor will turn the rotating top by a set distance each day or for however long the user wants to set the feeding schedule. When the rotating top rotates, the food is pushed into the opening of the stationary base. This system allows users to feed their pets with little to no management.

Refer to *Section 2.4* for images of the physical prototype

1.5 Water Clarity

In order to measure water clarity, we will be using a gravity Arduino turbidity sensor. This sensor would be able to tell the total amount of total suspended solids in the water, which is used to calculate the liquid turbidity level. High levels of turbidity can signal potential pollution stemming from bacteria and algae, which could cause harm to the fish tank. This sensor will notify for a water change(this would be output on the screen as a suggestion) and can help ensure a healthy fish and tank ecosystem. The top of this sensor is not waterproof, so we will be 3d printing a case for the top to avoid any possible splashing and from other water damage.

1.6 Alarm

An issue we thought of with the system is "what if the user doesn't check the system because they think everything is functioning properly?" The solution we designed is an alarm system that would set off if any water conditions would require user management. Such conditions could include food finishing its 7 cycles, the pH requiring a water change, or connection/damage to a sensor. In those scenarios, the Arduino, connected to an audio amplifier and a small speaker, will output an alarm for each foreseeable error. This would be especially useful for novice fish owners who likely forget to check the tank or simply be unaware of any specific problems.

1.7 Soldering

For soldering, we expect around 30-60 joints to solder. We already have our own soldering iron including solder, flux, solder pump, wick, and other soldering tools. Our soldering tools are packed to be easily mobile so that members can share the equipment with relative ease.

1.8 Design Changes and Setbacks

Due to the COVID-19 outbreak, various parts of the project were adapted to be better completed without contact between the members. Group meetings were held bi-weekly with one meeting on Friday during lab time and another held usually at different times depending on the stage of the project. These meetings were held originally in person but have transitioned to Google Hangouts after the outbreak. This allows group members to continue their meetings and to update one another on their respective goals and progress. Many shipping issues occurred with some parts being delayed by months. To combat this, various parts were replaced by a similar part. Instead of an ESP8266, which was delayed in shipping, a Raspberry Pi Zero W is used to provide WiFi functionality of the system. An Arduino is also used to connect the sensors and control the motor. The team chose this route instead of using a single Raspberry Pi Zero W because the majority of components and modules operate at 5V, which is incompatible with the Raspberry Pi's 3.3V pins. This is resolved by using a 5V to a 3.3V level shifter, but it was preferable to avoid ordering unnecessary components due to the risk of shipment delays. Using an Arduino as a bridge between the 5V and 3.3V signal worked as a sufficient compromise.

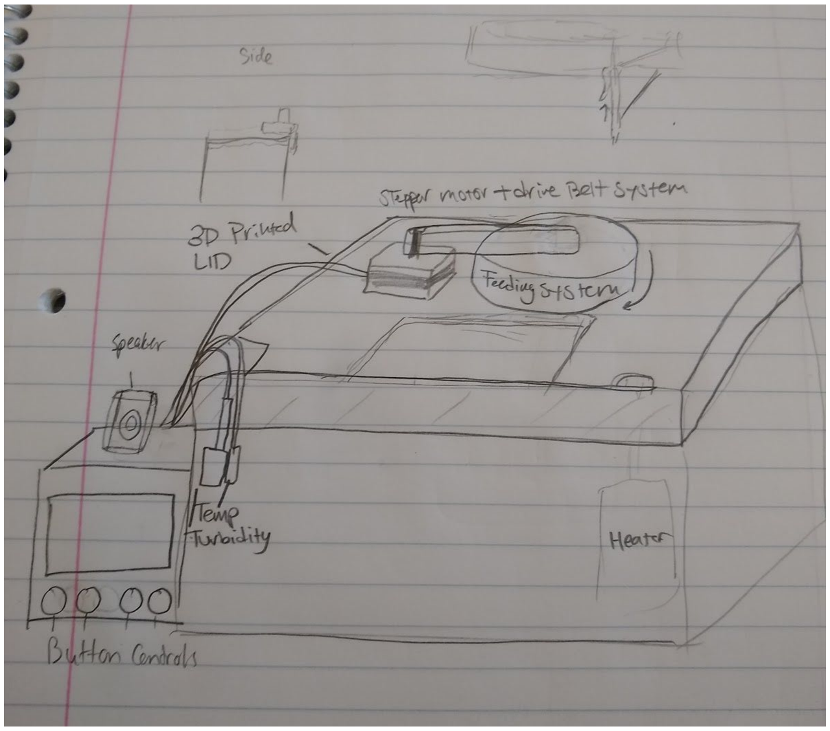
1.9 Safety

Due to the COVID-19 outbreak, measures are taken to properly sanitize the parts before giving the items to another team member. The tools have been wiped with isopropyl alcohol to minimize the potential spread of the virus.

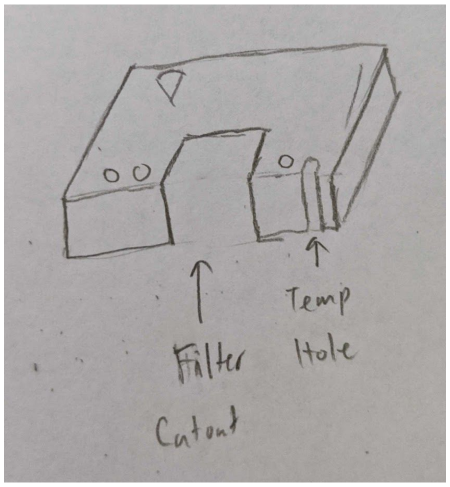
The power supply used for this project takes an AC voltage input and converts it to a 5V DC output. The members of the team properly researched before wiring the power supply to ensure that no mistakes are made. The team also purchased a power socket with a built in fuse to protect the circuit from shorts and sudden high current and the AC lines are securely protected by the power supply’s built-in cover.

2. Diagrams and Schematics

2.1 Initial Sketch



Overall System Sketch

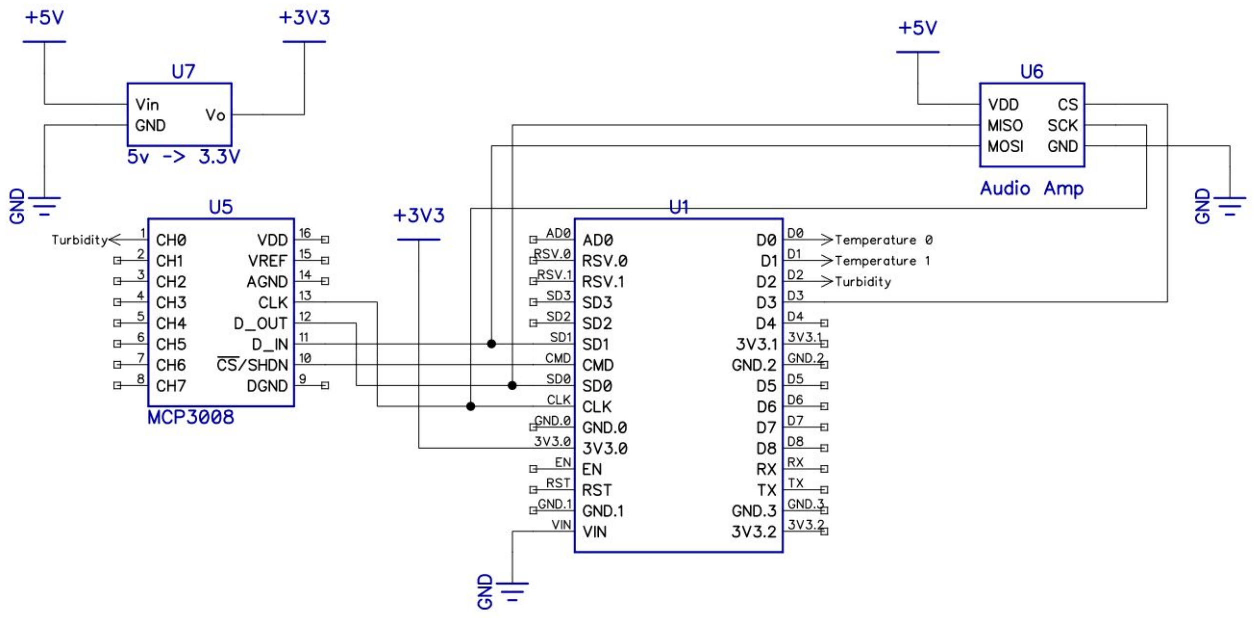


Fish Tank Cover Sketch

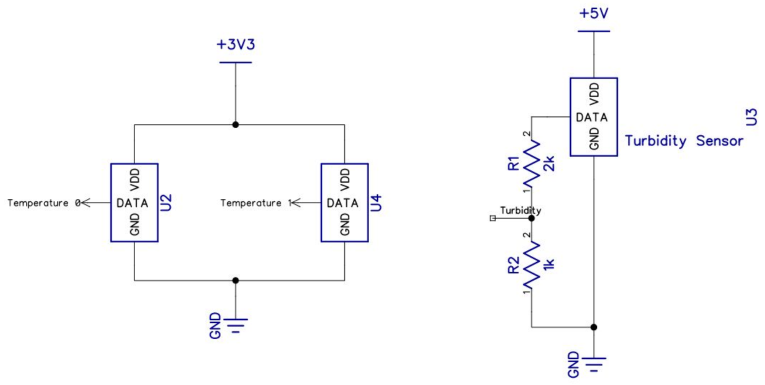
2.2 Schematics

2.2.1 Original Schematics

This is the original prototype schematic for the system's electronics. It uses an ESP32 - WiFi enabled Arduino. This is designed in the DipTrace ECAD software.



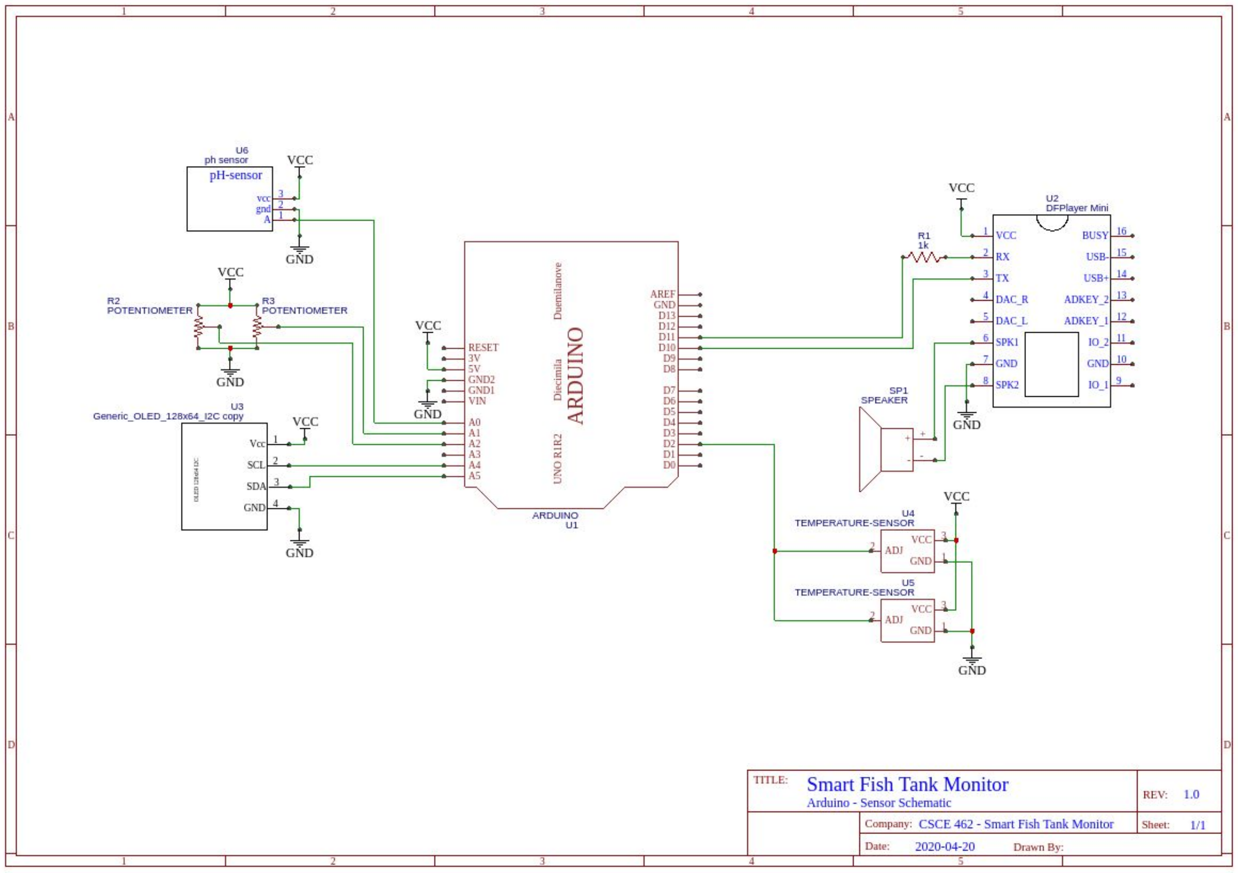
Work in Progress Arduino Schematic using ESP3822 Arduino



Work in Progress Schematic of Sensor Wiring

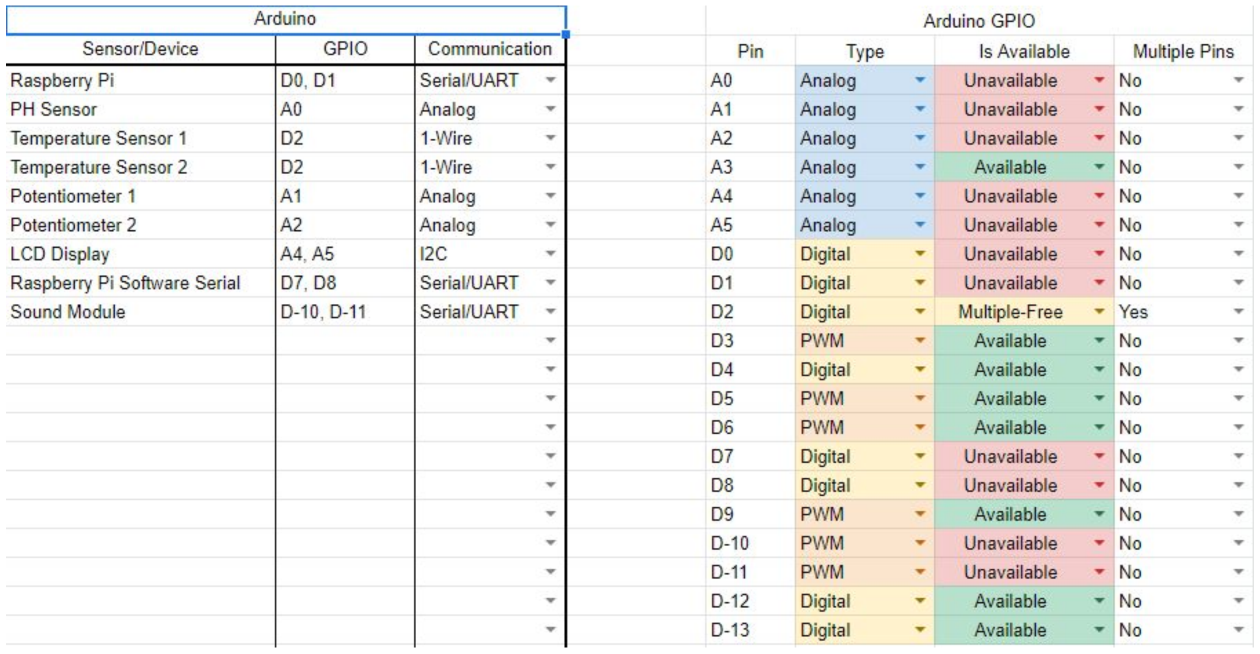
2.2.2 Updated Arduino Schematics

These are the schematics of the Arduino because we were not able to obtain some of the parts from the viral epidemic



New Arduino schematic with all sensors and modules

2.2.3 Arduino Pin Mapping

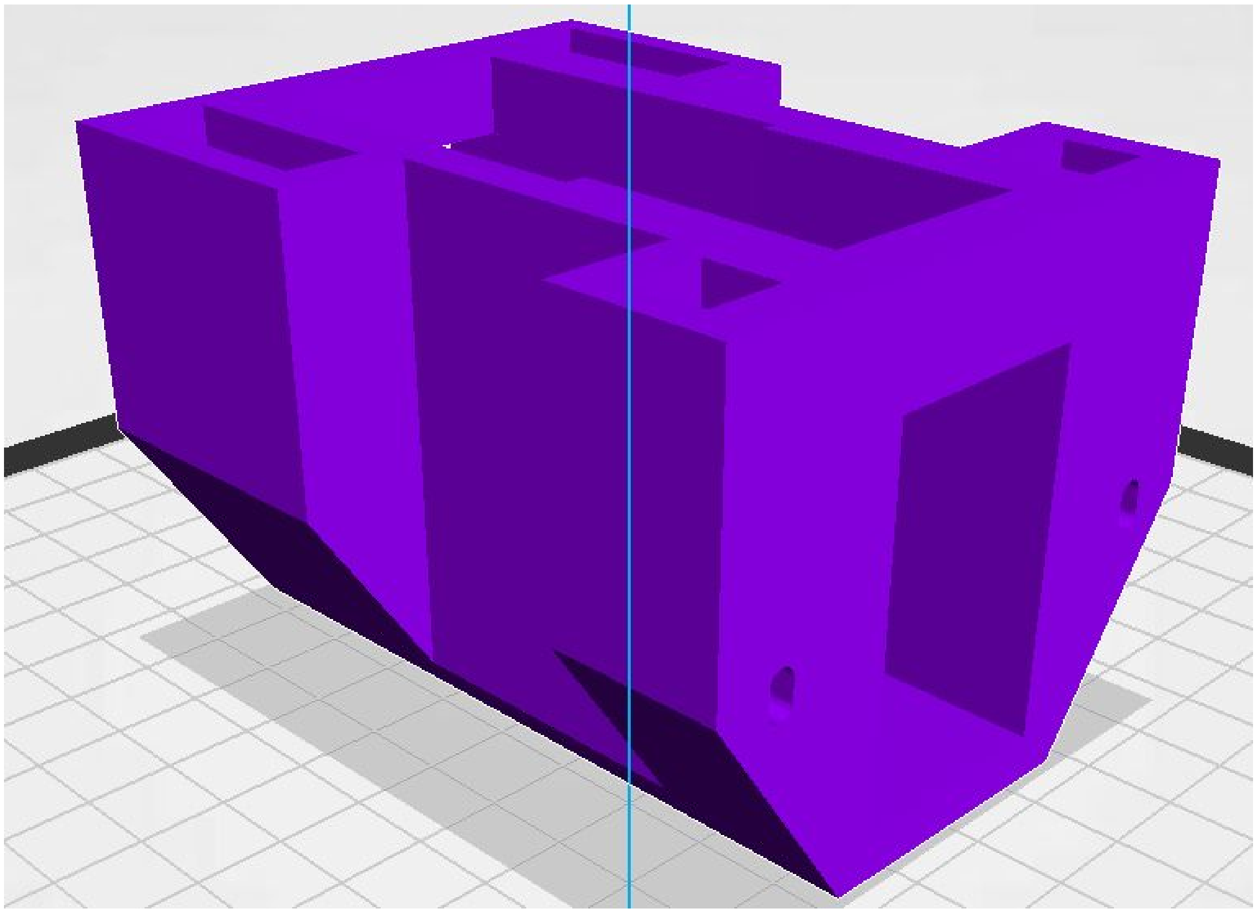


Arduino		
Sensor/Device	GPIO	Communication
Raspberry Pi	D0, D1	Serial/UART
PH Sensor	A0	Analog
Temperature Sensor 1	D2	1-Wire
Temperature Sensor 2	D2	1-Wire
Potentiometer 1	A1	Analog
Potentiometer 2	A2	Analog
LCD Display	A4, A5	I2C
Raspberry Pi Software Serial	D7, D8	Serial/UART
Sound Module	D-10, D-11	Serial/UART

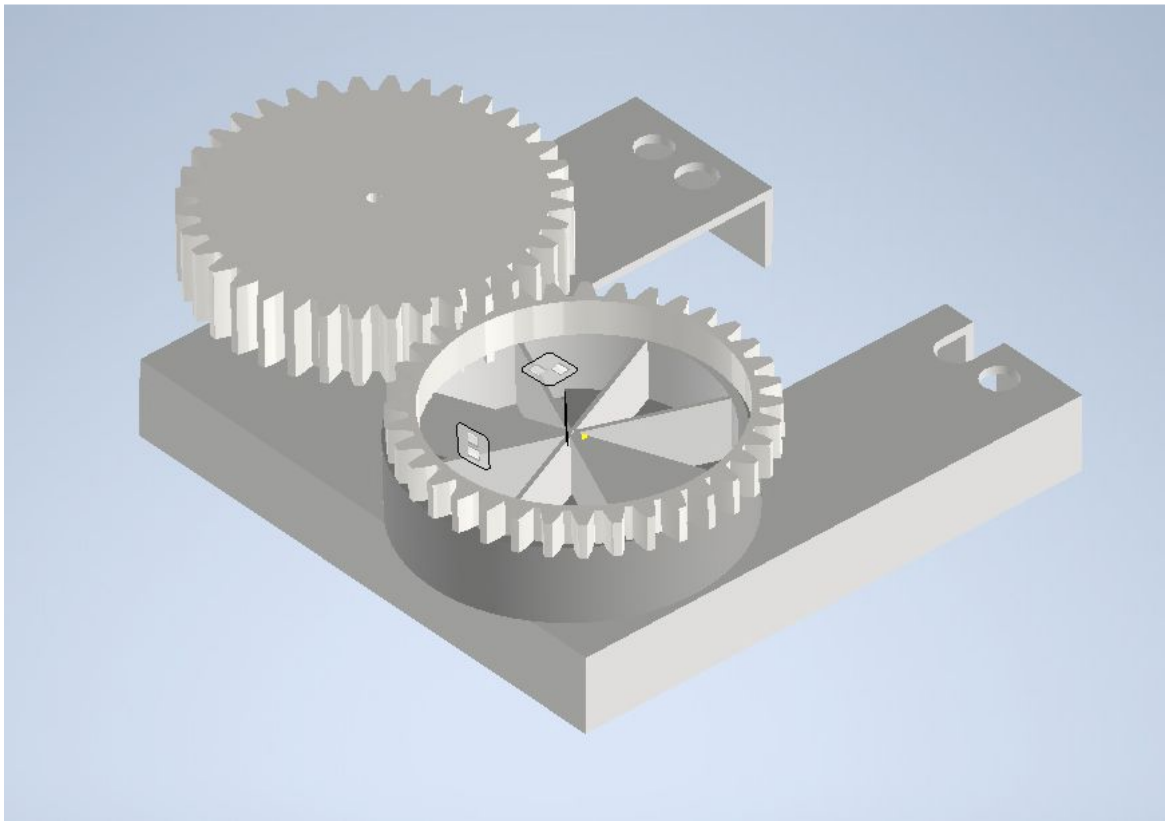
Arduino GPIO			
Pin	Type	Is Available	Multiple Pins
A0	Analog	Unavailable	No
A1	Analog	Unavailable	No
A2	Analog	Unavailable	No
A3	Analog	Available	No
A4	Analog	Unavailable	No
A5	Analog	Unavailable	No
D0	Digital	Unavailable	No
D1	Digital	Unavailable	No
D2	Digital	Multiple-Free	Yes
D3	PWM	Available	No
D4	Digital	Available	No
D5	PWM	Available	No
D6	PWM	Available	No
D7	Digital	Unavailable	No
D8	Digital	Unavailable	No
D9	PWM	Available	No
D-10	PWM	Unavailable	No
D-11	PWM	Unavailable	No
D-12	Digital	Available	No
D-13	Digital	Available	No

Arduino Pinout

2.3 CAD

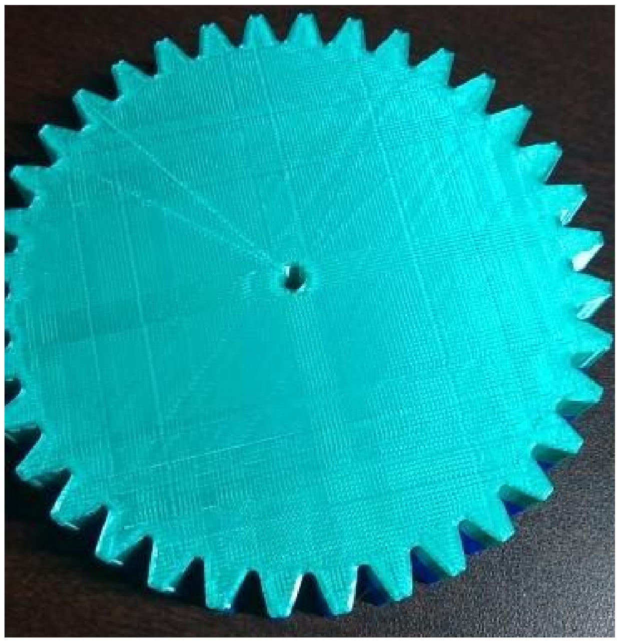


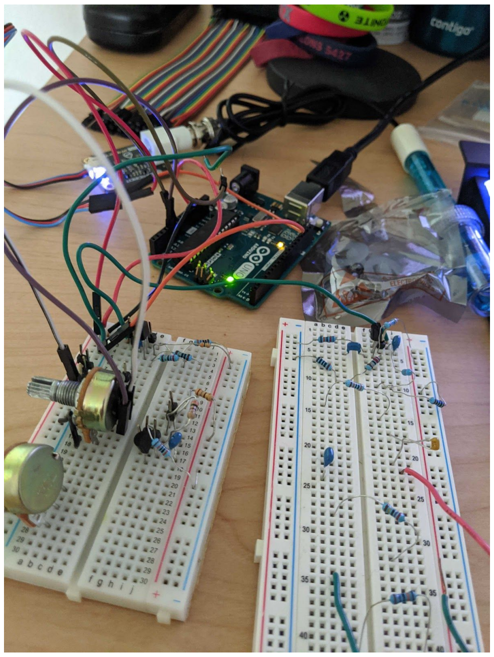
3D Model of “Box” to hold the display, potentiometers, and electronics

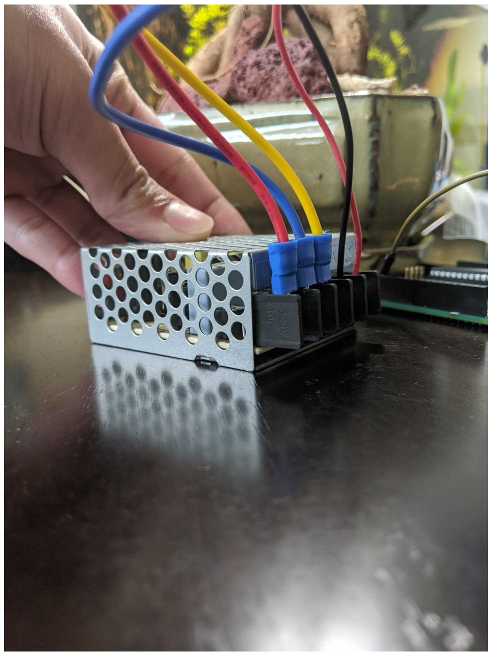


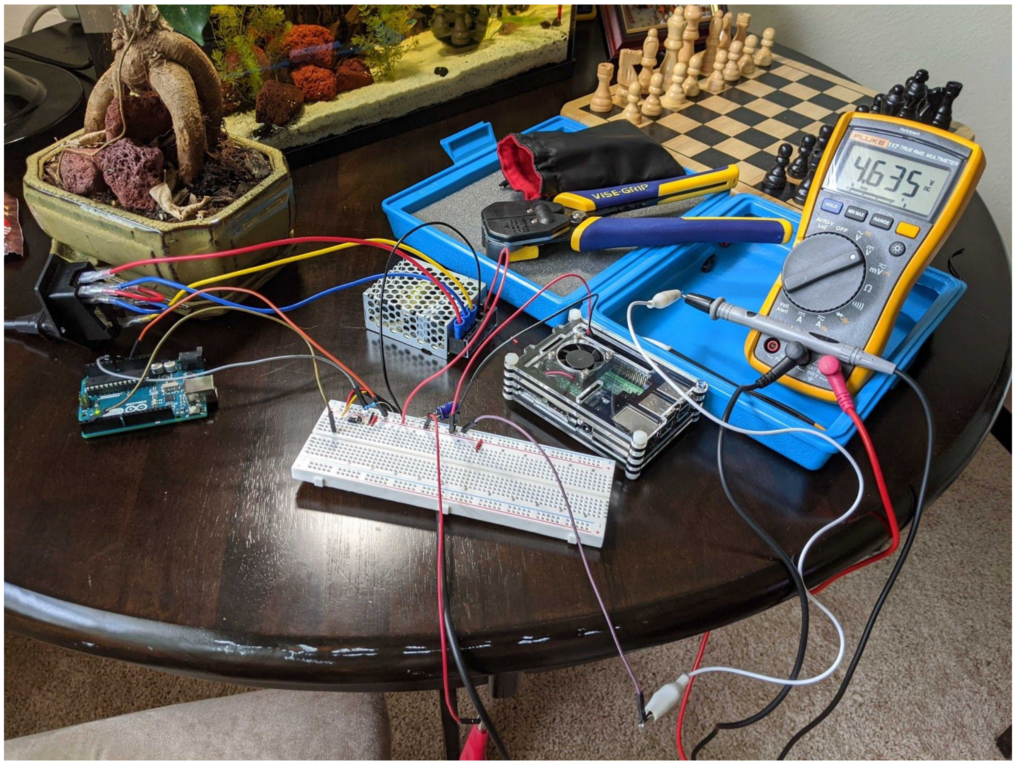
CAD of Tank Cover

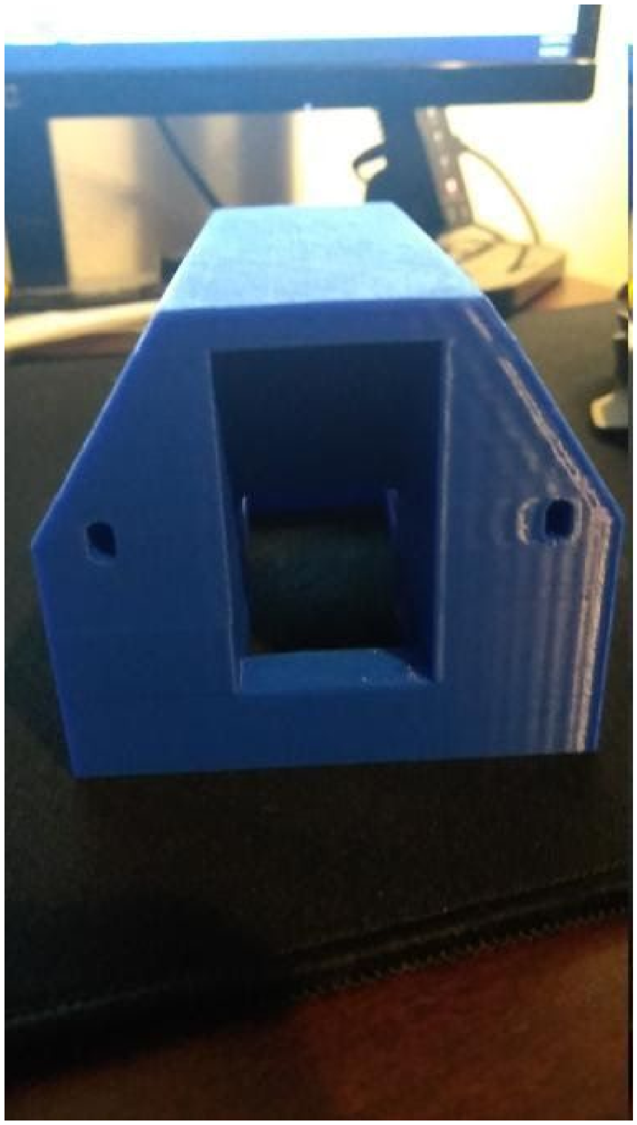
2.4 Prototype

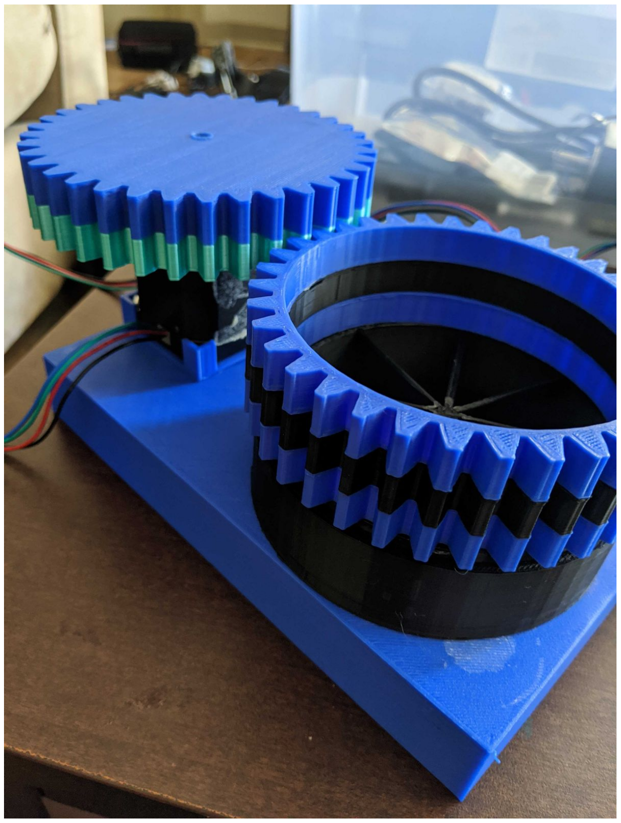








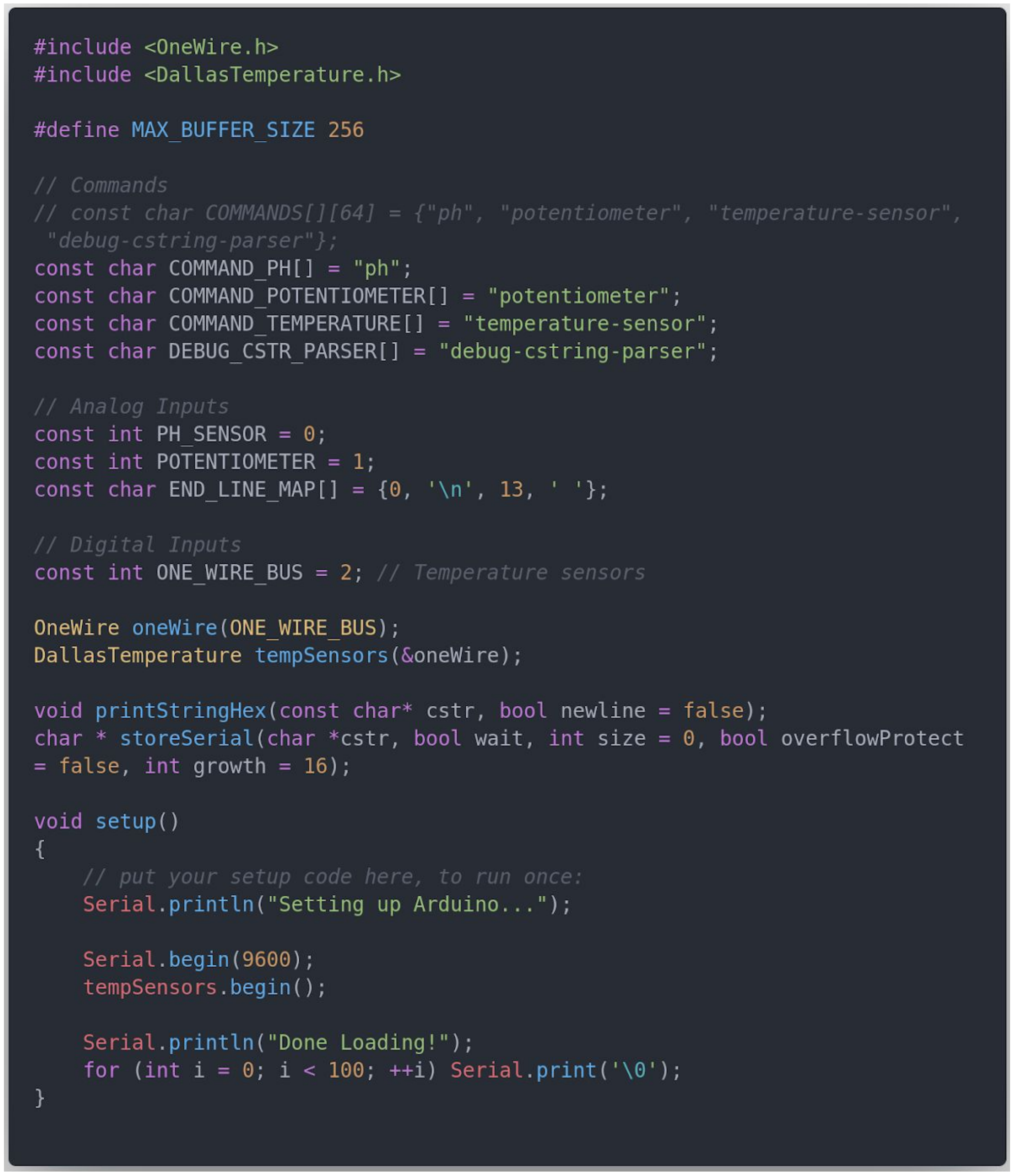




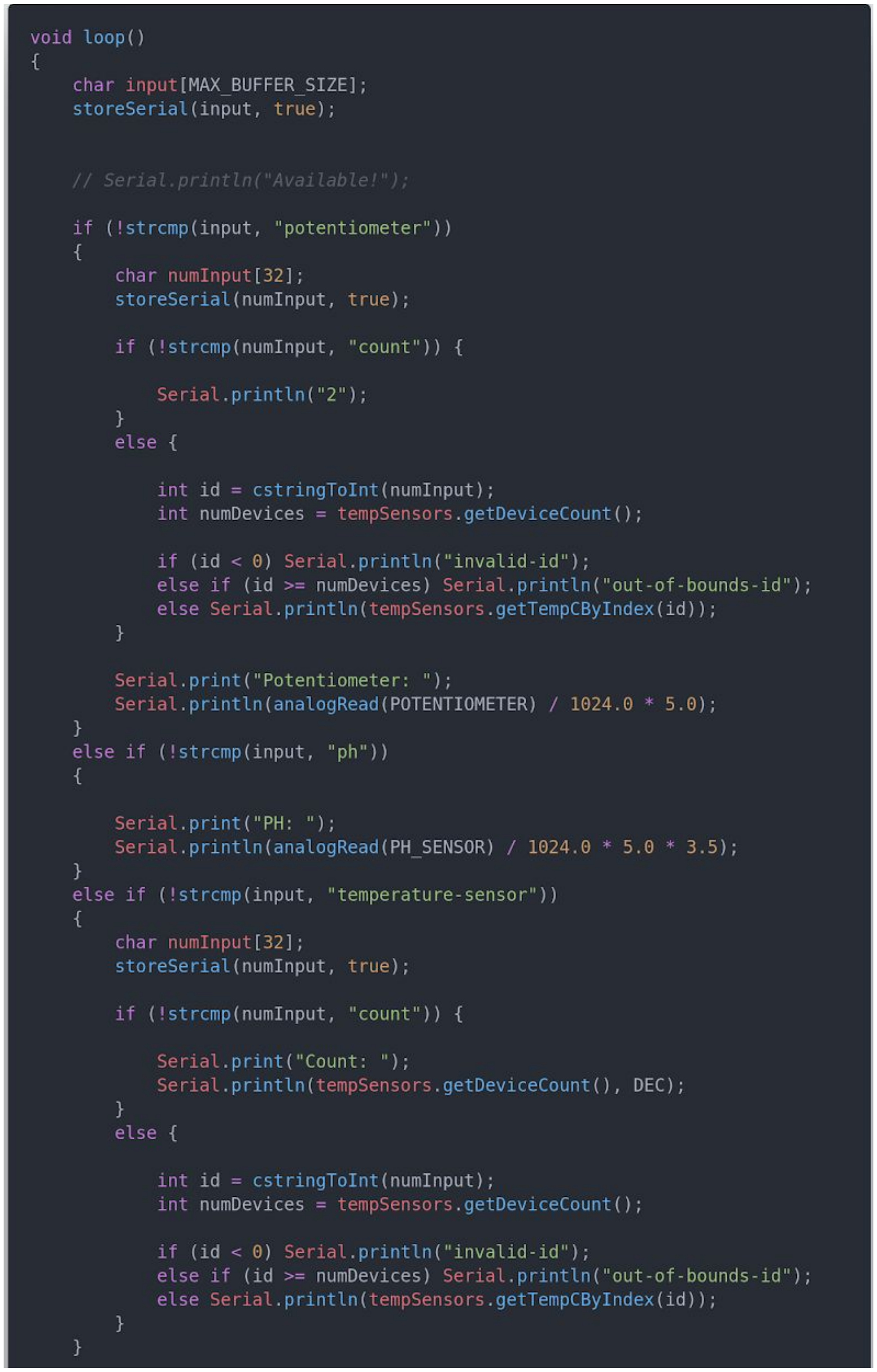
3. Code

3.1 Arduino Code

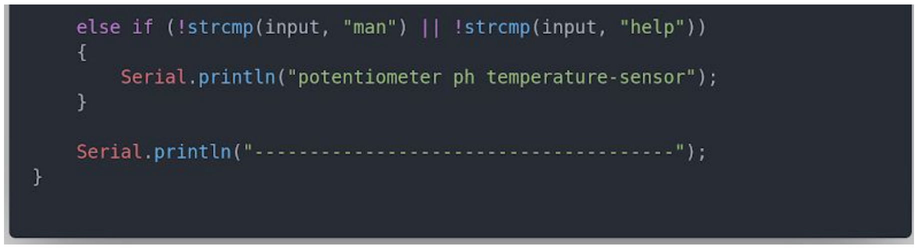
Github: https://github.com/cmasterx/Smart-Fish-Tank-Arduino



```
#include <OneWire.h>
#include <DallasTemperature.h>

#define MAX_BUFFER_SIZE 256

// Commands
// const char COMMANDS[][64] = {"ph", "potentiometer", "temperature-sensor",
 "debug-cstring-parser"};
const char COMMAND_PH[] = "ph";
const char COMMAND_POTENTIOMETER[] = "potentiometer";
const char COMMAND_TEMPERATURE[] = "temperature-sensor";
const char DEBUG_CSTR_PARSER[] = "debug-cstring-parser";

// Analog Inputs
const int PH_SENSOR = 0;
const int POTENTIOMETER = 1;
const char END_LINE_MAP[] = {0, '\n', 13, ' '};

// Digital Inputs
const int ONE_WIRE_BUS = 2; // Temperature sensors

OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature tempSensors(&oneWire);

void printStringHex(const char* cstr, bool newline = false);
char * storeSerial(char *cstr, bool wait, int size = 0, bool overflowProtect
= false, int growth = 16);

void setup()
{
    // put your setup code here, to run once:
    Serial.println("Setting up Arduino...");

    Serial.begin(9600);
    tempSensors.begin();

    Serial.println("Done Loading!");
    for (int i = 0; i < 100; ++i) Serial.print('\0');
}
```

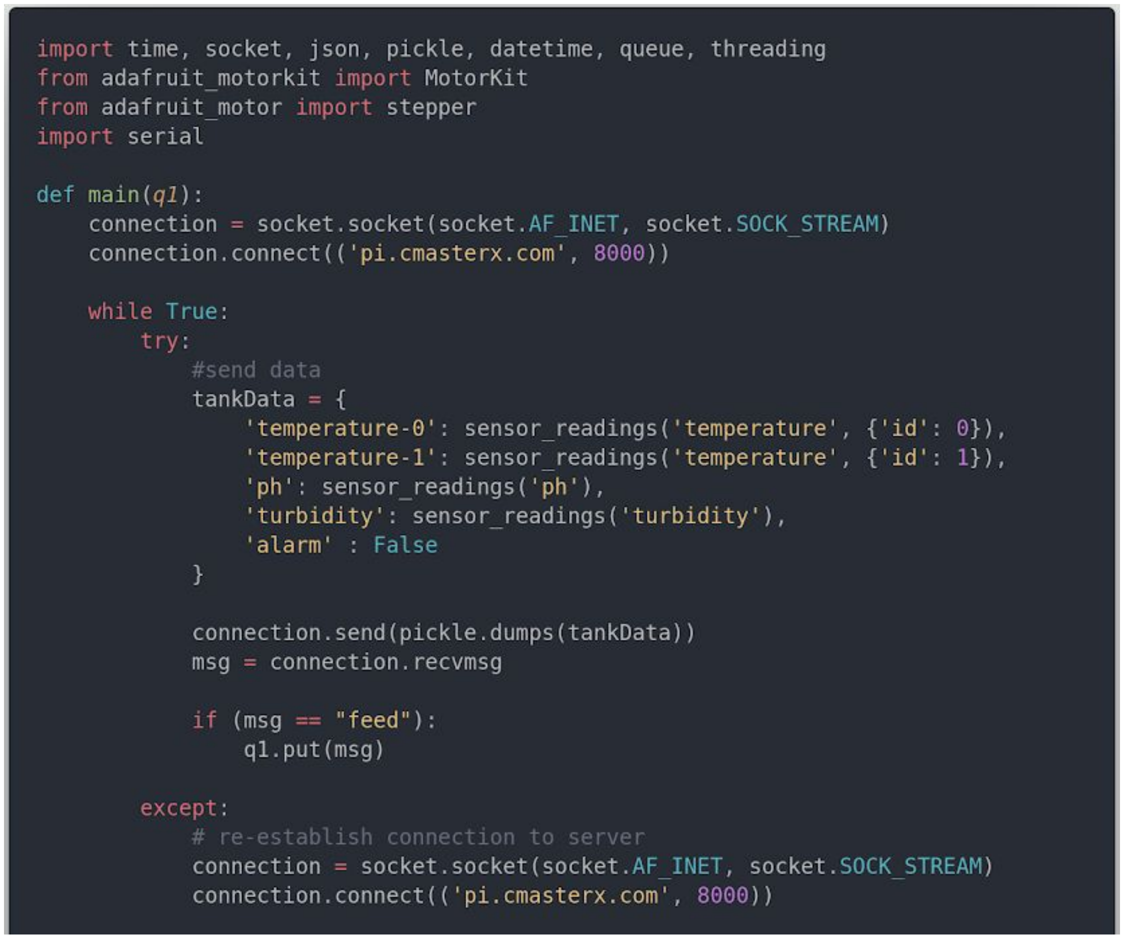


```
void loop()
{
    char input[MAX_BUFFER_SIZE];
    storeSerial(input, true);


    // Serial.println("Available!");

    if (!strcmp(input, "potentiometer"))
    {
        char numInput[32];
        storeSerial(numInput, true);

        if (!strcmp(numInput, "count")) {

            Serial.println("2");
        }
        else {

            int id = cstringToInt(numInput);
            int numDevices = tempSensors.getDeviceCount();

            if (id < 0) Serial.println("invalid-id");
            else if (id >= numDevices) Serial.println("out-of-bounds-id");
            else Serial.println(tempSensors.getTempCByIndex(id));
        }

        Serial.print("Potentiometer: ");
        Serial.println(analogRead(POTENTIOMETER) / 1024.0 * 5.0);
    }
    else if (!strcmp(input, "ph"))
    {

        Serial.print("PH: ");
        Serial.println(analogRead(PH_SENSOR) / 1024.0 * 5.0 * 3.5);
    }
    else if (!strcmp(input, "temperature-sensor"))
    {
        char numInput[32];
        storeSerial(numInput, true);

        if (!strcmp(numInput, "count")) {

            Serial.print("Count: ");
            Serial.println(tempSensors.getDeviceCount(), DEC);
        }
        else {

            int id = cstringToInt(numInput);
            int numDevices = tempSensors.getDeviceCount();

            if (id < 0) Serial.println("invalid-id");
            else if (id >= numDevices) Serial.println("out-of-bounds-id");
            else Serial.println(tempSensors.getTempCByIndex(id));
        }
    }
```

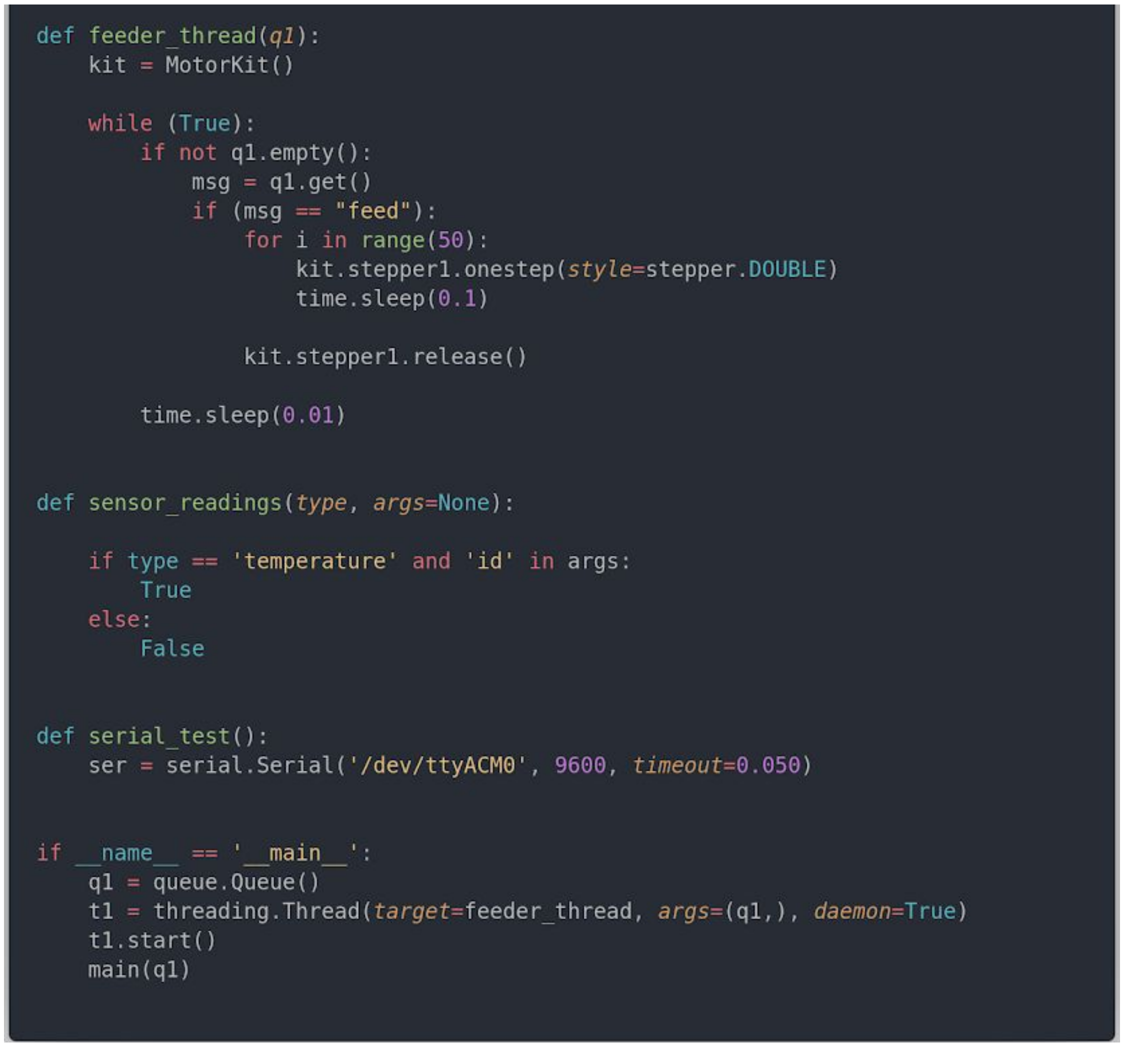


```
    else if (!strcmp(input, "man") || !strcmp(input, "help"))
    {
        Serial.println("potentiometer ph temperature-sensor");
    }

    Serial.println("-------------------------------------");
}
```

3.2 Main Raspberry Pi Code

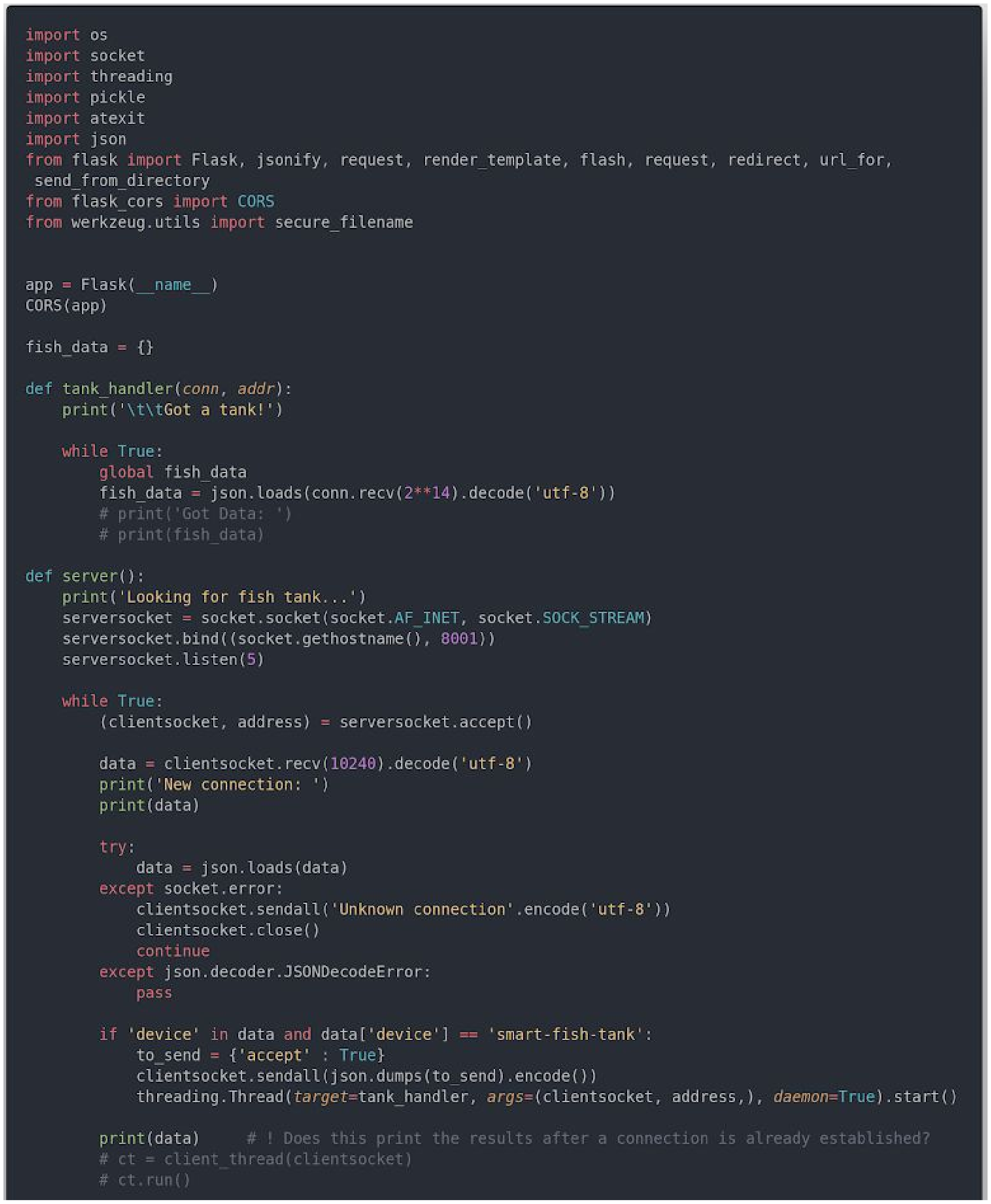


```
import time, socket, json, pickle, datetime, queue, threading
from adafruit_motorkit import MotorKit
from adafruit_motor import stepper
import serial

def main(q1):
    connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    connection.connect(('pi.cmasterx.com', 8000))

    while True:
        try:
            #send data
            tankData = {
                'temperature-0': sensor_readings('temperature', {'id': 0}),
                'temperature-1': sensor_readings('temperature', {'id': 1}),
                'ph': sensor_readings('ph'),
                'turbidity': sensor_readings('turbidity'),
                'alarm' : False
            }

            connection.send(pickle.dumps(tankData))
            msg = connection.recvmsg

            if (msg == "feed"):
                q1.put(msg)

        except:
            # re-establish connection to server
            connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            connection.connect(('pi.cmasterx.com', 8000))
```

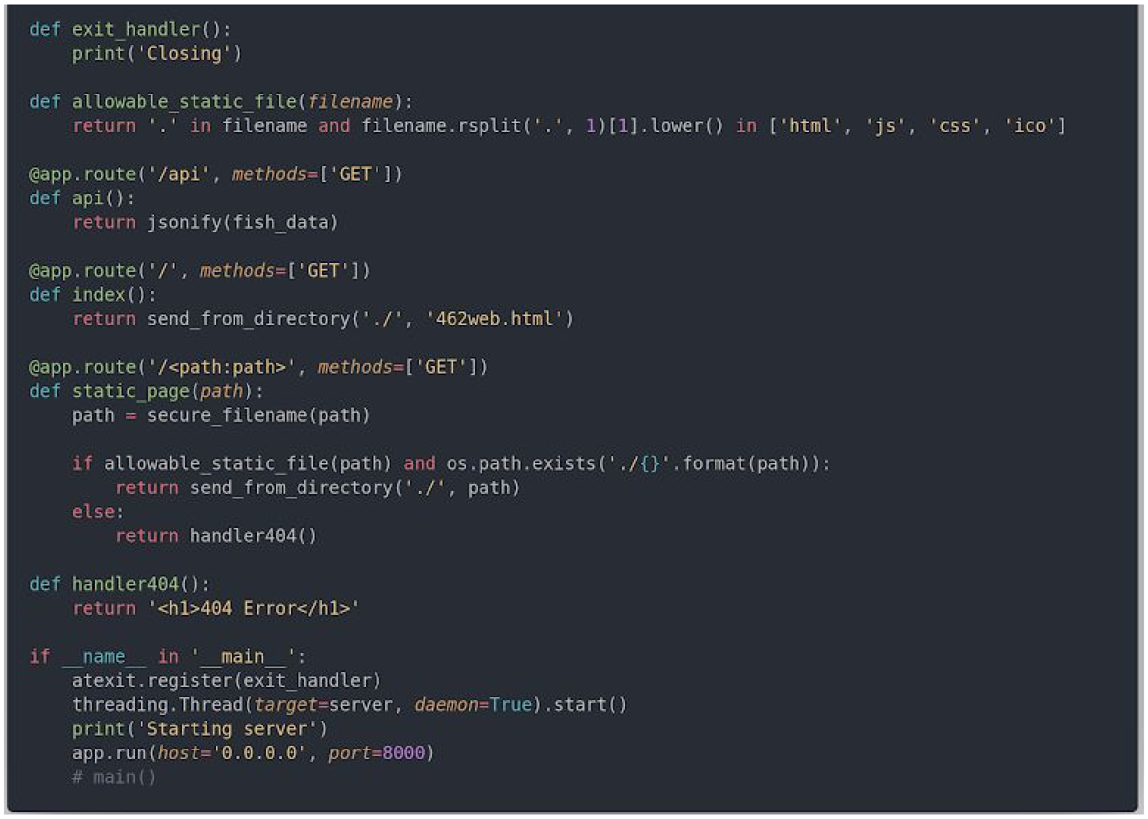


```
def feeder_thread(q1):
    kit = MotorKit()

    while (True):
        if not q1.empty():
            msg = q1.get()
            if (msg == "feed"):
                for i in range(50):
                    kit.stepper1.onestep(style=stepper.DOUBLE)
                    time.sleep(0.1)

                kit.stepper1.release()

        time.sleep(0.01)


def sensor_readings(type, args=None):

    if type == 'temperature' and 'id' in args:
        True
    else:
        False


def serial_test():
    ser = serial.Serial('/dev/ttyACM0', 9600, timeout=0.050)


if __name__ == '__main__':
    q1 = queue.Queue()
    t1 = threading.Thread(target=feeder_thread, args=(q1,), daemon=True)
    t1.start()
    main(q1)
```

3.3 Main Web Server Code



```python
import os
import socket
import threading
import pickle
import atexit
import json
from flask import Flask, jsonify, request, render_template, flash, request, redirect, url_for,
 send_from_directory
from flask_cors import CORS
from werkzeug.utils import secure_filename


app = Flask(__name__)
CORS(app)

fish_data = {}

def tank_handler(conn, addr):
    print('\t\tGot a tank!')

    while True:
        global fish_data
        fish_data = json.loads(conn.recv(2**14).decode('utf-8'))
        # print('Got Data: ')
        # print(fish_data)

def server():
    print('Looking for fish tank...')
    serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    serversocket.bind((socket.gethostname(), 8001))
    serversocket.listen(5)

    while True:
        (clientsocket, address) = serversocket.accept()

        data = clientsocket.recv(10240).decode('utf-8')
        print('New connection: ')
        print(data)

        try:
            data = json.loads(data)
        except socket.error:
            clientsocket.sendall('Unknown connection'.encode('utf-8'))
            clientsocket.close()
            continue
        except json.decoder.JSONDecodeError:
            pass

        if 'device' in data and data['device'] == 'smart-fish-tank':
            to_send = {'accept' : True}
            clientsocket.sendall(json.dumps(to_send).encode())
            threading.Thread(target=tank_handler, args=(clientsocket, address,), daemon=True).start()

        print(data)     # ! Does this print the results after a connection is already established?
        # ct = client_thread(clientsocket)
        # ct.run()
```



```python
def exit_handler():
    print('Closing')

def allowable_static_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ['html', 'js', 'css', 'ico']

@app.route('/api', methods=['GET'])
def api():
    return jsonify(fish_data)

@app.route('/', methods=['GET'])
def index():
    return send_from_directory('./', '462web.html')

@app.route('/<path:path>', methods=['GET'])
def static_page(path):
    path = secure_filename(path)

    if allowable_static_file(path) and os.path.exists('./{}'.format(path)):
        return send_from_directory('./', path)
    else:
        return handler404()

def handler404():
    return '<h1>404 Error</h1>'

if __name__ in '__main__':
    atexit.register(exit_handler)
    threading.Thread(target=server, daemon=True).start()
    print('Starting server')
    app.run(host='0.0.0.0', port=8000)
    # main()
```

4. Design/Physical Manufacturing

4.1 3D Models

Most of the 3d Modelling was done with Autodesk Inventor using the TAMU VOAL system. These models were made based on the initial design sketches and diagrams and were created with holes and cutouts that would fit the parts that were ordered for the rest. The auto-feeding system for the food was also sketched and designed primarily in Autodesk and then constrained and tested in software before it was printed to ensure that it would properly work.

4.2 Electronics

The main parts of the electronics were designed on DipTrace, a circuit CAD software. The arduino and the sensors were modeled in this software, as shown on *Section 2.2*. The other components were relatively simple, so they were immediately prototyped on a breadboard. After the components were tested for functionality, the components are moved to a protoboard where they are soldered for permanent placement.

The total components used are:

- Raspberry Pi
- Arduino Uno
- 2 Temperature Sensors
- pH Sensor
- 2 Potentiometers
- LCD Display
- Speaker Amplifier Module
- Stepper Motor Driver
- 5V 3A Power Supply

4.3 Software

Since both the Raspberry Pi Zero and the Arduino are used to control the functions of the system, the development is split into two parts: the main controller in Python running on the Raspberry Pi and C++ running on the Arduino.

The Python code functions as the main program that determines all the functions of the monitoring system. This code will communicate to a server where it will receive information on when to feed the fish, the set water temperature, and to report various water conditions of the tank. This software will also communicate to the Arduino via the Serial Port.

The C++ code in the Arduino functions as a slave device that will return requested values to the Raspberry Pi and will receive commands from the Raspberry Pi to feed the fishes. The Arduino will receive commands via its Serial Port. Additionally, it will send relevant data to the Raspberry Pi, such as sensor values, when requested by the master device.

4.4 Challenges

- Corona Virus makes collaboration difficult between team members who were working on the same parts of the project.
- Each member was assigned different parts of the project to work on after the rules about staying in place were made. This made it difficult because the parts for the project were divided up between multiple members and it had to be coordinated who had what parts and what they would work on.
- Parts no longer available or takes too long to ship
    - Used Arduino for 5V sensors and Raspberry Pi for WiFi
    - Original plan - single Arduino with WiFi
- Arduino ran out of memory and caused crashes because it was not the original arduino we wanted for the project.
- Could not purchase more 3D Printing Filament due to COVID-19 and amazon only shipping essential supplies, while printer filament was in short supply.
- Turbidity (Water Clarity) Sensor no longer available to be shipped in order to arrive on time even with expedited shipping so it was simulated in the software so that it could be added at a later date when shipping and amazon processes normalize.

5. Bill of Materials

5.1 Full Price Breakdown

Google Sheets: http://bit.ly/fish_budget

Link includes full price breakdown and links to individual components

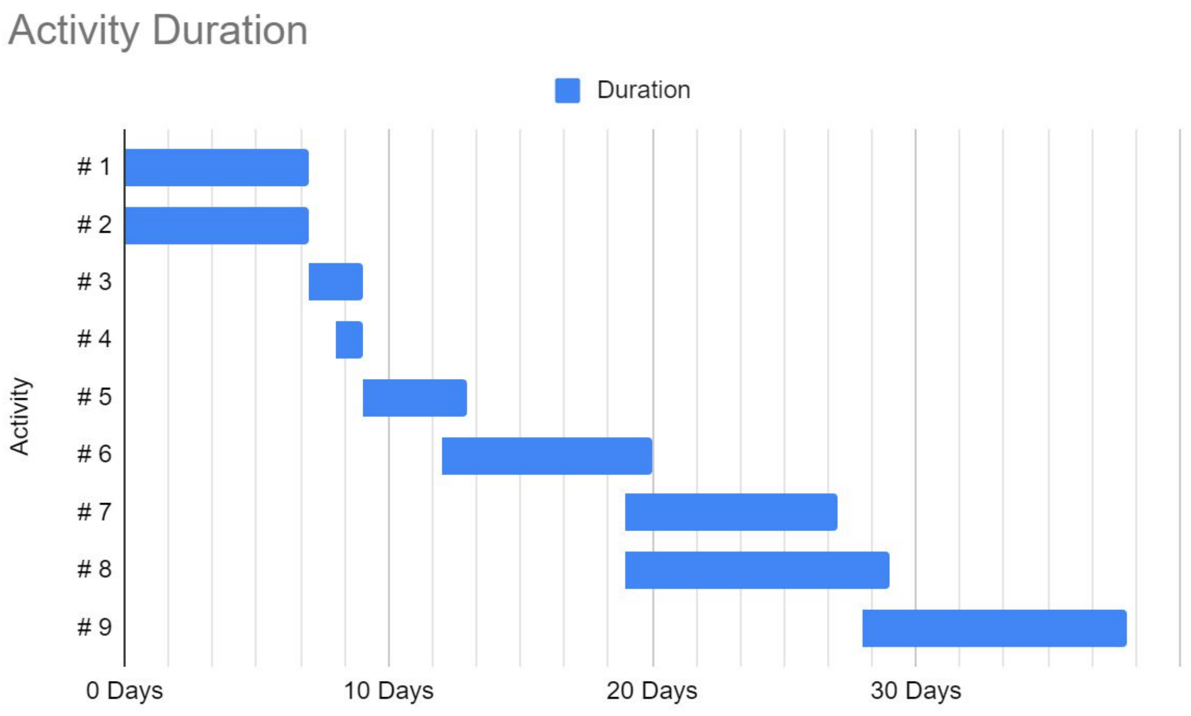
SCAN ME

5.2 Basic Price Breakdown

Item	Individual Price	Count	Shipping Cost	Total Price
Controller				
MCP3008 ADC	$12.50	1	$0.00	$12.50
Monitoring Sensors				
5 Pack Temperature Sensor	$11.98	1	$0.00	$11.98
PH Sensor Kit with Calibration Solution	$39.50	1	$15.00	$54.50
Turbidity Sensor	$14.90	1	$0.00	$0.00
Control Box				
2.8 Inches TFT Touch Screen	$6.99	1	$0.00	$6.99
Potentiometer 10-set	$6.99	1	$0.00	$6.99
4 pack speakers	$5.99	1	$0.00	$5.99
5V 3A Power Supply	$11.99	1	$0.00	$11.99
Socket Inlet	$8.99	1	$0.00	$8.99
Total		9		$119.93

6. Timeline

Activity #	Beginning Date	End Date	Duration	Milestone
1	3/9/2020	3/15/2020	7 Days	Design Fully Completed Feeding System and Top Cover
2	3/9/2020	3/15/2020	7 Days	Design Fully Completed Circuit Schematic
3	3/16/2020	3/17/2020	2 Days	Review designs
4	3/17/2020	3/17/2020	1 Days	Order Parts
5	3/18/2020	3/21/2020	4 Days	3D Print Designs
6	3/21/2020	3/28/2020	8 Days	Build Circuit
7	3/28/2020	4/4/2020	8 Days	Testing
8	3/28/2020	4/6/2020	10 Days	Program Arduinos
9	4/6/2020	4/15/2020	10 Days	Final Testing and Documentation


Activity Duration
Duration
Activity
# 1
# 2
# 3
# 4
# 5
# 6
# 7
# 8
# 9
0 Days
10 Days
20 Days
30 Days