# Embedded Aquatic Monitor Design Using the ESP32

Kyle Nilsson
Computer and Electrical Engineering
Colorado State Unviersy
Fort Collins, United States
kylenils@colostate.edu

Spencer Beer
Computer and Electrical Engineering
Colorado State University
Fort Collins, United States
beersc@colostate.edu

***Abstract* — This report describes the process of designing and creating a smart fish tank. The fish tank was automated using multiple esp32 microcontrollers. The fish tank can record water temperature data, water quality data, and a constant video live stream that is constantly watching your fish. The fish tank also has an automatic fish food dispenser that is connected to a stepper motor, and the user is able to set the feeding frequency. Last, there are LED lights attached to the fish tank that can alert the user if our system detects there is an issue. These functions are controlled and can be accessed via our webserver on the esp32.**

## I. Introduction

Animals and pets provide many benefits to humans, but there is a great responsibility to take care of one. In today's world, everything is becoming more automated, and makes mundane tasks more efficient. Taking care of fish is no exception, so we created the smart fish tank. Our goal of this prototype is to make fish owner's job easier. Whether you have a single fish or even an entire aquarium, the applications for our design are limitless. There are six main functions to our prototype:

- Live water temperature readings using the DS18B20 temperature sensor.
- Live water quality readings using the TDS water quality sensor.
- Timed automatic feeding using a stepper motor to dispense the food.
- LED lights to indicate if there is a problem in the fish tank that needs to be addressed.
- Live stream of fish tank using esp32 camera
- Webserver that hosts all the data received

The goal of this project was to make something, unlike other smart feeders, smart thermometers, and smart cameras, a user interface that provides real-time data providing information, and alerts about aquariums, or any other aquatic features, as well as customizability features of the aquarium. These six tasks are all controlled by two esp32 microcontrollers. Esp32 is a series of low-cost, low-power system on a chip microcontroller with integrated Wi-Fi and dual-mode Bluetooth. Most of the tasks are run from a single esp32 Devkit v1, and only the camera is run by the esp32 Wrover module. For the setup, the user only has to power up the esp32 and provide the Wi-Fi credentials. From there, the user has access to their own webserver that contains all the setting configurations, sensor readings, and live stream of the tank.

## II. Related work

### *A. Similar Projects and Choosing Componenets*

A smart fish tank is not a completely unique idea. Although, all the current designs we came across did not meet the standards we wanted for our smart fish tank. The majority of the designs had a method to automatically dispense food, but that was it. No design had all six of the tasks implemented into one. The other prototypes we saw also used different microcontrollers to control everything, most used raspberry pi's or a tinker board. After careful consideration, we chose the esp32 microcontroller as the optimal choice for our project's requirements. These are the main reasons.

- **Open Source:** Currently offered devices are either quite expensive or large in terms of weight and size. Moreover, very few modules are open-source devices and have no restriction in the operation purpose. Since it is open source, there is a lot of API's, llibraries, and ddocumentations on the microcontroller. Multiple different versions of the esp32 have been developed because of the free documentation and allow software developers to focus on optimizing the program. Since this chip is open source, everyone can develop an operating system for the esp32, thus there are also solutions on the Internet to program it in LUA, JavaScript, etc.
- **Optimizations:** ESP IDF platform, in which development is provided through C language with a native firmware downloader, allows you to make better use of hardware properties like scheduling tasks on a core, although still achievable in C++ through the Arduino IDE.
- **RTOS and Architecture:** The real-time operating system (RTOS) on esp32 is FreeRTOS. It is open source, designed for embedded systems and provides basic functions to the higher-level applications. The core functions are memory management, task management and API synchronization. The ESP-IDF

was developed for Linux, thus a Linux terminal is required in order to execute the bash files. However, it is possible to develop in Windows by using MSYS2. The esp32 microcontroller module is a dual-core system with two Harvard Architecture Xtensa LX6 CPUs that have all embedded memory, external memory, and peripherals located on the data bus and instruction bus of the CPUs. The microcontroller has two cores, PRO_CPU for protocol and APP_CPU for the application, and the address space for both data and instruction bus is 4GB, and the peripheral address space is 512KB. The ESP32 can use either the internal Phase Lock Loop (PLL) of 320MHz or an external crystal or oscillating circuit as a clock source. There are several low-power clocks, including the internal RTC_CLK, and four 64-bit timers for generic purposes. The microcontroller supports TCP/IP, full 802.11 b/g/n/e/i WLAN MAC, and Wi-Fi Direct specifications and can operate as a station and be connected to the internet or server and access point to provide a user interface to a mobile application. Additionally, the esp32 supports Bluetooth 4.2 BR/EDR and BLE and operates under various power modes, including active mode, modem-sleep mode, light and deep-sleep modes. It also provides a number of interfaces, such as Ethernet MAC Interface, SD/SDIO/MMC Host Controller, three UART interfaces up to 5Mbps, two I2C bus interfaces, and more.

- **Security:** Contains multiple protocols used in high-level security including MQTT and secured by TLS. It also has hardware accelerators for AES, SSL, and TLS.
- **Hardware Specifications:** We used the esp32 DOIT DevKit v1 with 30 pins and the esp32 Wrover Module with the camera attachment.
    - Wireless connectivity: Wi-Fi – 150.0 Mbps data rate with HT40
    - Bluetooth: Bluetooth Low Energy (BLE) and Bluetooth Classic
    - Processor: Tensilica Xtensa Dual-Core 32-bit LX6 microprocessor running at 160-240 MHz.
    - ROM: 448KB
    - SRAM: 520KB
    - Low Power: ensures that you can still use ADC conversions, for example, during deep sleep.
- **Peripheral Input/Output:** peripheral interface with DMA includes capacitive touch, Analog-to-Digital Converter, Digital-to-Analog Converter, Inter-Integrated circuit, Universal Asynchronous Reciever/Transmitter, Serial Peripheral Interface, Integrated Interchip sound, Reduced Media-Independent interface, and Pulse-With Modulation

The temperature sensor was chosen due to it having one data line to communicate with the esp32. The TDS sensor works best for water quality readings as it measures salts, minerals, metals, and other dissolved solids and produces a value that can determine the total amount of dissolved solids water. A stepper motor is used for the food dispenser because it is small and easily programmable in setting the feeding frequency. The esp32 Wrover module is used only for the camera attachment, and this was chosen because it is built into the microcontroller, there is no need to attach an external device.

### *B. Accessing the Data*

When researching what other smart fish tanks designs were using to access the data received by the microcontroller, we first came across Google Firebase. This was thought to be a good interface for out web app because we can write our sensor readings to a Firebase storage bucket if we have the API key and bucket tokens. This worked, but we wanted a live stream video and real time data readings, not just occasional snapshots of each. Another issue with Firebase is it slowed down the function of our code and required much more memory availability. We decided to use one esp32 as a webserver. Once the esp32 gathers all its data readings, it produces a DNS. If the user types in the DNS into a web browser, everything is available to the user now.

The esp32 contains a flash memory, which is accessed through SPIFFS (Serial Peripheral Interface Flash File System). It is implemented through the software and can be used on any SPI connected memory component. This allowed us to store html files, images, and data on the esp32. While the ep32 only has 520 KB of SRAM, it can support up to 16MB. This is not exactly ideal. For future work on this prototype will require more data acquisition and machine learning which takes up more space. Moving to an SQL server may be beneficial or explore Firebase deeper.

### *C. Modifying the Code*

Since our project isn't based on a single project currently out there, we now must research each component separately. There is code for all the sensors we used, although we had to modify it in order to connect to Wi-Fi and our webserver. Other segments of code we found provided useful ways to implement Wi-Fi connection, but they were using this code with different sensors. Overall, our final code became a mixture of correct Wi-Fi and webserver code modified with the correct sensors. The Arduino code utilizes libraries already out there to work with our esp32 microcontrollers and our sensors. Another problem that needed to be fixed is choosing different pin numbers on the esp32 because the provided ones will not work if other components are connected to it.

## III. METHODS (COMPONENTS OF DEVICE)

As previously stated, there are six main functions that we are trying to achieve. The sensors were the first step as it required the least amount of preparation to implement. The most challenging task associated with the sensors is not how to get data, it is how to access that data. Next was to get the

camera to constantly record the tank in real time and be able to access it. Last step was the motor and the LEDs. There are many factors to be considered with the motor, including, a way to set the frequency, a way to store and dispense food, and wire it correctly to the esp32. The LEDs came last as it is not necessary for the operation of our smart fish tank. The webserver was implemented towards the end of the project, but it works well in parallel with the setup of all our components.

A challenge of the design is attaching all the components onto one microcontroller. The esp32 microcontroller has two separate analog-to-digital converters (ADCs) that can be used to read analog signals. ADC1 is connected to some of the pins used by the WiFi.h library, while ADC2 is connected to other pins.

When using the WiFi.h library, it is recommended to use only the ADC1 pins for analog input. This is because the ADC2 pins are also used for other purposes, including controlling the built-in hall effect sensor and the ultra-low-power co-processor on the ESP32. If both ADC2 and WiFi are used simultaneously, they can interfere with each other, resulting in inaccurate or inconsistent readings. To avoid this issue, it is best to use only the ADC1 pins (GPIO32 to GPIO39) for analog input when also using the WiFi.h library. This ensures that the analog input and WiFi functions do not interfere with each other, providing more reliable and accurate readings. It is worth noting that this limitation only applies when using the WiFi.h library. If you are not using WiFi, you can use either ADC1 or ADC2 pins for analog input without any issues.

### *A. Temperature Data*

The DS18B20 Temperature sensor was the first component attached to the esp32. It reads the water temperature and outputs it in both Fahrenheit and Celsius. The DS18B20 is a digital temperature sensor manufactured by Dallas Semiconductor (now owned by Maxim Integrated). It is a popular and widely used temperature sensor in the electronics and IoT industries due to its high accuracy, low cost, and simplicity of use. The DS18B20 uses a single wire communication protocol called OneWire, which allows multiple devices to be connected to a single digital pin of a microcontroller. Each device has a unique 64-bit address that can be used to communicate with it individually. This feature makes the DS18B20 a popular choice for temperature monitoring in projects that require multiple temperature sensors to be used simultaneously. The DS18B20 can measure temperatures ranging from -55°C to +125°C with an accuracy of ±0.5°C over a temperature range of -10°C to +85°C. It provides temperature readings in 9 to 12-bit resolution, which can be configured by the user. The sensor is also highly resistant to noise and interference, making it suitable for use in harsh environments.

To use the DS18B20, the sensor is connected to a microcontroller using a single digital pin and the OneWire communication protocol. The microcontroller sends commands to the sensor to initiate temperature measurements and receive the resulting temperature data. This data can then be processed and used for a wide range of applications, including temperature monitoring, climate control, and industrial process control. The DS18B20 temperature sensor uses a digital measurement technique to measure temperature. It contains a temperature sensing element (a bandgap temperature sensor) that converts temperature changes into a voltage signal. This voltage signal is then converted into a digital value using an analog-to-digital converter (ADC) built into the sensor. The DS18B20 communicates with a microcontroller using the OneWire communication protocol. This protocol allows multiple devices to be connected to a single digital pin of a microcontroller, with each device having a unique 64-bit address. To read the temperature from the DS18B20, the microcontroller sends a series of commands to the sensor over the OneWire bus. These commands include commands to initiate temperature conversion, read the temperature value, and control the resolution of the temperature value. Once the temperature conversion is initiated, the DS18B20 converts the analog voltage signal from its temperature sensing element into a digital value using its built-in ADC. The digital temperature value is then stored in the sensor's internal memory, where it can be accessed by the microcontroller. The microcontroller reads the temperature value by sending a read command to the sensor and receiving the temperature value over the OneWire bus. The temperature value is then processed and used in the desired application. The DS18B20 temperature sensor is highly accurate, easy to use, and compatible with a wide range of microcontrollers and development boards. It is commonly used in temperature monitoring and control applications, including HVAC systems, industrial process control, and home automation.

In our prototype the sensor is powered from an external power supply, and it has three wires that need to be attached, Vdd, GND, and the DQ communication wire. Figure 1 displays the circuit diagram of the temperature sensor and the esp32. We used pin 4 on the esp32 in series with the DQ wires, a resistor, and the power supply.

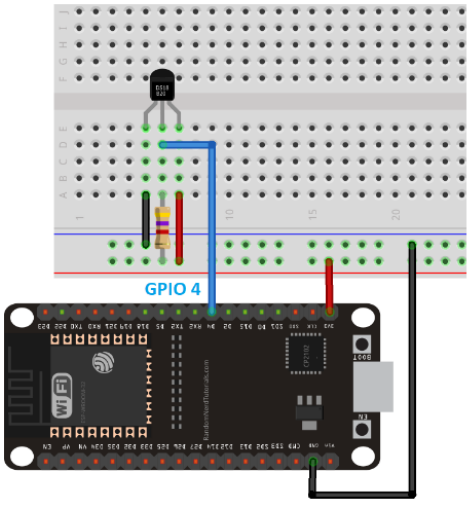


Figure 1: Circuit diagram with esp32 microcontroller and the temperature sensor

Due to its unique 64-bit serial code it can read multiple sensors using one wire, so it has the ability to be implemented across a system of aquariums. We were successful in getting correct and live temperature data which can be seen in our webserver.

### *B. Water Quality Data*

The Total Dissolved Solids (TDS) water quality sensor was the next component attached to the fish tank. This sensor reads water quality in units of total dissolved solids in ppm (mg/L). The TDS Meter V1.0 from keystudio is a water quality testing tool that is designed to measure the Total Dissolved Solids (TDS) in water. It is commonly used in aquaculture, hydroponics, and other applications where the quality of water is critical. The TDS Meter V1.0 uses a sensor called a TDS probe to measure the TDS in water. The TDS probe contains two electrodes that are submerged in the water being tested. When a voltage is applied across these electrodes, the dissolved solids in the water create an electrical current that is proportional to the TDS level. The TDS Meter V1.0 uses a microcontroller to measure the electrical current generated by the TDS probe and convert it into a TDS value. The TDS value is then displayed on an LCD screen, allowing the user to easily read the TDS level of the water being tested. Overall, the TDS Meter V1.0 is a simple and effective tool for measuring the TDS level in water. It provides accurate and reliable measurements, making it a valuable tool for anyone who needs to monitor the quality of water in their environment. As a result, this sensor outputs a number in ppm. The range we want for freshwater aquariums is 70-140ppm, so this sensor more than reaches that criteria at it can read up to 1000ppm.

Like the DS18B20 sensor, this only has one data wire, and we connected this to GPIO pin 34 since it's an ADC pin. The Arduino code we used had multiple steps to get an accurate value of the water quality. First, we define a reference voltage equal to the ADC pin voltage at 3.3V. The code then runs a median filtering algorithm to filter out noise on the analog values collected. This sensor gets a reading every 40 milliseconds. The filtered analog values are then converted to voltage values, and temperature compensation is applied to the voltage values. The temperature compensation considers the current temperature to adjust the voltage obtained from the sensor to get an accurate measurement. It is calculated by first finding the compensation coefficient.

$$compensationCoefficient = 1.0 + 0.02 * (temp - 25.0) \quad (1)$$

If we take the average voltage and divide it by this coefficient, then we will get our correctly compensated voltage. The last step in the algorithm is to convert the voltage readings into TDS values. The equation we used is derived from the relationship between electrical conductivity of a solution and its total dissolved salts. Computation voltage is denoted by *Vc*.

$$tdsValue=((133.42*Vc^3)-(255.86*Vc^2)+(857.39*Vc))*0.5 \quad (2)$$

This TDS value is what we are trying to measure. We testing it in our fish tank we got a reading that reflected our expected water quality measurement. The ppm is being constantly measured and is displayed in our webserver.

### *C. Camera Module*

The third component of our fish tank is the camera used for the live stream. This is the only component not connected to the esp32 Devkit V1, and instead it is its own component attached to the esp32 Wrover Module. The ESP-CAM AI Thinker module is a compact and affordable device that combines an ESP32-S chip and a camera module. ESP32-S is a smaller and more compact version of the ESP32 chip, with a reduced number of external pins and a smaller footprint. The ESP32-S is designed to be used in space-constrained applications where the size of the module is critical, such as smartwatches, fitness trackers, and other wearable devices. It is designed to provide an easy-to-use platform for developing AI and computer vision applications, such as face recognition, object detection, and image classification. The video streaming web server feature allows the module to capture video in real-time from the camera and stream it over a Wi-Fi network to any device with a web browser, such as a smartphone or a laptop. This feature makes it possible to monitor the camera's field of view remotely and in real-time, enabling a wide range of applications such as home security, monitoring of production lines, and environmental monitoring. To use the video streaming web server feature, you need to connect the ESP-CAM module to a Wi-Fi network and start the web server. You can then access the video stream from any device on the same network by typing the IP address of the module in a web browser. The video stream can be customized by adjusting parameters such as resolution, frame rate, and image quality to suit your specific application requirements.

1. First, the program defines some constants for the content type, boundary, and part of the stream. These will be used later in the code.
2. The stream_handler function is defined as the HTTP request handler. This function gets called every time a client connects to the server and requests the video stream.
3. In the stream_handler function, the content type is set to multipart/x-mixed-replace using httpd_resp_set_type.
4. The while loop runs continuously while the client is connected to the server. Inside the loop, the camera captures an image using esp_camera_fb_get.
5. The image is then checked for its width and format. If the width is greater than 400 pixels and the format is not already in JPEG format, the image is converted to JPEG using frame2jpg. If the conversion fails, the program sets the error status to ESP_FAIL.
6. If the image is successfully captured and converted to JPEG, its length and buffer are stored in _jpg_buf_len and _jpg_buf, respectively.

7. If the image capture and conversion was successful, the content of the HTTP response is sent in three parts. First, the part header is sent with the length of the JPEG data. Then, the JPEG data is sent. Finally, the boundary is sent to indicate the end of the part.
8. After sending the HTTP response, the function checks if there are any errors. If there are, the loop is broken, and the function returns the error status. Otherwise, the loop continues.
9. The startCameraServer function sets up the HTTP server on port 80 and registers the stream_handler function to handle incoming requests.

The code in figure 2 sets up an HTTP server on the esp32' that streams JPEG images from the camera in real-time over the network using the multipart/x-mixed-replace protocol.

### *D. Motor*

The motor in our design is what dispenses the food to the tank. We 3D printed a model that has a motor attached to a spiral that will spin the food out when the motor is running. The motor has code implemented to its driver that controls when the motor turns on and how much it rotates. In the Arduino code, we use PWM output to control the ULN2003 motor driver pins connected to the stepper motor. This allows you to control the current flowing through the motor coils, and therefore the motor's speed and position. Note that the 28BYJ-48 stepper motor requires a specific sequence of control signals to step properly. This is often referred to as the "wave drive" sequence, and it can be implemented using the ULN2003 motor driver's IN1, IN2, IN3, and IN4 pins. The exact sequence of signals depends on the motor's configuration, but it can typically be found in the motor's datasheet or online tutorials. The ULN2003 motor driver contains seven Darlington transistor pairs, which act as switches to control the current flowing through the motor's coils. The driver also provides protection against back electromotive force (EMF), which can be generated when the motor is decelerating or stopped and can potentially damage the microcontroller or other components.

```
}
if(res == ESP_OK){
  size_t hlen = snprintf((char *)part_buf, 64, _STREAM_PART, _jpg_buf_len);
  res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
}
if(res == ESP_OK){
  res = httpd_resp_send_chunk(req, (const char *)_jpg_buf, _jpg_buf_len);
}
if(res == ESP_OK){
  res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY, strlen(_STREAM_BOUNDARY));
}
if(fb){
  esp_camera_fb_return(fb);
  fb = NULL;
  _jpg_buf = NULL;
} else if(_jpg_buf){
  free(_jpg_buf);
  _jpg_buf = NULL;
}
if(res != ESP_OK){
  break;
}
```

Figure 2: Code for HTTP server

### *E. LEDs*

The purpose of the LEDs on our design is to indicate to a user that there are unusual measurements and if the user needs to take action. PWM is an efficient and effective way of controlling the brightness of an LED. By adjusting the pulse width and frequency, the LED can be dimmed or brightened as needed, making it a versatile and useful component in a wide range of applications. An NPN transistor is often used in conjunction with a RGB LED and a microcontroller that generates PWM signals. This is because the microcontroller's output pins may not be able to supply enough current to drive the RGB LED directly, especially if it is a high-power LED or if multiple LEDs are connected in parallel.

### *F. Webserver*

The data received from our prototype is displayed on the webserver implemented on the esp32. This utilizes the library ESPAsyncWebServer.h and ESPmDNS.h. The first library provides a way to create the web server o the microcontroller using the asynchronous communication protocol. The ESPmDNS.h library provides a way to register a hostname on the local network using Multicast DNS (mDNS). It enables devices on the network to access the esp32 using a domain name rather than an IP address.

The languages we used for the webserver was HTML and JavaScript. HTML GET and POST are two methods used to send data from a web page to a server. GET is a method of requesting data from a server by appending data to the end of a URL in the form of query parameters. The data is visible in the URL and can be bookmarked or shared. GET requests are typically used to retrieve data from a server, such as web pages, images, or other resources. However, they are not recommended for transmitting sensitive or private data because the data is visible in the URL and can be intercepted by third parties. POST, on the other hand, is a method of sending data to a server as part of the request body. Unlike GET, the data is not visible in the URL and cannot be bookmarked or shared. POST requests are commonly used for transmitting sensitive or private data, such as login credentials, credit card information, or personal information. However, they can also be used to submit form data or to send other types of data to a server. In summary, GET and POST are two different methods of sending data between a web page and a server. GET is used for requesting data from a server, while POST is used for submitting data to a server. The choice between using GET and POST depends on the type of data being sent and the security requirements of the application.

AJAX (Asynchronous JavaScript and XML) is a technique used in web development to create interactive web applications. It allows a web page to update parts of its content without requiring a full page refresh, making the user experience faster and smoother. Traditionally, web pages are static and require a full page reload every time the user interacts with them. AJAX enables dynamic content loading by making asynchronous requests to the server to retrieve the

data in the background without reloading the entire page. This means that only the specific part of the web page that needs updating is refreshed, making the process faster and more efficient. AJAX is commonly used in conjunction with JavaScript, which enables the processing and manipulation of the retrieved data on the client-side, without the need for additional server requests. In addition, AJAX can also work with other data formats besides XML, such as JSON, HTML, or plain text. Overall, AJAX provides a powerful tool for creating interactive, responsive, and efficient web applications that can enhance the user experience by minimizing page reloads and reducing server load. The webserver as well as the Wi-Fi manager can be seen in figure 3.

## IV. Experiments

The experiments we conducted verify the prototype is working correctly. The first step was to test the sensors accuracy of the readings. In order to test this, we got water from a fridge set at 70°F. After putting our sensor in the water, the webserver was displaying the correct temperature. For the water quality reading we didn't have a baseline ppm of the water we were testing, so instead we slowly added salt to the water to in order to verify it would detect the change. This graph as well as the temperature reading can be seen in figure 4. We also have to test the webserver correctly sets the desired temperature and ppm. We put ice cubes in water to drop the temperature below the desired range, and the LED lights did indicate that there was an anomaly. The water quality sensor settings were tested in a similar way. The camera code is built onto the Arduino IDE, so the only testing we did with the camera was calibrating the setting to get the best image. The stepper motor is the last part that needed to be tested. On the webserver we set the feeding frequency to 10 times per day. We monitored the motor to confirm it dispenses food when it is supposed to and it did. The other testing that we are currently conducting on the prototype is to make sure that it continues to operate correct after a long period of time.


Figure 3: Display of the Wi-Fi Manager and the settings page on the web server.

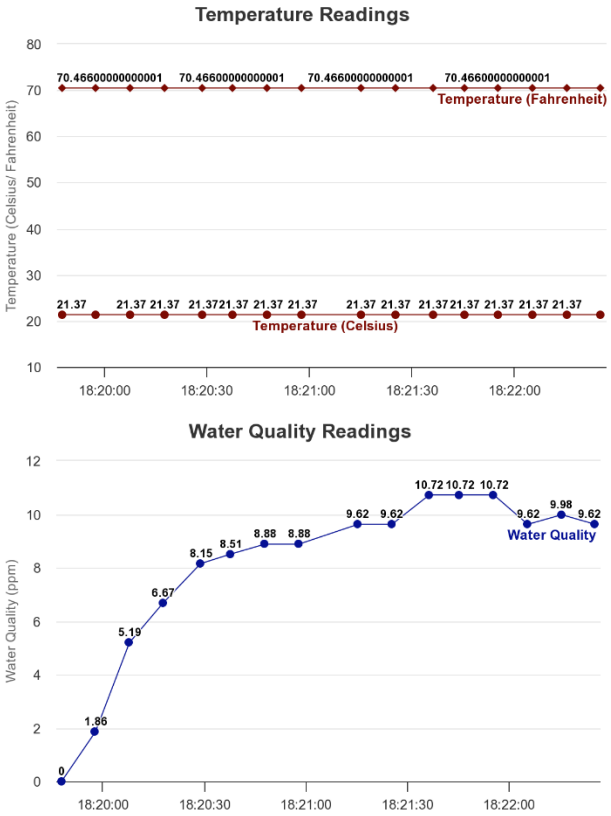


Figure 4: Water temperature data and the water quality readings

## V. Conclusion and Future Work

This was a successful implementation of an automated fish tank. The webserver is live and the temperature, ppm, and live stream are all functioning. The settings for the motor, LED, and temperature configuration are also working correctly. The sensors were tested and they record live data accurately, as well as an active live stream. Overall, the prototype is operational and we implemented all the components we wanted on a PCB board. There is still more that can be improved on our prototype. As we move forward we will have more data acquisition and this is not ideal since the esp32 is small in storage. This means we may have to move from a webserver on the esp32 and explore firebase further or an SQL server. Other future developments we want to add is machine learning, and possibly add an external TPU. Currently, our motor gets hot quickly so to avoid it overheating we may want to move to a stepper motor with less RPM or a heat sink with a motor. Last, we want to send alerts to the user's email if it detects something is wrong. The LEDs work for now but an email notification sent to the user's phone would be a nice addition. The goal of the project was to create an autonomous aquatic monitor that provides information and alerts about the condition of the aquarium, and the smart features to be customizable. We achieved our goal and have a successful design of an embedded aquatic monitor using the esp32.

## REFERENCES


[1] 1] Maier, Alexander, Andrew Sharp, and Yuriy Vagapov. "Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things." *2017 Internet Technologies and Applications (ITA)*. IEEE, 2017.

[2] [2] Babiuch, Marek, Petr Foltýnek, and Pavel Smutný. "Using the ESP32 microcontroller for data processing." *2019 20th International Carpathian Control Conference (ICCC)*. IEEE, 2019.

[3] [3] Gatial, Emil, Zoltán Balogh, and Ladislav Hluchý. "Concept of energy efficient ESP32 chip for industrial wireless sensor network." *2020 IEEE 24th International Conference on Intelligent Engineering Systems (INES)*. IEEE, 2020.

[4] Santos, Rui. "Guide for DS18B20 Temperature Sensor with Arduino." *Random Nerd Tutorials*, 16 July 2019, https://randomnerdtutorials.com/guide-for-ds18b20-temperature-sensor-with-arduino/.

[5] "KS0429 Keyestudio TDS Meter v1.0." *KS0429 Keyestudio TDS Meter V1.0 - Keyestudio Wiki* ,https://wiki.keyestudio.com/KS0429_keyestudio_TDS_Meter_V1.0.

[6] "Controlling Stepper Motor with l293d." *Instructables*, Instructables, 23 Aug. 2017, https://www.instructables.com/Controlling-Stepper-Motor-With-L293D/.

[7] "Speed Control of DC Motor Using Pulse Width Modulation." *ElectronicsHub*, 5 Mar. 2023, https://www.electronicshub.org/speed-control-of-dc-motor-using-pulse-width-modulation/.