

Real-Time Subsea Turbulence and Sediment Monitoring System Final Report

Prepared for: Dr. Boris Bam Nges

Abdulla Sadoun B00900541

Sohaib Zahid B00908166

Table of Contents

Table of Contents.....	1
Table of Figures.....	3
Executive Summary.....	4
1. Introduction.....	5
1.1. Problem Statement.....	5
1.2. Project Objectives.....	6
2. System Architecture and Design.....	7
2.1. Hardware Components.....	7
Nucleo F411RE: STM32F411 microcontroller.....	7
MS5837-30BA Pressure Sensor.....	8
SEN-HZ21WA Flow Rate Sensor.....	8
HTI-96MIN Acoustic Sensor.....	9
DS18B20 Temperature Sensor.....	9
Power System Design:.....	10
2.2. Software Architecture.....	11
3. Implementation Details.....	13
3.1. Sensor Integration.....	13
3.2. Power Needs Analysis.....	14
3.3. Data Processing Pipeline.....	15
- Real-time data acquisition methods - Processing algorithms - Alert generation system - Data visualization implementation.....	15
Real-time Data Acquisition Methods.....	15
Processing Algorithms.....	16
Alert Generation System.....	16
Data Visualization Implementation.....	16
4. Testing and Validation.....	19
4.1. Test Methodology.....	19
4.2. Results and Analysis.....	20
5. Cost Analysis and Economic Viability.....	22
5.1. Component and System cost.....	22
5.2. Comparison with existing solutions.....	23
6. Future Improvements and Scalability.....	25
6.1. Proposed Enhancements -.....	25
6.1.1. Additional sensor integration possibilities.....	25
6.1.2. Wireless connectivity options.....	25
6.1.3. User interface improvements.....	25
6.1.4. Fault and anomaly detection and handling.....	25
6.1.5. Advanced data analysis capabilities.....	25
6.2. Scalability Considerations -.....	26

7. Conclusions.....	27
7.1. Summary of achievements and outcome.....	27
7.2. Lessons learned.....	27
Supporting Information and Appendices.....	28
A. Project Timeline and Milestones.....	28
B. Technical Component Datasheets and Tools Specifications.....	28
C. Other Resources.....	29

List of Diagrams, Tables and Figures

Nucleo F411RE Board with our STM32F411 Chip.....	7
Overview of our system Architecture (from Lab 4).....	7
MS5837-30BA Pressure Sensor.....	8
SEN-HZ21WA Flow Rate Sensor.....	9
HTI-96MIN Acoustic Sensor.....	9
DS18B20 Temperature Sensor.....	10
Overview of our system's electrical Connections.....	10
Overview of our system's Structure and flow.....	11
System's Serial Logs 1.....	16
System's Real Time Generated Graphs.....	17
System's Functions that simulates Real Time Data based on Real Conditions and Scenarios.....	19
System's Real Time UI Prior to Final Update (No option to set thresholds in real time after execution)...20	
Table highlighting system prototype cost.....	22
System's Timeline and milestones gantt chart.....	28

Executive Summary

The Real-Time Subsea Turbulence and Sediment Monitoring System addresses critical challenges in marine environmental monitoring through an embedded systems solution. Current methods of underwater sediment and turbulence monitoring are usually costly, hard to use, and often result in delayed detection of potential environmental hazards. Our project presents a comprehensive solution that leverages real-time data collection and processing capabilities to provide immediate insights into underwater conditions.

This project successfully meets its objectives of creating a deployable, low-cost, user-friendly, low-power, scalable, durable and open-source real-time monitoring system that can effectively track and analyze underwater environmental conditions, representing a significant step forward in marine ecosystem protection and management.

1. Introduction

1.1. Problem Statement

Marine ecosystems face significant challenges due to the lack of efficient, low cost, low power monitoring systems for underwater sediment levels and turbulence. The current state of underwater environmental monitoring presents several critical challenges that impact both the marine environment and our ability to protect it effectively along with using data for analysis to evaluate risk of environmental changes. .

The challenge lies in the inefficient monitoring of underwater sediment levels and turbulence due to the preexisting solutions either having high costs, high power requirements, lack of user friendly interface and ease of use and adaptability along with integration into current workflow and systems. Monitoring systems currently often rely on manual data collection and periodic sampling, which creates gaps in acquired data, limits and our ability to detect rapid changes in underwater conditions not to mention that these old systems and methods come with a cluster of problems and inaccuracies due to human error, high reaction and response speed as well as low frequency of sampling and lack of modularity/compatibility with pre-existing systems while still maintaining a scalable and elastic solution. These problems are highlighted when dealing with dynamic underwater harsh environments where conditions can change rapidly and affect marine ecosystems, the environment and acquired data.

Delayed detection of environmental hazards represents another significant concern. Having a time gap between the process of collecting and analyzing the data introduces numerous issues and lost opportunities as well as avoidable disasters or negative impacts that either could've been avoided or capitalized upon for the sake of the greater good, this is a key concept and idea when it comes to analyzing data in a rapidly changing environment.

Furthermore, the limitations in real-time data collection capabilities present a challenge when it comes to effective environmental management and environmental disaster/risk mitigation. Without continuous, real-time monitoring, it becomes extremely difficult to track sudden changes in underwater conditions that can affect marine/human lives. It would also be hard to identify patterns in sediment movement and accumulation which can aid in detecting and responding to sudden changes in turbulence that can point towards environmental disturbances. This also means that detrimental time bound decision making becomes unnecessarily difficult.

Manual monitoring methods are not only time-consuming and redundant but are economically impactful and hard to maintain for prolonged periods.

Addressing these challenges requires an innovative approach that combines real-time monitoring capabilities, cost-effectiveness, power-efficiency, reliability, ease of use, integration and scalability. Our Real-Time Subsea Turbulence and Sediment Monitoring System aims to directly address these issues through a scalable, user-friendly, and efficient solution that serves as the skeleton and base of tackling all the previously mentioned problems.

1.2. Project Objectives

Our Real-Time Subsea Turbulence and Sediment Monitoring System was developed with the end-user in mind. Our objectives were carefully designed and selected to create a comprehensive and open-ended solution that balances technical capabilities with practical implementation requirements giving the end user the power to easily integrate, set parameters and build upon our solution as they please.

Real-Time Data Collection and Processing serve as the base objective focusing on creating a robust real-time monitoring capability through simultaneous data collection and analysis from multiple underwater sensors. The system is designed to achieve the following goals and performance metrics including:

- Sensor sampling rates of 200 ms per sensor, ensuring high-frequency data collection
- Data processing completion within 500 ms cycle time, enabling near-instantaneous analysis
- Real-time graph updates to facilitate rapid trend analysis and pattern recognition
- Efficient data transmission through UART protocols for immediate access to environmental information
- User defined Parameters and critical thresholds
- Immediate notification generation when thresholds are exceeded
- Accompanied by friendly User Interface to edit/select parameters, view alerts and live-graphs
- Modules' Sensor Cluster and platform can be deployed and run at full power for a minimum of 5 continuous days without intervention while maintaining an easily swappable power solution if their use case requires more/less run time.
- Maintaining a low-cost system design with a total component cost of approximately \$242

Our monitoring capabilities are aimed for general purpose underwater monitoring so we aim to cover the basic array of essential sensors to get the user started and they can easily integrate more sensors as they please. Our base prototype would be shipped with the following measuring capabilities:

- Turbulence level monitoring through acoustic data collection using the HTI-96MIN sensor
- Sediment concentration tracking via pressure measurements from the MS5837-30BA sensor
- Water flow pattern analysis using the SEN-HZ21WA flow sensor
- Temperature monitoring via the DS18B20 sensor for comprehensive environmental assessment

These objectives serve as our systems roadmap and don'ts only addresses the technical challenges of underwater monitoring but also considers practical aspects such as cost-effectiveness, power efficiency, and operational reliability as well as ease of use.

2. System Architecture and Design

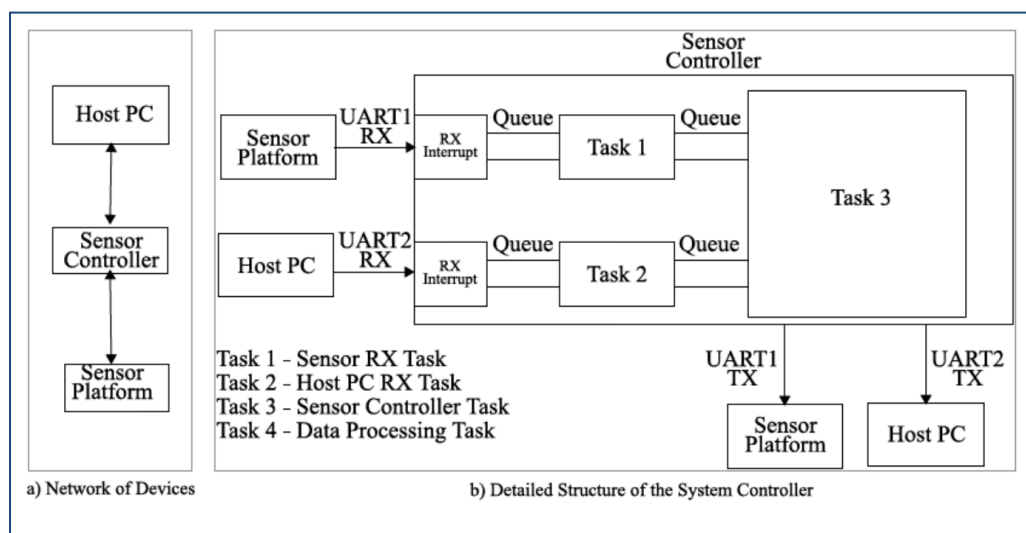
2.1. Hardware Components

Nucleo F411RE: STM32F411 microcontroller

Our Real-Time Subsea Turbulence and Sediment Monitoring System is built around a carefully selected set of hardware components that work together to provide comprehensive environmental monitoring capabilities. The Nucleo F411RE board equipped with the STM32F411 microcontroller is the primary processing unit in our system. Our design would require two (2) of these boards as one would be used for communication and managing tasks between the host PC and the sensor platform (the other board(s)). The sensor platform(s) are boards that would have the sensors connected to them, housed in an underwater housing and aim to live well below the sea level. Each Sensor Controller Board can support connection and communication with up to 11 sensor platforms.



Nucleo F411RE Board with our STM32F411 Chip



Overview of our system Architecture (from Lab 4)

Other than scalability this board has also been chosen for this task due to its wide range of real-time processing capabilities and extensive peripheral support. It supports I2C, SPI, UART, ADC and various other communication protocols giving it endless possibilities when it comes to compatible sensors and their configurations. The STM32 family of microcontrollers are extensively used in the industry for various real-time systems applications from soft, firm to even hard real-time systems. We can find this product or a very similar competitor in almost every single automotive vehicle on the market today.

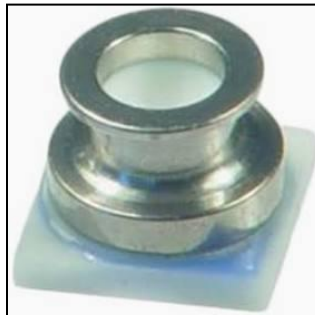
Additionally this microcontroller can be run with freeRTOS which would serve as the core of our software build.

Power is supplied through a 6000mAH power bank to the platform unit, which was selected based on detailed power requirement analysis to support extended deployment periods. The controller unit would not need an external power source as the usb connection intended to be used with the Host PC would provide sufficient power for its operation and our use case.

As for our sensor Array Integration: The system incorporates four specialized sensors, each serving a specific monitoring function:

MS5837-30BA Pressure Sensor

- Operating capability up to 300 meters below sea level and 30 bar pressure
- I2C protocol implementation for reliable data communication
- Current draw of 1.2mA at 3.3V
- Essential for depth and sediment monitoring



MS5837-30BA Pressure Sensor

SEN-HZ21WA Flow Rate Sensor

- Hall Effect-based water flow measurement
- Simple GPIO pulse-based communication (regular DC digital pin required)
- Current consumption of 15mA at 3.3V
- Critical for water movement analysis



SEN-HZ21WA Flow Rate Sensor

HTI-96MIN Acoustic Sensor

- Low-frequency wave detection for current and sediment movement monitoring
- ADC pin interface for data acquisition (Analog to Digital Converter pin required)
- Current requirement of 2-3mA at 3.3V
- Enables turbulence detection and analysis



HTI-96MIN Acoustic Sensor

DS18B20 Temperature Sensor

- One-wire protocol implementation
- Minimal power consumption at 0.5mA
- Provides essential environmental temperature data

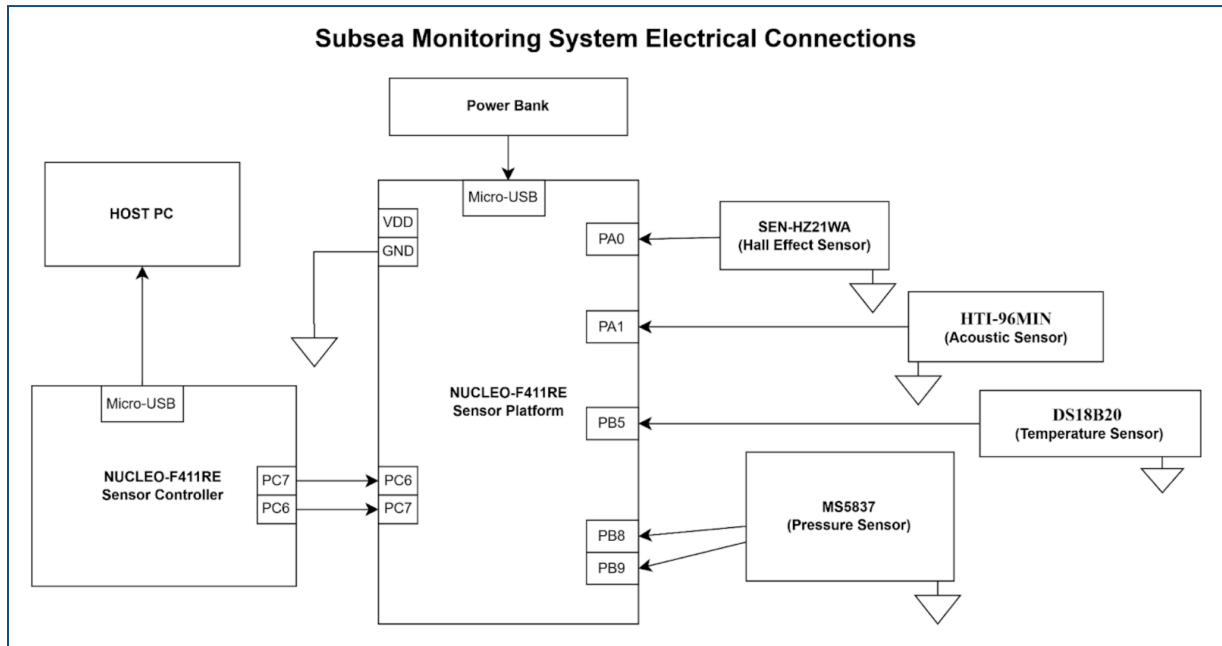


DS18B20 Temperature Sensor

Power System Design:

The power system was engineered based on comprehensive current requirements analysis:

- Total sensor array current draw: 20mA at 3.3V
- Microcontroller base operation: 30mA at 5V
- Combined system current requirement: approximately 50mA
- Power bank selection: 6000mAH capacity supporting 5-day continuous operation

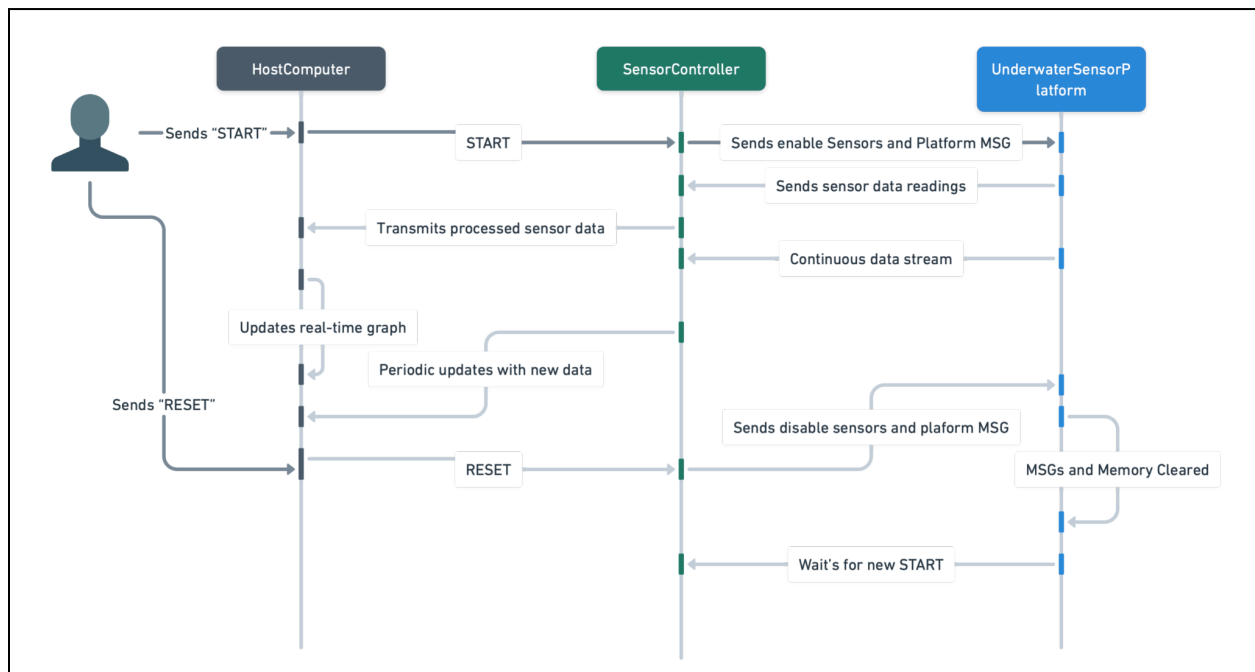


Overview of our system's electrical Connections

2.2. Software Architecture

The software architecture is built primarily on FreeRTOS for data collection and real-time communication between boards and PC, implementing a sophisticated task management system that ensures efficient real-time operation and reliable data handling. On the Host Computer Tkinter (a python framework and library) is used in conjunction with PYserial to communicate with Sensor Controller board or module through the COM ports through a serial connection at the pre-specified baud rate of 115200.

The flow of our software can best be described using the subsequent Software Structure/flow diagram which is often used to aid in visualizing API requests and endpoints at different services or modules within a software or system. In our case our program works in a very similar manner that can be visualized through this diagram



Overview of our system's Structure and flow

Task Management Structure: The system implements four primary FreeRTOS tasks:

1. Sensor Data Acquisition: Manages the timing and collection of data from all sensors
2. Data Processing: Handles the analysis and interpretation of collected sensor data
3. Alert Generation: Monitors thresholds and generates appropriate alerts
4. Communication Control: Manages data transmission through UART channels

Data Flow and Communication: The software architecture implements a queue-based data management system that ensures efficient handling of sensor data and system communications:

- UART1 dedicated to sensor platform communication
- UART2 handling host PC interface

- Queue-based data management ensuring proper data flow between tasks

Processing Pipeline: The software implements a sophisticated processing pipeline that includes:

- Real-time data filtering and validation
- Threshold monitoring and alert generation
- Data formatting for transmission
- Real-time graphing and visualization support with real time alerts for user defined threshold and sensor data parameters.

The entire architecture is designed with modularity in mind, allowing for easy maintenance and future integration of more platform boards or sensors. Reliable and secure operation is guaranteed in our system as it takes advantage of all the methods preventive measures offered by default with freeRTOS to prevent deadlocks and race conditions like timeout mechanisms, interrupt-safe design (ability to distinguish between an interrupt and a task to prevent accessing the same resource simultaneously, regular use semaphore and mutex when accessing shared resources and the queue implementation for a producer-consumer pattern making it impossible to access the same resource for the same purpose (read/write or use) simultaneously.

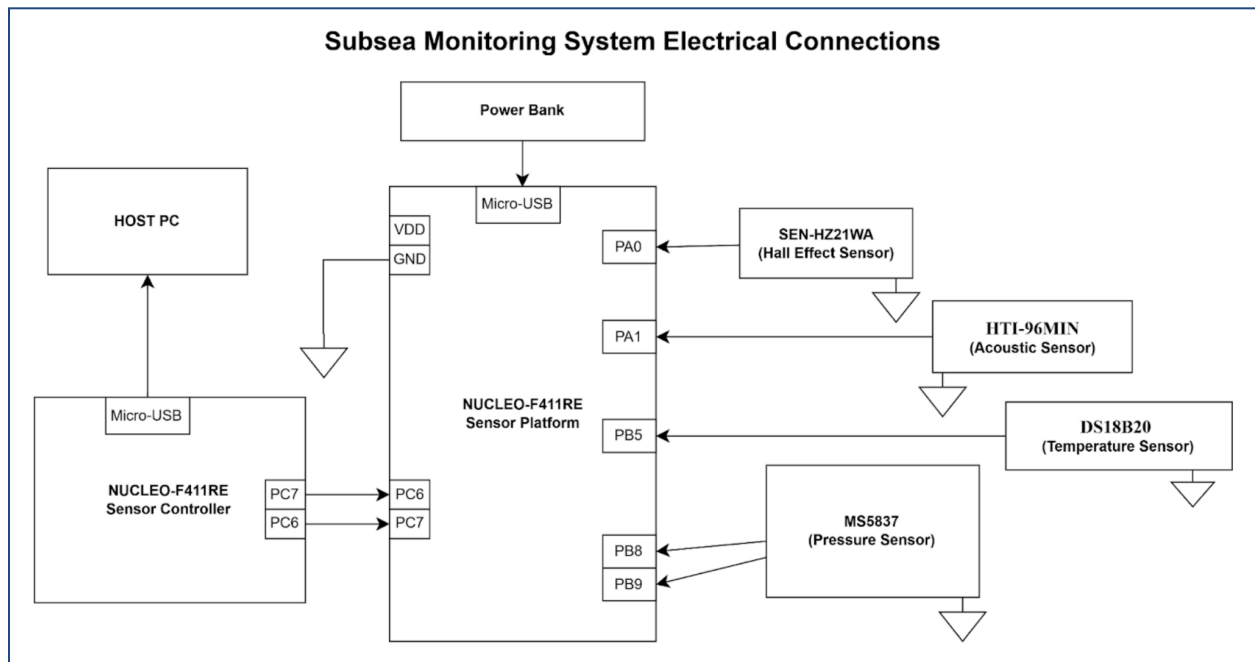
Additionally, our system does not only rely on what freeRTOS has to offer by default and has taken some preventive measures to avoid deadlocks like priority based scheduling where we give tasks the same priority to avoid accessing a shared resource in use by another task due to a priority privilege. We used semaphores for UART mutual exclusion allowing data to only flow one way at a time and controllers wouldn't be able to both access the connection simultaneously.

3. Implementation Details

3.1. Sensor Integration

Due to hardware availability constraints, the system was tested using simulated sensor data designed to replicate the expected ranges and behaviors of the physical sensors. This approach allowed for comprehensive testing of the system's processing capabilities while maintaining realistic data patterns.

We have however included a schematic of how the final system would look like in terms of electrical connections which can be shown in the following diagram schematic.



Overview of our system's electrical Connections

In the diagram above we can see which pins in each board are used up and where they are connected. As for the other settings and configurations for each component we have:

Nucleo-F411RE Boards: The 2+ boards are connected together through their RX and TX pins which will facilitate their UART communication capabilities allowing one of them to be deployed underwater while the other one can control it as well as have multiple other platform boards connected to it for control and scalability.

MS5837-30BA Pressure Sensor: The pressure sensor integration was designed around I2C communication protocol with the following specifications:

- Data acquisition rate: 200ms intervals
- Operating range: 0-30 bar
- Resolution: 0.2 mbar

- Would use up pins PB8 and PB9 which are actually the I2C1SCL and I2C1SDA for our board to facilitate its I2C communication capabilities.
- Simulated data ranges were maintained within expected underwater pressure variations
- Error checking implemented for out-of-range values

SEN-HZ21WA Flow Sensor: The flow sensor implementation utilized GPIO pulse counting with specific characteristics:

- Pulse frequency measurement for flow rate calculation
- Data sampling every 200ms
- Flow rate conversion algorithms implementing manufacturer specifications
- Range validation and error detection systems
- Would use up any of the GPIO-A pins like PA0 to communicate with our board and system

HTI-96MIN Acoustic Sensor: The acoustic sensor data acquisition was implemented through ADC readings:

- Continuous sampling at specified intervals
- Frequency analysis for turbulence detection
- Signal filtering for noise reduction
- Pattern recognition for anomaly detection
- Would use up an ADC pin like PA1 to convert incoming analog signals from the sensor to Digital ones that can be understood by our board and used in our system.
- This sensor was chosen for its waterproof and resistant nature and capabilities

DS18B20 Temperature Sensor: Temperature monitoring implementation followed the one-wire protocol:

- Regular sampling at 200ms intervals
- Temperature range validation
- Error detection and handling
- Calibration compensation algorithms
- Can use any pin in our system like PB5 for communication

3.2. Power Needs Analysis

Power management implementation focused on acquiring the system's electrical power needs for operation while maintaining reliable data collection for the specified operation time of 5 continuous days subsequent to deployment.

Looking at our board's and sensors' power requirements from their datasheets, we have gotten to the following numbers:

- Nucleo F411REMicrocontroller board: Requires a Maximum of 30mA @5V to operate under full load
- MS5837 Pressure Sensor: Requires a Maximum of 1.2mA @ 3.3V to operate under full load
- SEN-HZ21WA Flow Sensor: Requires a Maximum of 15mA @ 3.3V to operate under full load
- HTI-96MIN Acoustic Sensor:: Requires a Maximum of 15mA 2-3mA @ 3.3V to operate under full load
- DS18B20 Temperature Sensor:: Requires a Maximum of 15mA 0.5mA @ 3.3V to operate under full load

In conclusion, Our Sensors would need an approximate total of 20mA @3.3V for operation. Meaning our System would need approximately 50mA of current @5V to guarantee sufficient power and safe operation of the deployable underwater platform unit.

Looking at our requirements, we wanted the system to have 5 days of continuous operation without having to be interrupted or retrieved and redeployed so that the deployment crew would only have to recover the system weekly.

For 5 days of continuous operation at full load (120 Hours) we would need a battery with a capacity of 6000mAH (50mA * 120H) assuming all the sensors would be running at full load.

So we opted for a simple power bank of 6000mAH 5V rating to satisfy this requirement which is extremely cheap, widely available and would be easy to integrate or swap depending on the user's operation times.

3.3. Data Processing Pipeline

- Real-time data acquisition methods - Processing algorithms - Alert generation system - Data visualization implementation

The data processing implementation consists of several key components working together to ensure reliable and efficient data handling:

Real-time Data Acquisition Methods

The system uses UART-based communication defined USART_Driver.c for real-time data collection In it we can see two separate UART channels are implemented, one for sensor data communication (huart6), And the other one (huart2) to host PC communication Interrupt-driven data reception using HAL_UART_RxCpltCallback ensures no data loss FreeRTOS queues (Queue_extern_UART and Queue_hostPC_UART) buffer incoming data for processing

Processing Algorithms

The system implements a multi-stage processing pipeline where message parsing is done by the `parse_sensor_message` function (in `Comm_Datalink.c`), it decodes incoming sensor data using a state machine approach. Data Validation is done through conducting a quick checksum verification to insure integrity, a Checksum of 0 is what would indicate improper data or an error has occurred within the message containing the data somewhere in this pipeline. The data from the sensors is then passed through one of the following functions to determine if there is a concerning value or a threshold is passed

```
analyzeAcousticValue(int acousticValue)
```

```
analyzePressureValue(int pressureValue)
```

```
analyzeHallEffectValue(int hallValue)
```

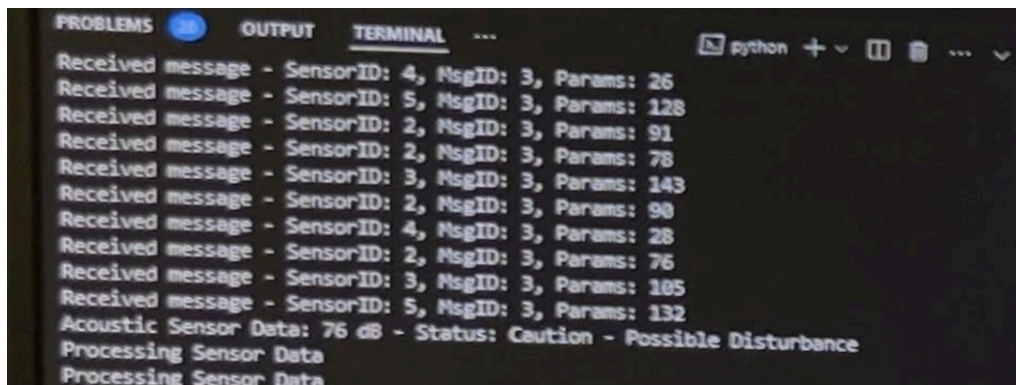
```
analyzeTemperatureValue(int tempValue)
```

Alert Generation System

Threshold-based alert generation for each sensor type is returned by the sensor analysis functions shown and mentioned above according to the user specified and predefined thresholds. Status messages are generated.

Data Visualization Implementation

Real-time data plotting is done through an external python script then doesn't go through freeRTOS and instead focuses on the output of the board which is retrieved from the serial connections terminal from the COM port the controller is using up. The script sends a start message if the system has not started and is in the `_WAIT` state; printing "polling.." in the terminal.



```
PROBLEMS OUTPUT TERMINAL ...
Received message - SensorID: 4, MsgID: 3, Params: 26
Received message - SensorID: 5, MsgID: 3, Params: 128
Received message - SensorID: 2, MsgID: 3, Params: 91
Received message - SensorID: 2, MsgID: 3, Params: 78
Received message - SensorID: 3, MsgID: 3, Params: 143
Received message - SensorID: 2, MsgID: 3, Params: 90
Received message - SensorID: 4, MsgID: 3, Params: 28
Received message - SensorID: 2, MsgID: 3, Params: 76
Received message - SensorID: 3, MsgID: 3, Params: 105
Received message - SensorID: 5, MsgID: 3, Params: 132
Acoustic Sensor Data: 76 dB - Status: Caution - Possible Disturbance
Processing Sensor Data
Processing Sensor Data
```

System's Serial Logs 1

```

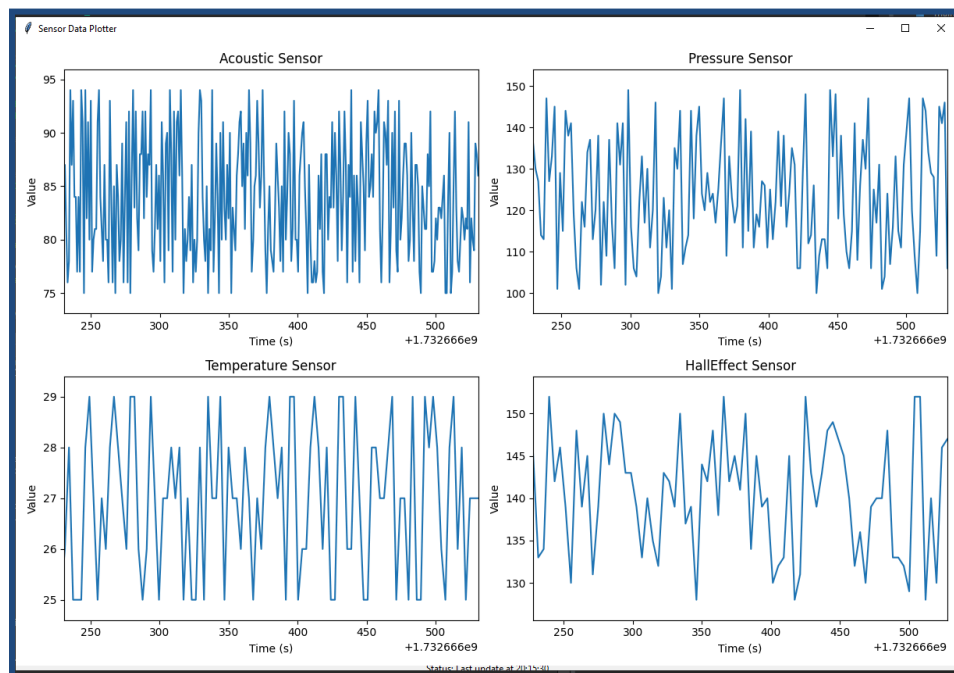
Received message - SensorID: 2, MsgID: 3, Params: 100
Pressure Sensor Data: 106 PSI - Status: Normal Pressure - Pipeline Operating Safely
Processing Sensor Data
Processing Sensor Data
Received message - SensorID: 2, MsgID: 3, Params: 89
Received message - SensorID: 4, MsgID: 3, Params: 25
Received message - SensorID: 2, MsgID: 3, Params: 82
Received message - SensorID: 3, MsgID: 3, Params: 127
Received message - SensorID: 5, MsgID: 3, Params: 138
Received message - SensorID: 2, MsgID: 3, Params: 93
Received message - SensorID: 2, MsgID: 3, Params: 78
Received message - SensorID: 3, MsgID: 3, Params: 110
Received message - SensorID: 4, MsgID: 3, Params: 27
Received message - SensorID: 2, MsgID: 3, Params: 89
Received message - SensorID: 2, MsgID: 3, Params: 79
Received message - SensorID: 3, MsgID: 3, Params: 121
Received message - SensorID: 5, MsgID: 3, Params: 150
Received message - SensorID: 2, MsgID: 3, Params: 80
Received message - SensorID: 4, MsgID: 3, Params: 25
Received message - SensorID: 2, MsgID: 3, Params: 86
Received message - SensorID: 3, MsgID: 3, Params: 129
Received message - SensorID: 2, MsgID: 3, Params: 92
Received message - SensorID: 2, MsgID: 3, Params: 76
Received message - SensorID: 3, MsgID: 3, Params: 130
Received message - SensorID: 4, MsgID: 3, Params: 26
Received message - SensorID: 5, MsgID: 3, Params: 128
Hall Effect Sensor Data: 128 - Status: Normal Magnetic Field - Structure Intact
Processing Sensor Data

```

System's Serial Logs 2

At this point, The user has a chance to change the thresholds for the messages here or they can opt to receive alerts based on the predefined default thresholds.

The system then takes the logs with the received message states and adds them to a data structure associated with that sensor which then updates the generated graphs every second based on the received data in real time.



System's Real Time Generated Graphs

Under Normal Operation, the graphs would display data as normal with the “Normal Operation” message displayed under the Graph’s title. But if one of the thresholds is passed. The graph’s background would turn red indicating and alerting the user of a potential danger, risk or error. The “Normal Operation” message is then changed to an appropriate error message that is specific to that sensor and the error eg. “Danger - High Pressure Detected” etc.

This pipeline ensures robust data handling from acquisition through processing to visualization, with built-in error checking and alert generation capabilities.

4. Testing and Validation

4.1. Test Methodology

Our testing approach focused on validating the core functionalities of our system using the STM32F411 microcontroller with FreeRTOS implementation. Since physical sensors were not available, we developed a comprehensive testing framework that utilized simulated sensor data that are relatively close to what we would expect to see in a real life underwater environment to verify system performance and reliability.

These function responsible for generating these values are shown here:

```
void RunAcousticSensor(TimerHandle_t xTimer) // Default 1000 ms
{
    char str[60];
    sprintf(str, "Acoustic sensor callback executing\r\n");
    print_str(str);

    const uint16_t variance = 20; // Variance for simulated acoustic data
    const uint16_t mean = 75; // Mean for simulated acoustic data

    // Simulate acoustic sensor data
    uint16_t simulatedAcousticData = (rand() % variance) + mean;

    // Send the simulated data
    send_sensorData_message(Acoustic, simulatedAcousticData);
    send_plot_data(Acoustic, simulatedAcousticData, xTaskGetTickCount());
}

void RunFlowRateSensor(TimerHandle_t xTimer) // Default 1000 ms
{
    const uint16_t variance = 30; // Variance for simulated flow rate
    const uint16_t mean = 200; // Mean for simulated flow rate

    // Simulate flow rate sensor data
    uint16_t simulatedFlowRate = (rand() % variance) + mean;

    // Send the simulated data
    send_sensorData_message(Acoustic, simulatedFlowRate);
    send_plot_data(Acoustic, simulatedFlowRate, xTaskGetTickCount());
}

void RunPressureSensor(TimerHandle_t xTimer) // Default 1000 ms
{
    const uint16_t variance = 50; // variance for simulated pressure data
    const uint16_t mean = 100; // mean for simulated pressure data

    // Simulate pressure sensor data
    uint16_t simulatedPressure = (rand() % variance) + mean;

    // Send the simulated data
    send_sensorData_message(Pressure, simulatedPressure);
    send_plot_data(Pressure, simulatedPressure, xTaskGetTickCount());
}

void RunTemperatureSensor(TimerHandle_t xTimer) // Default 1000 ms
{
    const uint16_t variance = 5; // Variance for simulated temperature data
    const uint16_t mean = 25; // Mean for simulated temperature data (DS18B20 typical range)

    // Simulate temperature sensor data
    uint16_t simulatedTemperature = (rand() % variance) + mean;

    // Send the simulated data
    send_sensorData_message(Temperature, simulatedTemperature);
    send_plot_data(Temperature, simulatedTemperature, xTaskGetTickCount());
}
```

System's Functions that simulates Real Time Data based on Real Conditions and Scenarios

Test Environment:

- Two STM32F411 Nucleo-64 development boards
- FreeRTOS real-time operating system
- Serial communication through UART
- Development environment: STM32CubeIDE
- Testing equipment: PC with serial monitoring capabilities

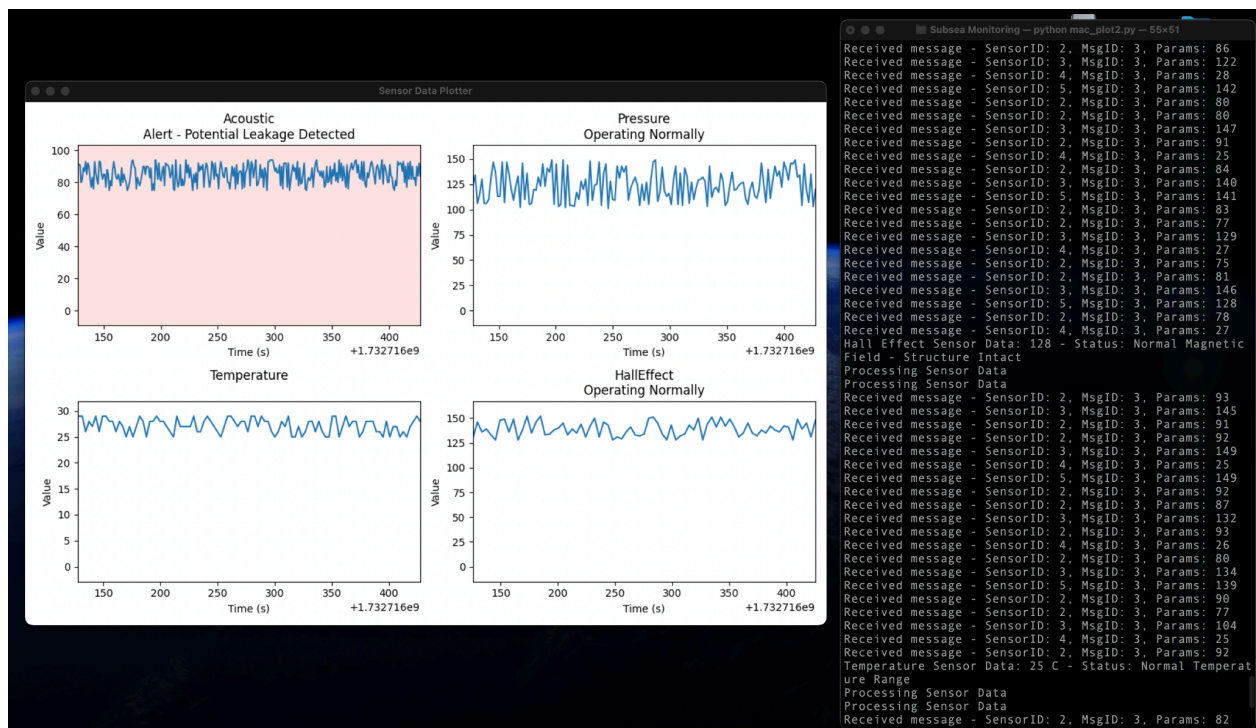
Test Categories: The testing process was divided into several key areas:

Task Scheduling Verification We implemented and verified multiple FreeRTOS tasks to ensure proper execution and verifying that our pipeline is adhering to the requirements and our standards and goals. Here is what we aimed to verify in our testing phase:

- The Deadlock prevention mechanisms put in place both by the us (the developers) and freeRTOS
- Data acquisition tasks running at 200ms intervals

- Processing task operating within the 500ms cycle requirement
- Communication task managing UART data transmission
 - Testing UART communication by verification of data packet transmission at 115200 baud rate
 - Confirmation of proper data formatting and packet structure
 - Validation of bidirectional communication between boards and Host PC (START/RESET messages and logs)
- Alert generation task responding to threshold violations communicated in the terminal logs
- The Graphs Generated in real-time as well as the user interface, thresholds defined by the user an overall alerting and notification system that would be used by our targeted audience in a real life scenario.
 - Verification of working accurate graphs
 - Testing different user set parameters
 - Verification of Alert messages under normal operation and the case where the thresholds are exceeded.

4.2. Results and Analysis



System's Real Time UI Prior to Final Update (No option to set thresholds in real time after execution)

Task Timing and System Reliability Results:

- Data acquisition task consistently met the 200ms sampling requirement

- Processing cycles completed within the specified 500ms window
- Bi-Directional Communication tasks maintained stable throughput at 115200 bps between the controller and the Host PC as well as between the two microcontroller boards.
- Updated Graphs, Alert generated as expected every second.
- User Defined Thresholds did result in alerts being generated at different thresholds as expected

These test results validate our system's core functionalities and demonstrate its ability to meet the specified performance requirements. While testing was conducted using simulated sensor data and not in a real-life environment so we were **unable to test** for prolonged use and operation as well as meeting the power requirements for our specified time period. The results did however provide strong evidence that the system architecture and implementation are suitable for real-world deployment with actual sensors. We did also manage to find numerous areas for potential improvements after testing, especially when it comes to the data handling after acquisition which we would get into in section 6 of this report.

5. Cost Analysis and Economic Viability

5.1. Component and System cost

Our system demonstrates remarkable cost-effectiveness, with a total implementation cost of approximately \$242. This cost breakdown includes all essential components required for a fully functional monitoring system assuming only one sensor platform is intended for deployment.

The table below highlights our costs for our system. Core Processing Components:

Component	Estimated Cost per unit(\$)	Quantity	Source
STM32F411 Nucleo-64 Development Board	20.00	2	https://www.mouser.ca/ProductDetail/STMicroelectronics/NUCLEO-F411RE?qs=Zi3UNFD9mQidE-Jg18RwZ2g%3D%3D&mgh=1&utm_id=17633666059&qad_source=1&gbraid=0AAAAADn_wf2qVIMJIPtzoellrPSLA-M-KV&qclid=Cj0KCQIAgJa6BhCOARisAMIL7V9GoYGCXgn6gI-J6WoDUrmJEPP6DcDU5BDMRFTME9ztbNv0nHL-SfgawaApmHEALw_wcB
MS5837-30BA Pressure Sensor	20.00	1	https://www.mouser.ca/ProductDetail/Measurement-Specialties/MS583730BA01-50?qs=SvmF4ZFxmIz%2F53mFym%252BZPQ%3D%3D&mgh=1&utm_id=17633666059&qad_source=1&gbraid=0AAAAADn_wf2qVIMJIPtzoellrPSLA-M-KV&qclid=Cj0KCQIAgJa6BhCOARisAMIL7V_umHu74A3CdglPyKfRgGTLH8Ar--DToAFYn9ZRO0NlunyssYn-7ElaAnoIEALw_wcB
SEN-HZ21WA Flow Rate Sensor	14.00	1	https://www.mouser.ca/ProductDetail/Seeed-Studio/314150005?qs=SEIPoaY2y5IMuXe9y0k8Qw%3D%3D&mgh=1&utm_id=17633666059&qad_source=1&gbraid=0AAAAADn_wf2qVIMJIPtzoellrPSLA-M-KV&qclid=Cj0KCQIAgJa6BhCOARisAMIL7V9ovAG01tCd5FMG26nKQqyo-yi90xvLeUmkhoFduXwdBmTd19LJAaAsRNEALw_wcB
HTI-96-MIN Hydrophone Acoustic Sensor	100.00	1	N/A
DS18B20 Temperature Sensor	11.00	1	https://www.mouser.ca/ProductDetail/Analog-Devices-Maxim-Integrated/DS18B20+PAR?qs=0Y9aZN%252BMVCWPX1JasrmQg%3D%3D&mgh=1&utm_id=17633666059&qad_source=1&gbraid=0AAAAADn_wf2qVIMJIPtzoellrPSLA-M-KV&qclid=Cj0KCQIAgJa6BhCOARisAMIL7V8YLQwh8yO57q0iZqUsXnINEEcXG7Emv10JMp3ORJZfLsin-rsubQaAp8DEALw_wcB
Power Bank (6000mAh)	07.00	1	https://www.canadacomputers.com/en/power-banks/130514/orico-6000mah-power-bank-for-smart-phone-white-orico-firefly-m6-wh-pro.html
Waterproof Enclosure	30.00	1	N/A
Miscellaneous (Cables, Connectors, etc.)	20.00	1	N/A
Total Estimated Cost	242.00		

Table highlighting system prototype cost

The system's cost structure demonstrates careful consideration of component selection for our specific use case given what our team had in hand to start with. We tried to balance performance requirements with economic constraints. Each component was chosen not only for its technical specifications but also for its

cost-effectiveness in meeting system requirements like water resistance, power requirements, compatibility, ease of use etc.

5.2. Comparison with existing solutions

Traditional underwater monitoring systems typically cost between \$5,000 and \$15,000, depending on their complexity and capabilities. System we are referring to when it comes to our comparison that we have come across while research are:

- Aquasens Marine Environmental Monitoring System by RBR which would require the use of their software:

<https://rbr-global.com/products/>

- Suite **Cost: Approximately** \$12,000 (considering you have to buy elements of their system (sensors, controller, software separately)
- Features similar to our system:
 - Pressure sensor for depth monitoring
 - Water flow measurement
 - Temperature sensing
 - Real-time data collection
 - Scalable with the chance to add more of their sensors or multiple systems
- Key differences:
 - Uses proprietary sensors rather than commercial ones which means you NEED their sensors
 - Requires specialized deployment equipment
 - High power consumption requiring custom battery packs (More money gone)
 - Complex maintenance procedures
 - Monthly subscription for data services (their software)

- HOBO Water Level Data Logger by Onset:

<https://www.onsetcomp.com/products/data-loggers/u20l-0x?srsltid=AfmBOorcab2Skc5DnC7iEWd0t61MzdRDOPzhGpLA5Q7TPnfA2OBblRXO>

- **Cost:** Around **\$375** for basic setup (software “HOBOWare Pro” not included in price)
- Similar functionalities:
 - Underwater pressure monitoring
 - Temperature sensing
 - Data logging capabilities
 - Real-time alerts
- Key differences:
 - Limited to single-point measurements

- **Requires** dedicated software (HOBOWare Pro) license
- No acoustic monitoring capabilities
- Less scalable (cannot network multiple units easily like our system)

Our solution provides several advantages when put against the alternatives including:

Extremely Low-Cost, Open-Source, Scalable and Easy-to-use when it comes to the alternatives and would be the go-to solution for someone starting out in the industry, beginners or hobbyists. This is due to the fact that our system:

- Utilizes of readily available, commercial-grade sensors
- Is Implemented on standard free-to-use development platforms
- Is Simple yet effective when it comes to the use case, power management and scalability
- Streamlined communication protocols requiring minimal additional hardware for scalability, upgrades or integration.

6. Future Improvements and Scalability

6.1. Proposed Enhancements -

6.1.1. Additional sensor integration possibilities

As our current implementation is very open ended and would serve as a great base for a similar solution with a larger ecosystem. Since we only use 4 sensors taking up around 5 pins on our microcontroller, this leaves an endless array of possibilities when it comes to additional sensor integration.

6.1.2. Wireless connectivity options

Adding an optional wireless connection option to allow for communication between the two microcontrollers would be a great add-on to our system. It would make deployment and sometimes even retrieval of the underwater sensor platform along with the sensor cluster a lot more forgiving, it also comes with the added benefit of potentially increased water resistance since there are less wires that would have to come in contact with the water.

As good as this feature can sound and be, integrating it is extremely difficult as detrimental changes on how the boards communicate along with how the data is parsed would have to be put in place.

This application also depends highly on the intended use case and environment, radio, infrared, or other light signals can be used but all of these options come with their potential down sides and cost.

SIM cards or satellite connectivity can be introduced but this is either extremely expensive or unreliable when it comes to off-shore applications.

6.1.3. User interface improvements

Incorporate a better user friendly front end dashboard with additional pages for one-time access things like settings and configurations. Additionally enabling acquired data and configuration saving would be nice considering who our target audience is.

6.1.4. Fault and anomaly detection and handling

Add multiple sensors even if they already exist as they would help the user or system figure out which ones might be giving potentially wrong values and are potentially faulty.

Add algorithms to detect faulty sensors/errors by checking with historical data at this specific location, date, time, environment etc. This would help find weaknesses in our sensors and potentially wrong data, and eliminate them.

6.1.5. Advanced data analysis capabilities

When we say this we refer to many aspects that can make a data-analysts or the end users life better. This

involves:

Implementing a way to save/export the data acquired in the session to .csv, .txt, or JSON This would aid in later using the data for further analysis or implementation of new models or algorithms etc.

Adding predictive algorithms (ML) to analyze and predict changes in data in real time based on current or previous data entry. This can later involve integrating this with the alerts feature to see if not only the current data is exceeding thresholds but what if the current data indicate that the upcoming or subsequent data have a possibility of bypassing or exceeding thresholds

6.2. Scalability Considerations -

Our Real-Time Subsea Turbulence and Sediment Monitoring System has been designed with significant scalability potential, particularly leveraging the STM32F411's capability to support up to 11 sensor platforms simultaneously. This scalability framework encompasses several key dimensions that enable the system to grow and adapt to larger monitoring requirements.

Additionally, adding more platforms is not the only benefit here, we can simply integrate new sensors to the current platforms if the remaining pins comply with the sensors communication protocols and needs.

Adding a different power bank is our “plug-and-play” solution to solving the different end user’s complex and varying power needs depending on their use case, operation time and convenience. If the user want to operate the unit for a longer time, all they have to do is unplug the current power bank and add another one with the battery capacity they need knowing our total current requirements for operation of 50mA. So if they would want it to operate for X hours then their needs is $50 \times X =$ minimum required battery capacity in mAH.

Our Software’s codebase is also open-source and available on github.com/abdullasadoun giving the user the change to view/change or update the code to detect errors, improve performance, integrate sensors, add new alerting features or other software features in general etc.

These scalability features make our system highly adaptable to various deployment scenarios while maintaining its core functionality and performance characteristics. The ability to expand from a single unit to a network of up to 11 sensor platforms provides significant flexibility in addressing different environmental monitoring needs while maintaining system reliability and real-time performance capabilities.

7. Conclusions

7.1. Summary of achievements and outcome

Our Real-Time Subsea Turbulence and Sediment Monitoring System successfully achieved several significant milestones in creating an efficient, cost-effective, open source, and scalable solution for underwater environmental monitoring we have successfully designed, implemented and tested our system which achieved all of our goals and gave us the chance to view potential areas for future development and improvement.

- Developed a complete system architecture integrating multiple sensors for comprehensive environmental monitoring
- Implemented real-time data and analysis
- Created a reliable alert and data visualization system for environmental threshold violations that are end-user defined.
- Identified an easy to follow recipe to evaluate the system's power needs.
- Achieved significant cost reduction compared to traditional monitoring systems and competitors, with a total system cost of \$242
- Successfully implemented FreeRTOS for efficient task management and real-time processing
- Demonstrated reliable communication between system components

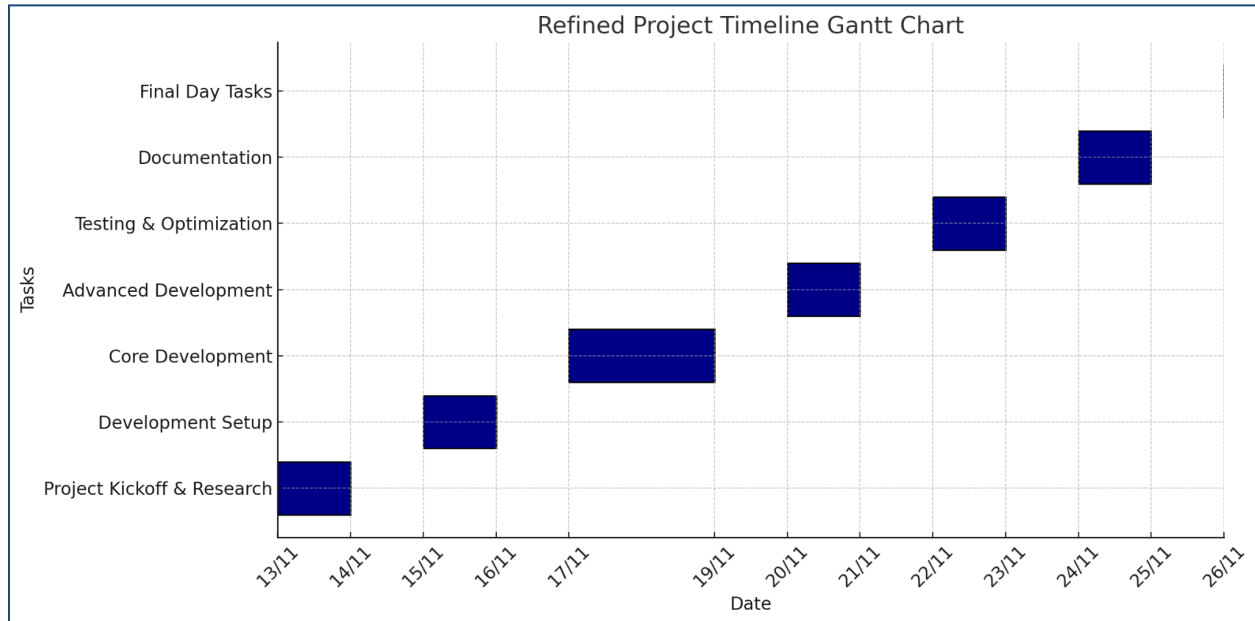
7.2. Lessons learned

The development process provided valuable insights into real-time embedded system design enabling our team members to gain a lot of knowledge, lessons and skills when it comes to Real-Time systems, some of our key lesson learned are listed below

- Importance of thorough system requirement analysis before implementation
- The importance of careful task scheduling in real-time systems
- Design of system architecture given the requirements and use-case
- Value of modular system design in system development and the benefits of scalable architecture in system design
- Significance of robust error handling in environmental monitoring systems
- Better understanding of our microcontroller and freeRTOS along with the development environment (STMCube32 IDE)
- Better understanding of Real-Time Operating System Concepts and techniques like scheduling, priority queues, deadlock prevention, using semaphores and mutex etc.

Supporting Information and Appendices

A. Project Timeline and Milestones



System's Timeline and milestones gantt chart

- Key Phases shown in the gantt chart:
 - Project planning phase
 - Design phase
 - Implementation phase
 - Testing phase
 - Documentation phase
 - Key milestone achievements
 - Include your project timeline diagram from the presentation

B. Technical Component Datasheets and Tools Specifications

- MS5837-30BA Pressure Sensor Datasheet:** TE Connectivity. (n.d.). *MS5837-30BA Miniature Pressure Sensor*. Retrieved from [TE Connectivity Datasheet](#).
- SEN-HZ21WA Flow Rate Sensor Datasheet:** e-Gizmo. (n.d.). *SEN-HZ21WA Flow Meter Specifications*. Retrieved from [e-Gizmo Datasheet](#).
- HTI-96-MIN Hydrophone Datasheet:** High Tech, Inc. (n.d.). *HTI-96-MIN Hydrophone Specifications*. Retrieved from [High Tech Inc. Datasheet](#).

- d. **DS18B20 Temperature Sensor Datasheet:** Maxim Integrated. (n.d.). *DS18B20 Programmable Resolution 1-Wire Digital Thermometer*. Retrieved from [Digi-Key Datasheet](#).
- e. **STM32F411 Nucleo-64 User Manual:** STMicroelectronics. (n.d.). *UM1724: STM32 Nucleo-64 Boards (MB1136) User Manual*. Retrieved from [STMicroelectronics Manual](#).
- f. Requirement for our **Python** Script
 - Python 3.12.0
 - contourpy==1.3.1
 - cycler==0.12.1
 - fonttools==4.55.0
 - kiwisolver==1.4.7
 - matplotlib==3.9.2
 - numpy==2.1.3
 - packaging==24.2
 - pillow==11.0.0
 - pyparsing==3.2.0
 - pyserial==3.5
 - python-dateutil==2.9.0.post0
 - six==1.16.0
 - tk==0.1.0
- g. UART Communication reference: <https://cs140e.sergio.bz/notes/lec4/uart-basics.pdf>
- h. STM32Cube Documentation:
<https://www.st.com/en/ecosystems/stm32cube/documentation.html>
- i. Pyserial Library Docs: [pyserial python documentation](https://pyserial.github.io/)
- j. Tkinter Library Docs: <https://docs.python.org/3/library/tkinter.html>
- k. Whimsical to create Architecture and flow diagrams:
<https://whimsical.com/underwater-sensor-platform-communication-flow>

C. Other Resources

IEEE Guide to Software Requirements Specifications – Referenced in Table of Contents, Section 1.4 (References)