ECED 4402 Mini Project

Marine environment detector

Written by

Han Zhang (B00852157)

Koushalram Kumaran (B00842320)

Dalhousie University     
Halifax, Nova Scotia     
Nov 23rd, 2023

**Ocean Temperature and Acidification Monitoring Network**

**Project Background and Objectives**

Global climate change is causing an increase in ocean temperatures and ocean acidification, profoundly affecting marine ecosystems. This project aims to develop an Ocean Temperature and Acidification Monitoring Network, utilizing a series of sensors deployed at different depths to real-time monitor and record changes in temperature, oxygen content and acidity across the global oceans. By collecting this data, the project will support climate change research, ecological conservation plans, and sustainable fisheries management.

**Ocean Environment Sensors**

****

1. Temperature Sensor:

The submersible temperature sensor measures sea water temperature at a specified time and location. It provides real-time temperature data with a precision of 1°C. The dynamic range covers the typical temperature range of sea water (0°C to 35°C), making it suitable for various marine applications.

* + Units: Degrees Celsius (°C)
  + Dynamic Range: 0°C to 40°C
  + Precision: 1°C
  + Sample Rate: 1Hz

1. PH Sensor:

The seawater pH sensor is designed for ocean acidification monitoring. Operating at a 1 Hz sample rate, it provides accurate and precise pH values ranging from 5.0 to 9.0, with a precision of 0.1 pH units. This sensor is crucial for understanding changes in ocean acidity and alkalinity.

* + Units: pH
  + Dynamic Range: 5.0 to 9.0 pH
  + Precision: 0.1 pH units
  + Sample Rate: 1 Hz

1. Dissolved Oxygen Sensor:

The optical dissolved oxygen sensor measures the concentration of dissolved oxygen in water. it provides accurate data in the range of 0 to 9 mg/L, with a precision of 0.1 mg/L. This sensor is essential for assessing water quality and understanding the oxygen levels in the marine environment.

* + Units: Milligrams per Liter (mg/L)
  + Dynamic Range: 0 to 9 mg/L
  + Precision: 0.1 mg/L
  + Sample Rate: 1 Hz

1. Ocean Depth Sensor:

The pressure-based depth sensor measures the depth of the ocean at a specific location. Operating at a 0.1 Hz sample rate, it provides precise depth readings in the range of 0 to 500 meters, with a high precision of ±0.5 meters. This sensor offers valuable insights into the vertical dimensions of the marine environment, aiding in underwater topography studies and navigation.

* + Sample Rate: 0.1 Hz
  + Units: Meters
  + Dynamic Range: 0 to 500 meters
  + Precision: ±0.5 meters

**Communication Protocols**

**Table 1: Controller and Platform**

|  |  |  |  |
| --- | --- | --- | --- |
| **Device** | **Protocol** | **Direction** | **Parameters** |
| ResetSensor | $CNTRL,00„\*, CS\n | TX | - |
| SensorResetAck | $CNTRL,01„\*, CS\n | RX | - |
|  |  |  |  |
| Enable PH Sensor | $PHVAL,00,PERIOD,\*,CS\n | TX | PERIOD (8 characters) is the period for the update of the (PH sensor) data request |
| Enable Depth Sensor | $DEPTH,00,PERIOD,\*,CS\n | TX | --(Depth Sensor) -- |
| Enable Temperature Sensor | $TEMPE,00,PERIOD,\*,CS\n | TX | --(Temperature Sensor) -- |
| Enable Oxygen Sensor | $OXYGE,00,PERIOD,\*,CS\n | TX | --(Oxygen Sensor) -- |
|  |  |  |  |
| PH Sensor Enabled Ack | $PHVAL,01„\*,CS\n | RX | - |
| Depth Sensor Enabled Ack | $DEPTH,01„\*,CS\n | RX | - |
| Temperature Sensor Enabled Ack | $ TEMPE,01„\*,CS\n | RX | - |
| Oxygen Sensor Enabled Ack | $OXYGE,01„\*,CS\n | RX | - |
|  |  |  |  |
| PH Data | $PHVAL,03,DATA, \*, CS\n | RX | D (8 characters) is the (PH data) identifier re-quested |
| Depth Data | $DEPTH,03,DATA, \*, CS\n | RX | --(Depth Data) -- |
| Temperature Data | $TEMPE,03,DATA, \*, CS\n | RX | --(Temperature Data) - |
| Oxygen Data | $OXYGE,03,DATA, \*, CS\n | RX | -- (Oxygen Data) -- |

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

**Sensor Emulation:**

1. Temperature Sensor:

A white screen with black text

Description automatically generated

The temperature of seawater is generated by a random function. Because the temperature range of seawater in nature varies between 2 and 30 degrees, the temperature generated by our simulator will be simulated between 0 and 35 degrees.

1. PH Sensor:

A white background with black text

Description automatically generated

The values of the pH simulator are generated by a random function. Considering that the measurement target is the ocean surface, the range of randomly generated numbers is 75 to 85 (7.5 to 8.5. The surface pH value of seawater is usually between pH 7.5 and pH 8.5.

1. Dissolved Oxygen Sensor:

**A white background with black text and numbers

Description automatically generated**

The value of the oxygen content simulator is generated by a random function. Considering that the measurement target is the ocean surface, the range of randomly generated numbers is 40 to 90 (4.0 to 9.0). The oxygen content of seawater is usually between pH 4.0 to 9.0 mg/L.

1. Ocean Depth Sensor:

The depth sensor simulator is provided by ECED 4402 Lab 4. Considering that the measurement target is the ocean surface, the depth conversion range is 0 to 250m. Starting from 0, it increases by 10m each time. After reaching 250m, it decreases by 10m each time.

A screenshot of a computer program

Description automatically generated

**Acquire the Data from the Three Sensors:**

Code is attached in the appendix.

A screen shot of a computer

Description automatically generated

This is the data displayed by the screen.

**Task 4 And Compression:**

Code of Task 4 is attached in the appendix.

According to the normal transmission method, numbers will be converted into characters before transmission, which means that transmitting a three-digit number 120 will occupy 3 bytes. Now the numbers (0 to 250) are all transmitted in their 8-bit binary format. This means that transmitting the value 120 only takes 1 byte. And a unique binary code will be used to indicate the sensor type. The corresponding meanings of the 8-bit binary code are as follows:

Compression Password Table:

|  |  |  |  |
| --- | --- | --- | --- |
| Meaning | Value(uint8\_8) | Size | symbol |
| Depth | 252 | 1 byte | Extended ASCII |
| Oxygen | 253 | 1 byte | Extended ASCII |
| Temperature | 254 | 1 byte | Extended ASCII |
| PH | 255 | 1 byte | Extended ASCII |
| NANA | 251 | 1 byte | Extended ASCII |
| Sensor Data | 0-250 | 1 byte | (0-127) ASCII (>127) Extended ASCII |

Here is the compressed data:

A black background with white text

Description automatically generated

And the data received by the host PC will be decompressed and display on host PC monitor.

A screen shot of a computer

Description automatically generated

**Conclusion:**

The Ocean Temperature and Acidification Monitoring Network project addresses some critical challenges posed by global climate change that significantly impact marine ecosystems, specifically the rise in ocean temperatures and acidification. By deploying a set of ocean environment sensors monitoring pH levels, water temperature, dissolved oxygen concentrations, and ocean depth at different depths, the project aims to provide real-time information monitoring and recordings across global oceans. These sensors offer precise and high-resolution real-time data essential for climate change research, ecological conservation planning, and sustainable fisheries management. The communication protocols integrated into the system ensure sensor control and data retrieval, improving its effectiveness in contributing to our understanding of the marine environment and making informed decisions for the betterment of ocean ecosystems.

**Code:**

**Data Processing:**

**void** **parse\_sensor\_message**(**struct** CommMessage\* currentRxMessage)

{

**static** **enum** ParseMessageState\_t currentState = *Waiting\_S*;

uint8\_t CurrentChar;

**static** uint16\_t sensorIdIdx = 0, MessageIdIdx = 0, ParamIdx = 0, checksumIdx = 0;

**static** **char** sensorId[6],CSStr[3];

**static** uint8\_t checksum\_val;

**static** **const** **struct** CommMessage EmptyMessage = {0};

**while**(xQueueReceive(Queue\_extern\_UART, &CurrentChar, portMAX\_DELAY) == pdPASS && currentRxMessage->IsMessageReady == false) // as long as there are characters in the queue.

{

**if** (CurrentChar == '$'){ //Reset State Machine

checksum\_val = CurrentChar;

sensorIdIdx = 0;

MessageIdIdx = 0;

ParamIdx = 0;

checksumIdx = 0;

currentState = *SensorID\_S*;

\*currentRxMessage = EmptyMessage;

**continue**;

}

// TO DO: we must calculate the received checksum!

**switch** (currentState)

{

**case** *Waiting\_S*: // Do nothing

**break**;

**case** *SensorID\_S*: //Get Sensor ID Code

checksum\_val ^= CurrentChar;

**if**(CurrentChar == ','){

currentState = *MessageID\_S*;

**break**;

}

**else** **if** (sensorIdIdx < 5){

sensorId[sensorIdIdx++] = CurrentChar;

}

**if**(sensorIdIdx == 5){

//Add NULL Terminator

sensorId[sensorIdIdx] = '\0';

**if**(**strcmp**(sensorId, "CNTRL") == 0)//Sensor ID: Controller

currentRxMessage->SensorID = *Controller*;

**else** **if**(**strcmp**(sensorId, "PHVAL") == 0)//Sensor ID: Acoustic

currentRxMessage->SensorID = *PH*;

**else** **if**(**strcmp**(sensorId, "DEPTH") == 0)//Sensor ID: Depth

currentRxMessage->SensorID = *Depth*;

**else** **if**(**strcmp**(sensorId, "TEMPE") == 0)

currentRxMessage->SensorID = *Temperature*;

**else** **if**(**strcmp**(sensorId, "OXYGE") == 0)

currentRxMessage->SensorID = *Oxygen*;

**else**{//Sensor ID: None

currentRxMessage->SensorID = *None*;

currentState = *Waiting\_S*;

}

}

**break**;

**case** *MessageID\_S*: //Get Message Type

checksum\_val ^= CurrentChar;

**if**(CurrentChar == ','){

currentState = *ParamsID\_S*;

}

**else**{

**if**(MessageIdIdx < 2){

currentRxMessage->messageId = currentRxMessage->messageId \* 10;

currentRxMessage->messageId += CurrentChar - '0';

}

MessageIdIdx++;

}

**break**;

**case** *ParamsID\_S*: //Get Message Parameter (Period/Data)

checksum\_val ^= CurrentChar;

**if**(CurrentChar == ','){

currentState = *Star\_S*;

}

**else** **if**(ParamIdx < 8){

currentRxMessage->params = currentRxMessage->params \* 10;

currentRxMessage->params += CurrentChar - '0';

}

**break**;

**case** *Star\_S*:

checksum\_val ^= CurrentChar;

**if**(CurrentChar == ','){

currentState = *CS\_S*;

}

**break**;

**case** *CS\_S*:

**if**(checksumIdx < 2){

CSStr[checksumIdx++] = CurrentChar;

}

**if**(checksumIdx == 2){

currentState = *Waiting\_S*;

CSStr[checksumIdx] = '\0';

currentRxMessage->checksum = **strtol**(CSStr, NULL, 16);

**if**(currentRxMessage->checksum == checksum\_val){

currentRxMessage->IsMessageReady = true;

currentRxMessage->IsCheckSumValid = true;

}**else**{

currentRxMessage->IsCheckSumValid = false;

}

}

**break**;

}

}

}

**Controller:**

/\*

\* SensorController.c

\*

\* Created on: Oct 24, 2022

\*

\*/

**#include** <stdio.h>

**#include** <stdbool.h>

**#include** "main.h"

**#include** "User/L2/Comm\_Datalink.h"

**#include** "User/L3/PHSensor.h"

**#include** "User/L3/DepthSensor.h"

**#include** "User/L3/TemperatureSensor.h"

**#include** "User/L3/OxygenSensor.h"

**#include** "User/L4/SensorPlatform.h"

**#include** "User/L4/SensorController.h"

**#include** "User/util.h"

// Required FreeRTOS header files

**#include** "FreeRTOS.h"

**#include** "Timers.h"

**#include** "semphr.h"

QueueHandle\_t Queue\_Sensor\_Data;

QueueHandle\_t Queue\_HostPC\_Data;

bool IsRemotePlatformEnabled = false;

**struct** SensorStates sensorState = {0};

**static** **void** **ResetMessageStruct**(**struct** CommMessage\* currentRxMessage) {

**static** **const** **struct** CommMessage EmptyMessage = {0};

\*currentRxMessage = EmptyMessage;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This task is created from the main.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **SensorControllerTask**(**void** \*params) {

**enum** HostPCCommands HostPCCommand = *PC\_Command\_NONE*;

**struct** CommMessage currentRxMessage = {0};

//bool IsRemotePlatformEnabled = false;

**do** {

// Check if there is any received Sensor Data

**if** (xQueueReceive(Queue\_Sensor\_Data, &currentRxMessage, 0) == pdPASS) {

**switch** (currentRxMessage.SensorID) {

**case** *Controller*:

**switch** (currentRxMessage.messageId) {

**case** 0: // Do Nothing

**break**;

**case** 1: // Received Acknowledgment

sensorState.IsPHAck = false;

sensorState.IsDepthAck = false;

sensorState.IsTemperatureAck = false;

sensorState.IsOxygenAck = false;

**break**;

**case** 3: // Do Nothing

**break**;

}

**break**;

**case** *PH*:

**switch** (currentRxMessage.messageId) {

**case** 0: // Do Nothing

**break**;

**case** 1: // Received Acknowledgment

sensorState.IsPHAck = true;

**break**;

**case** 3: // Received PH Data

sensorState.PHData = currentRxMessage.params;

**break**;

}

**break**;

**case** *Depth*:

**switch** (currentRxMessage.messageId) {

**case** 0: // Do Nothing

**break**;

**case** 1: // Received Acknowledgment

sensorState.IsDepthAck = true;

**break**;

**case** 3: // Received Depth Data

sensorState.DepthData = currentRxMessage.params;

**break**;

}

**break**;

**case** *Temperature*:

**switch** (currentRxMessage.messageId) {

**case** 0: // Do Nothing

**break**;

**case** 1: // Received Acknowledgment

sensorState.IsTemperatureAck = true;

**break**;

**case** 3: // Received Temperature Data

sensorState.TemperatureData = currentRxMessage.params;

**break**;

}

**break**;

**case** *Oxygen*:

**switch** (currentRxMessage.messageId) {

**case** 0: // Do Nothing

**break**;

**case** 1: // Received Acknowledgment

sensorState.IsOxygenAck = true;

**break**;

**case** 3: // Received Oxygen Data

sensorState.OxygenData = currentRxMessage.params;

**break**;

}

**break**;

**default**: // Should not get here

ResetMessageStruct(&currentRxMessage);

}

}

// Check if there is any received Host PC Command Data

**if** (xQueueReceive(Queue\_HostPC\_Data, &HostPCCommand, 0) == pdPASS) {

**switch** (HostPCCommand) {

**case** *PC\_Command\_START*:

//print\_str("Sending Start Command\r\n");

send\_sensorEnable\_message(*PH*, 5000);

send\_sensorEnable\_message(*Depth*, 5000);

send\_sensorEnable\_message(*Temperature*, 5000);

send\_sensorEnable\_message(*Oxygen*, 5000);

IsRemotePlatformEnabled = true;

**break**;

**case** *PC\_Command\_RESET*:

//print\_str("Sending Reset Command\r\n");

send\_sensorReset\_message();

IsRemotePlatformEnabled = false;

**break**;

**default**:

**break**;

}

}

vTaskDelay(1000 / portTICK\_RATE\_MS);

} **while** (1);

}

/\*

\* This task reads the queue of characters from the Sensor Platform when available

\* It then sends the processed data to the Sensor Controller Task

\*/

**void** **SensorPlatform\_RX\_Task**() {

**struct** CommMessage currentRxMessage = {0};

Queue\_Sensor\_Data = xQueueCreate(80, **sizeof**(**struct** CommMessage));

request\_sensor\_read(); // requests a usart read (through the callback)

**while** (1) {

parse\_sensor\_message(&currentRxMessage);

**if** (currentRxMessage.IsMessageReady == true && currentRxMessage.IsCheckSumValid == true) {

xQueueSendToBack(Queue\_Sensor\_Data, &currentRxMessage, 0);

ResetMessageStruct(&currentRxMessage);

}

}

}

/\*

\* This task reads the queue of characters from the Host PC when available

\* It then sends the processed data to the Sensor Controller Task

\*/

**void** **HostPC\_RX\_Task**() {

**enum** HostPCCommands HostPCCommand = *PC\_Command\_NONE*;

Queue\_HostPC\_Data = xQueueCreate(80, **sizeof**(**enum** HostPCCommands));

request\_hostPC\_read();

**while** (1) {

HostPCCommand = parse\_hostPC\_message();

**if** (HostPCCommand != *PC\_Command\_NONE*) {

xQueueSendToBack(Queue\_HostPC\_Data, &HostPCCommand, 0);

}

}

}

**void** **Compress\_Task**()

{

**char** Msg[4];

**while** (1)

{

**if** (IsRemotePlatformEnabled == true) {

**if** (sensorState.IsDepthAck == true) {

**sprintf**(Msg, "%c%c",252,sensorState.DepthData);

print\_str(Msg);

} **else** {

**sprintf**(Msg, "%c%c",252,251);

print\_str(Msg);

}

**if** (sensorState.IsOxygenAck == true)

{

**sprintf**(Msg, "%c%c",253, sensorState.OxygenData);

print\_str(Msg);

} **else** {

**sprintf**(Msg, "%c%c",253, 251);

print\_str(Msg);

}

**if** (sensorState.IsTemperatureAck == true)

{

**sprintf**(Msg, "%c%c",254, sensorState.TemperatureData);

print\_str(Msg);

} **else**{

**sprintf**(Msg, "%c%c",254, 251);

print\_str(Msg);

}

**if** (sensorState.IsPHAck == true) {

**sprintf**(Msg, "%c%c",255, sensorState.PHData);

print\_str(Msg);

} **else** {

**sprintf**(Msg, "%c%c",255, 251);

print\_str(Msg);

}

vTaskDelay(1000 / portTICK\_RATE\_MS);

}

}

}

**Task 4:**

// This task is responsible for compressing and sending sensor data.

**void** **Compress\_Task**()

{

**char** Msg[4]; // Buffer to hold the message to be sent.

**while** (1) // Infinite loop to continuously check and send data.

{

// Check if the remote platform is enabled for data transmission.

**if** (IsRemotePlatformEnabled == true) {

// Check if there is an acknowledgment for depth data.

**if** (sensorState.IsDepthAck == true) {

// Prepare the message with depth sensor ID and data.

**sprintf**(Msg, "%c%c",252,sensorState.DepthData);

print\_str(Msg); // Send the message.

} **else** {

// Prepare the message with depth sensor ID and no data indication.

**sprintf**(Msg, "%c%c",252,251);

print\_str(Msg); // Send the message.

}

// Check if there is an acknowledgment for oxygen data.

**if** (sensorState.IsOxygenAck == true)

{

// Prepare the message with oxygen sensor ID and data.

**sprintf**(Msg, "%c%c",253, sensorState.OxygenData);

print\_str(Msg); // Send the message.

} **else** {

// Prepare the message with oxygen sensor ID and no data indication.

**sprintf**(Msg, "%c%c",253, 251);

print\_str(Msg); // Send the message.

}

// Check if there is an acknowledgment for temperature data.

**if** (sensorState.IsTemperatureAck == true)

{

// Prepare the message with temperature sensor ID and data.

**sprintf**(Msg, "%c%c",254, sensorState.TemperatureData);

print\_str(Msg); // Send the message.

} **else**{

// Prepare the message with temperature sensor ID and no data indication.

**sprintf**(Msg, "%c%c",254, 251);

print\_str(Msg); // Send the message.

}

// Check if there is an acknowledgment for pH data.

**if** (sensorState.IsPHAck == true) {

// Prepare the message with pH sensor ID and data.

**sprintf**(Msg, "%c%c",255, sensorState.PHData);

print\_str(Msg); // Send the message.

} **else** {

// Prepare the message with pH sensor ID and no data indication.

**sprintf**(Msg, "%c%c",255, 251);

print\_str(Msg); // Send the message.

}

// Delay the task for a second to reduce CPU usage.

vTaskDelay(1000 / portTICK\_RATE\_MS);

}

}

}