H2O ROBOTICS

Underwater localization & communication

Triangulation by acoustic signal

Alyssa AGNISSAN

August 12, 2022

Contents

1 Project overview

The objective of this project is the development of an underwater localization and communication system.

This system takes the form of one or more groups of three buoys (GoB), placed on the surface of the water, and one or more acoustic units (AU), immersed under the water. A central unit (CU), located on the shore or onboard a boat controls it.

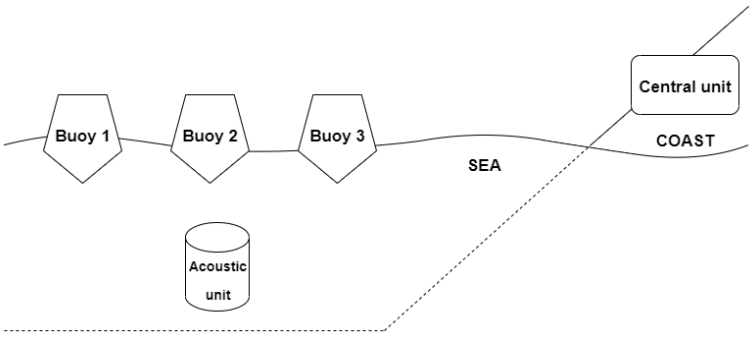

Figure 1: Overview of the localization and communication system

2 Hardware architecture

This section details the hardware of each element of the system.

2.1 Central unit

The CU consists of a mobile phone connected, via Bluetooth or USB serial, to an ESP32 micro-controller equipped with LoRa and powered by a battery.

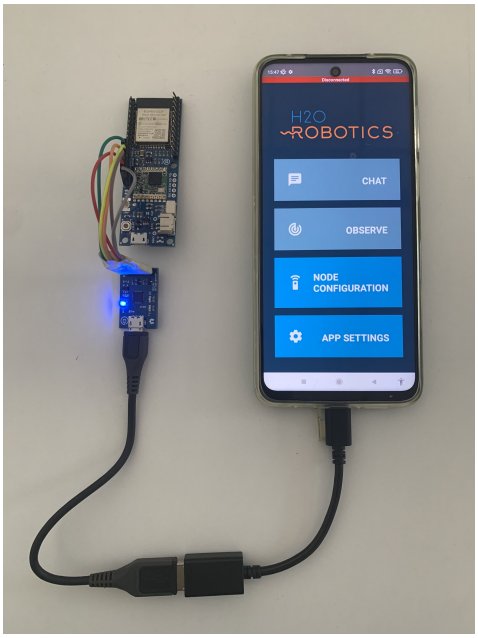

Figure 2: Central unit - phone with H2O application *(right)*, USB cable for serial connection *(bottom)*, ESP32 board with LoRa *(top left)* and battery *(not represented)*

ESP32 is a series of system-on-a-chip (SoC) micro-controllers from Espressif Systems, based on the Xtensa LX6 architecture from Tensilica. It integrates dual-mode support for Wi-Fi and Bluetooth (up to LE 5.0 and 5.11), and a Digital Signal Processor (DSP), which is a microprocessor optimised to run digital signal processing applications as fast as possible.

LoRa (from "long range") is a physical proprietary radio communication technique. It is based on spread spectrum modulation techniques derived from chirp spread spectrum (CSS) technology. LoRaWAN is its software communication protocol and system architecture. Together, LoRa and LoRaWAN define a Low Power Wide Area (LPWA) networking protocol designed to wirelessly connect battery operated devices to the internet in regional, national or global networks, and targets key Internet of things (IoT) requirements such as bi-directional communication, end-to-end security, mobility and localization services. It can provide communication up to five km in urban areas, and up to 15 km or more in rural areas. The low power, low bit rate, and IoT use distinguish this type of network from a wireless WAN that is designed to connect users or

businesses, and carry more data, using more power.

This protocol only allows the communication of small amounts of data: their rate ranges from 0.3 kbit/s to 50 kbit/s per channel.

LoRa has two different topologies: point-to-point or network, as shown on the figure below.

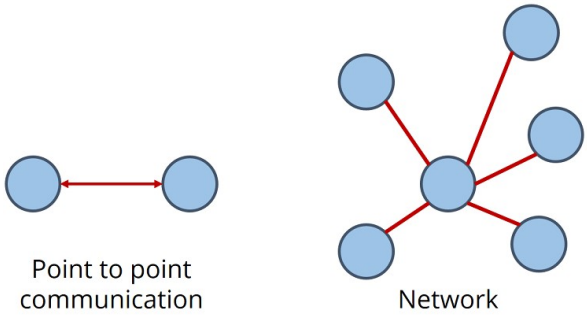

Figure 3: LoRa topologies

In point-to-point communication, two LoRa enabled devices talk with each other using radio frequency signals.

On another hand, a LoRaWAN-based network is made up of end devices, gateways, a network server, and application servers. End devices send data to gateways, and the gateways pass it on to the network server, which, in turn, passes it on to the application server as necessary. Additionally, the network server can send messages (either for network management, or on behalf of the application server) through the gateways to the end devices. Figure 4 illustrates the LoRaWAN network process. The key difference between the LoRaWAN approach and others is that end devices are paired with the network itself and are not exclusively tied to a single gateway. Rather, end devices broadcast their signals to all gateways within range. Each of the receiving gateways pass the data packet along to the network server, and then the network server de-duplicates the message and sends a single version of it to the application server.

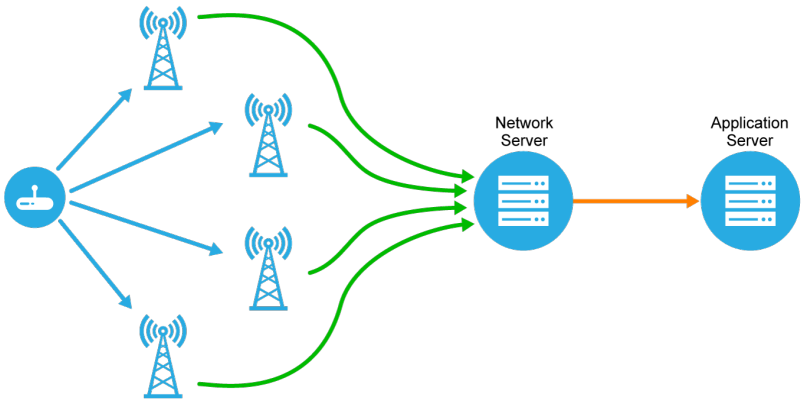

Figure 4: LoRa network process

Additionally, a mobile application, called *PingerApp* and developed by H2O Robotics to select the AU(s) to be located, is necessary for the proper functioning of the CU. We will get into the details of this application a little later.

2.2 Buoys

As mentioned earlier, the system includes three buoys for position triangulation. Two of these buoys are "slave" buoys while the last one is the "master": these two roles will be discussed in the next section.

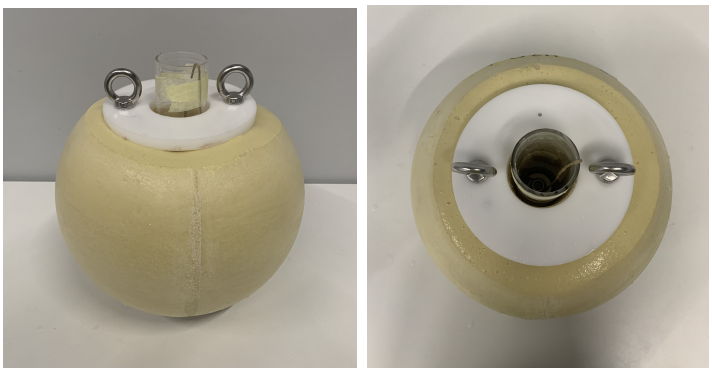

Figure 5: Buoy prototype

Each buoy has a device within it that we will call a "witness". It is this witness that allows the sending and receiving of acoustic data. It is arranged as follows:

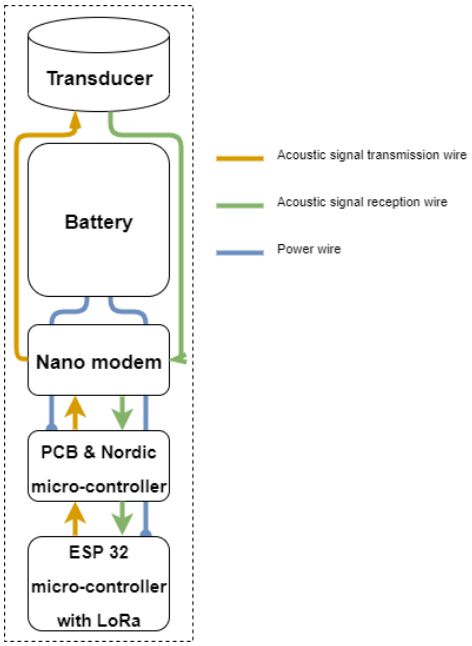

Figure 6: Scheme of the witness placed inside the buoy

The witness, as the CU, has an ESP32 micro-controller provided with LoRa, meant to communicate with this latter. The board is connected to a Printed Circuit Board (PCB) with Nordic micro-controller, powered by a battery: its role is to emit the request for sending an acoustic signal. This request is then converted into a modulated

signal by a nanomodem. Finally, the data, now analogue, is transmitted to a transducer. This latter ensures the transformation of the analogue signal into an acoustic signal.

The buoy, in addition to emitting acoustic signals, can also receive them. In this case, the chain of command described above is simply run in the other direction.

2.3 Acoustic unit

Figure 7: Acoustic unit

The AU, intended to be immersed in water, also consists of a PCB, a nanomodem and a transducer. It works exactly the same way as the PCB-nanomodem-transducer unit of the buoys. Each AU has a unique three-digit identifier.

3 Software architecture

The architecture of the system is as follows:

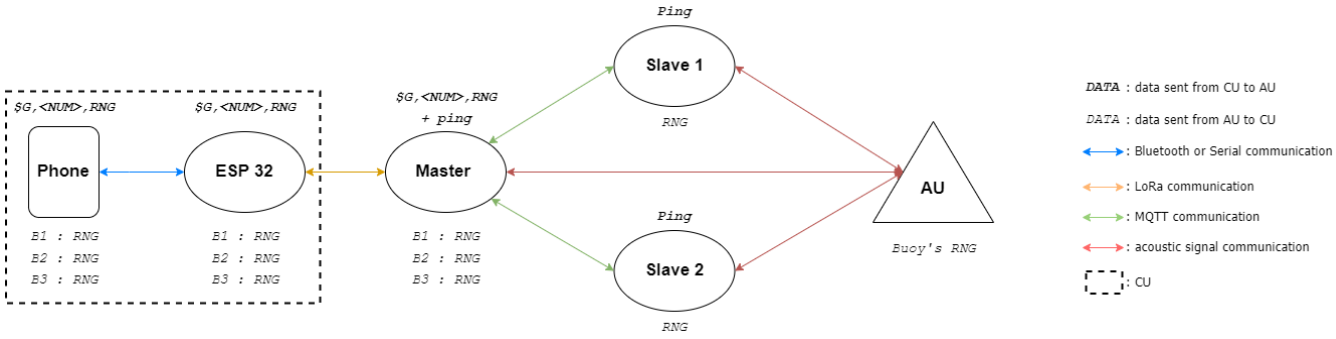

Figure 8: Overview of the project's software architecture

This architecture can then be divided into two distinct parts. The first is the above water communication between the CU, the master buoy and the slave buoys. The second is the underwater communication between the three buoys and the AU. The AU software part is not described in this document.

3.1 Mobile application

Before studying the software architecture of the project, let's take a look at the *PingerApp* application which, as mentioned earlier, is intended to indicate to the buoys which AUs to ping. The software part is not described.

It has four main features: *Node configuration*, *Chat*, *Observe* and *App settings*. The *Node configuration* section allows the user to scan the mobile phone's surroundings for GoBs or AUs. Once the devices are detected, the user can select one and access information about it (battery voltage, depth, pressure and temperature of the environment). In the *Chat* section, messages can be sent from the selected device to a specific AU. The *Observe* part is intended to ???. Finally, the *App settings* allows the configuration of some settings of the application.

The application is still under development.

3.2 Central unit - master buoy communication

Let's now study the above water communication architecture, starting by the communication between the CU and the master buoy.

From the mobile application, the user selects the AU(s) to be located. After validation, a message of the form *"$G,* <NUM>*, RNG"*, where NUM corresponds to the three digits identifier of the unit, is transmitted via Bluetooth or serial port to the ESP32 coast board. This message is called a ping request. The board then sends it to the master buoy via LoRa. This step is repeated every 10 seconds until the user send a different ping request.

As the CU can theoretically be located several kilometers from the buoys, it is essential to use a wireless communication protocol capable of covering long distances such as LoRa. In addition, the unit must be easily transportable: the low energy consumption represented by the LoRa protocol means that the CU can be supplied with a power supply that does not make it difficult to transport.

Furthermore, when data is transmitted via LoRa, it is broadcast to all devices equipped with this communication protocol. The establishment of a control point is therefore necessary to avoid undesirable communication between two devices other than the CU and the designated master buoy. To do this, a unique identifier is assigned to each of this two buoys. When sending the ping request, the CU then communicates, in its message, its identifier as well as the identifier of the recipient. If the device receiving the message does not have the identifier indicated in the message, it is ignored. The same process is performed from the master buoy to the CU.

3.3 Master - slave buoys communication

As explained above, the 3 buoys in the sea are divided into a master buoy and two slave buoys. The master buoy acts as a transfer node through which all the information between the CU and the sea is passed. Once the master buoy has received the ping request from the CU, it then transmits it to its two slave buoys so that all three can ping the AU.

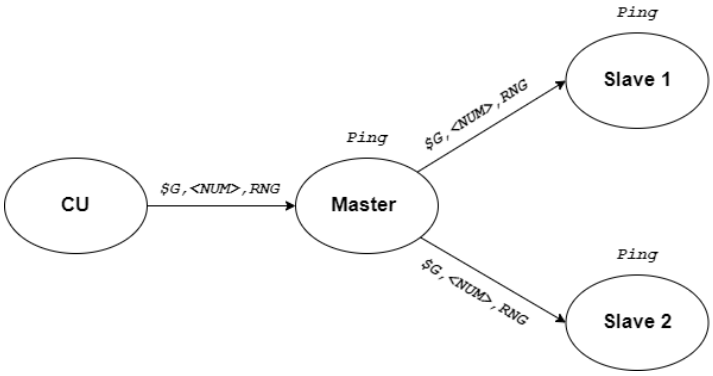

Figure 9: Master - slave buoys communication and roles

In such a configuration, the use of the LoRa protocol is no longer appropriate. Indeed, here, the reception by the master buoy of data from multiple senders - the slave buoys - in a simultaneous manner is necessary. However, given the type of equipment used in this project, only point-to-point communication is possible with LoRa. To meet this need, another wireless protocol is used: Message Queuing Telemetry Transport (MQTT).

MQTT is a lightweight, publish-subscribe, machine to machine network protocol. It is designed for connections with remote locations that have devices with resource constraints or limited network bandwidth. It must run over a transport protocol that provides ordered, lossless, bi-directional connections—typically, TCP/IP. The MQTT protocol defines two types of network entities: a message broker and a number of clients. An MQTT broker is a piece of software running on a computer, that can be self-built or hosted by a third party, and that receives all messages from the clients and then routes the messages to the appropriate destination clients. An MQTT client is any device - from a micro-controller up to a fully-fledged server - that runs an MQTT library and connects to an MQTT broker over a network. Information is organized in a hierarchy of topics. When a publisher has a new item of data to distribute, it sends a control

message with the data to the connected broker. The broker then distributes the information to any clients that have subscribed to that topic. The publisher does not need to have any data on the number or locations of subscribers, and subscribers, in turn, do not have to be configured with any data about the publishers.

If a broker receives a message on a topic for which there are no current subscribers, the broker discards the message unless the publisher of the message designated the message as a retained message. A retained message is a normal MQTT message with the retained flag set to true. The broker stores the last retained message and the corresponding Quality of Service (QoS) for the selected topic. Each client that subscribes to a topic pattern that matches the topic of the retained message receives the retained message immediately after they subscribe. The broker stores only one retained message per topic. This allows new subscribers to a topic to receive the most current value rather than waiting for the next update from a publisher.

In this project, the architecture of the MQTT communication between the master and the slave buoys is quite simple:

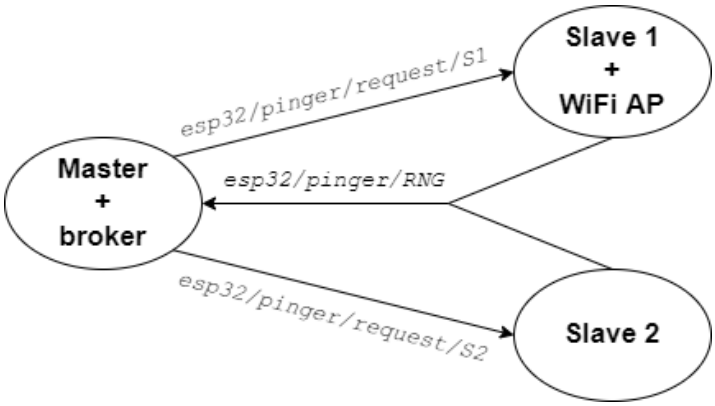

Figure 10: MQTT architecture between the three buoys

With the ESP32 having 2 cores, the master buoy runs the broker program on core 1, in parallel with its main program running on core 0. On the other hand, as explained above, it is imperative that the three client buoys are connected to a network for communication to be established. To do so, it is therefore necessary to have a WiFi access point (AP). This AP - of WiFi standard 802.11 b/g/n - is created by the ESP32 of one of the slave buoys. However, this WiFi connection requires that the three buoys are not more than 150 meters apart from each other.
Regarding the topics, the communication from the master to the slaves is done via two distinct topics: *esp32/pinger/request/S1* and *esp32/pinger/request/S2*. Communication in the opposite direction is done on the *esp32/pinger/RNG* topic.

Once the ping request has been transmitted to each of the three buoys, they send a signal to the selected AU. As simultaneous pinging of a single unit by several devices is

not possible, a delay of 100 ms is set between the sending from each buoy. This delay is kept as short as possible so that the AU has time to process each of the signals on one side, and that the buoys remain synchronized on the other. Indeed, as the AU is theoretically mobile, if the buoys were to stop pinging it synchronously, the distances calculated would no longer match. This would in fine distort the determination of the position of the unit.

After the signal is sent and if it reaches the AU, the buoys will receive two responses. The first one is of the form *"$R,TOP,ACK,002,006,* <AU_NUM>*"*, where AU_NUM corresponds to the identification number of the AU. This message acknowledges that the AU to be pinged has received the request correctly. However, if it is of the form *"$R,TOP,ACK,**000**,006,* <AU_NUM>*"*, that means that a problem occurred during the reception of the ping request. The AU will not be pinged. The second message, of the form *"$R,* <NUM>*,* <DIST>*"*, corresponds to the distance (*DIST*) between a buoy and the unit. It is sent if the acoustic signal emitted by the AU, in response to the ping request, was able to reach the buoy. Otherwise, the response is *"$R,TOP,LOG,Ping Timeout"*.

The slave buoys then transmit their response via MQTT to the master, which stores them in a list. Finally, this list is sent to the CU via LoRa.

3.4 Data back-up

The data retrieved by the CU are saved on a web server. To access it, a ThingsBoard dashboard has been created, which makes it easier to visualise the data. ThingsBoard is an open-source server-side platform that allows to monitor and control IoT devices. The software architecture of this data back-up is as follows:

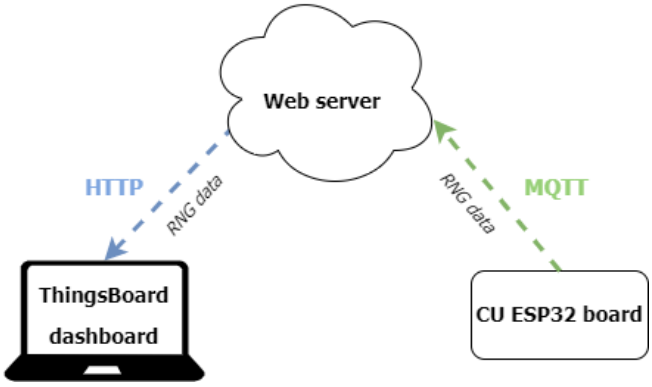

Figure 11: Data back-up software architecture

Data is sent from the CU to the web server via MQTT: to do so, the CU's ESP32 board must be able to connect to a local network or the Internet. The dashboard then collects the data from the server via HTTP protocol. To access the dashboard, from a computer, simply go to the following address: https://demo.thingsboard.io/dashboard/5bd46900-069e-11ed-8857-89a1708eda91?publicId=e95a4020-0e72-11ed-9c79-ad222c995ab7. Be careful ! A trial was used to create this dashboard, so it may not be accessible anymore the moment you read this.

The dashboard is very basic: it only contains a time-series table, displaying the data received by the CU in the last 24 hours. Also, in order to be able to differentiate the data of different AUs, if the system is communicating with more than one, their ping request is displayed above the line where their RNG data is located in the table.

4 How to use the system ?

4.1 Procedure

This section explains the different steps to respect to use the system properly. The procedure is common to all the modes of use described in the next section.

Step 1: First, the IP address of the broker in the slave buoys' codes has to be changed so the MQTT communication can be established. To do so, the value of the *mqtt_ broker* variable has to be changed with the correct IP address. Since the broker code is run by the master buoy, its IP address corresponds to the master's IP address. Thus, to retrieve the master's IP address, after uploading the master buoy's code on an ESP32 board, open the Serial Monitor on Arduino IDE and look at the line where it is written *"IP address"*. Then, copy this address and paste it to change the *mqtt_ broker* variable value.

Step 2: Secondly, the WiFi AP on which the CU's ESP32 board will be connected has to be configured. As said before, this WiFi connection is intended to send RNG data from the CU to the Thingsboard dashboard. This step is currently carried out using a mobile application providing a serial USB terminal. From this terminal, after you have turned on the board, enter the name of the desired WiFi AP when the *"Enter WiFi name :"* instruction is displayed. Then enter the password when the *"Enter WiFi password :"* instruction is displayed. A message confirming the connection to the AP and to the ThingsBoard server will appear when it is established. Be careful ! To establish the connection with the server, make sure to activate the WiFi **after** you have enter the name and password in the terminal. Finally, if you want to change the AP the board is connected to, you will have to reboot the board by pressing its *RESET* button.

Step 3: The third step is to turn on the three buoys' ESP32 boards. Please note that the buoys' boards must be switched on in a specific order to allow them to establish the MQTT communication: first the master buoy, then the slave buoy with the WiFi AP (number 1) and finally the other slave buoy (number 2). Indeed, with this startup order, the master first waits for a WiFi connection. Then, once slave 1 is turned on, the master's WiFi connection is established and it can then create its broker. With the broker available, slave 1 can connect to it. Similarly, slave 2, once turned on, can also connect to the WiFi AP and the broker, both of which are available. Once the MQTT communication is established by one of the buoys, this latter will send a confirmation message, displayed on the CU's phone terminal. This confirmation message is of the form *"<NAME> BUOY : MQTT connection established"*, where NAME corresponds to the role of the buoy. This way, if the phone does not receive a confirmation message for each of the 3 buoys, either wait for the buoy(s) to connect or turn the ESP32 boards off and on again in the order indicated above.

Step 4: The final step consists in informing the system of the AU(s) to be pinged. As the development of the *Pinger App* application is still in progress, this step is also carried out using the serial USB terminal for the moment. From this terminal, send the address(es) of the AU(s) to be pinged. If only one AU is to be pinged, the message

to be sent should be of the form *"$G,* <NUM>*, RNG"*, where NUM corresponds to the three digits identifier of the unit. If several AUs are to be pinged, then the message to be sent should be of the form *"$G,* <NUM1>*, RNG ; $G,* <NUM2>*, RNG"*. Be careful! Do not add any space before or after the semicolon, at the end of the message or between words, otherwise the ping of some AUs will not be taken into account. For each AU ping, if it went well, the CU will receive six different messages: one ACK message and one RNG message per buoy. This way, if one message is missing, it is more easy to understand where the problem is coming from.

4.2 Modes of use

The underwater system can offers multiple modes of localization and communication. Not all of them have been developed yet, but they represent interesting prospects for further development. This section details each of them.

4.2.1 1 CU, 1 GoB and 1 AU

This is the most basic of the four modes of use. It works in the way described in section 3, about the software architecture of the project (cf. figure 8).

4.2.2 1 CU, 1 GoB and multiple AUs

This second mode allows, from a single CU and a single GoB, to establish up to 10 parallel communications with AUs.

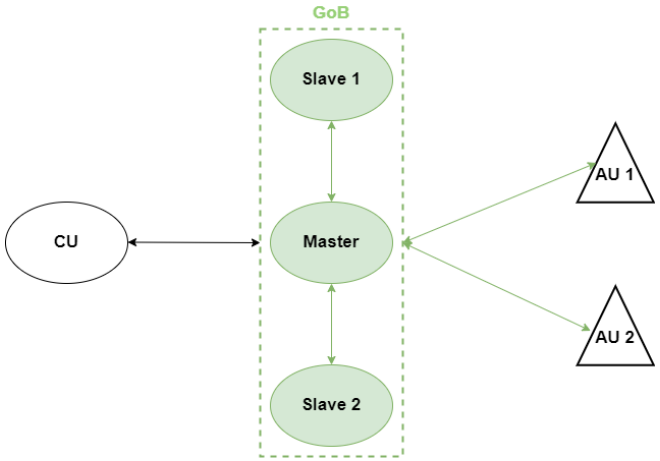

Figure 12: Architecture of *1 CU - 1 GoB - multiple AUs* communication

It is impossible for a buoy to ping several AUs simultaneously. To overcome this, the communication procedure is based on a regular alternation of the ping request to be sent. If the CU wants to communicate with N AUs, it will send a first ping request to address 001, wait 8 seconds to receive data if there is any and repeat the same steps with the remaining *N-1* addresses. Once the N AUs have been pinged, the CU will repeat the steps described indefinitely, until it receives a new instruction from the user.

4.2.3 1 CU, multiple GoB and 1 AU

This mode allows the user to communicate with several GoB almost simultaneously. It has not been developed yet.

In this configuration, the CU must be able to receive data from multiple master buoys. This multi-reception must be managed in such a way that it does not receive several information flows at the same time, otherwise data may be lost. However, regarding the hardware used in the project, only a point-to-point LoRa communication is allowed

between the CU and a GoB (cf. figure 3). Thus, in order to enable such a communication structure, the LoRa-equipped micro-controller of the CU has to be replaced by a getway, to create a LoRaWAN network.

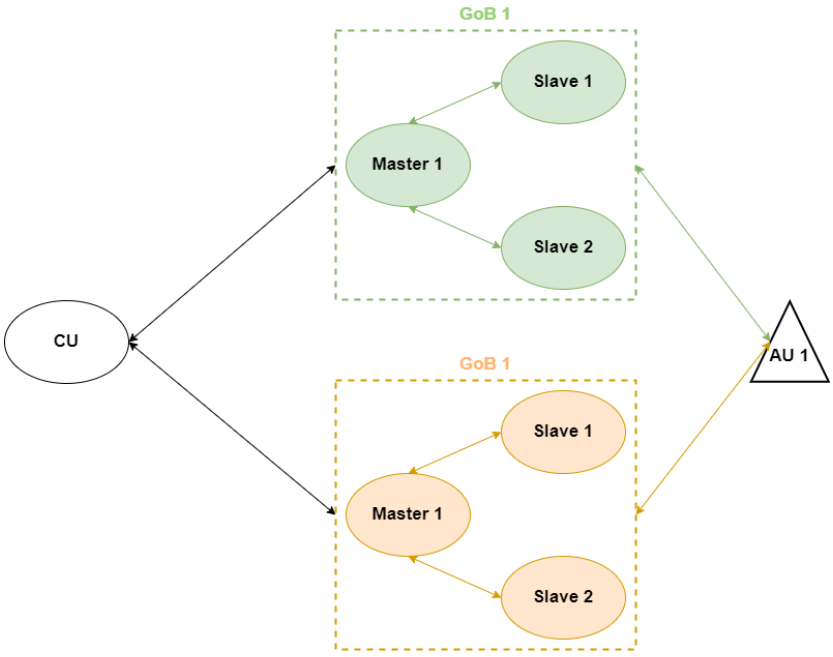

Figure 13: Architecture of *1 CU - multiple GoB - 1 AU* communication

4.2.4 1 CU, multiple GoB and multiple AUs

This last mode is a combination of modes 2 and 3. It has not been developed neither.

As the GoB can be positioned in different locations from each other, within a radius of 15 km from the CU, the available communication/localization area is considerably extended. The user will be able to set up a complex underwater communication network, based on a single central unit.

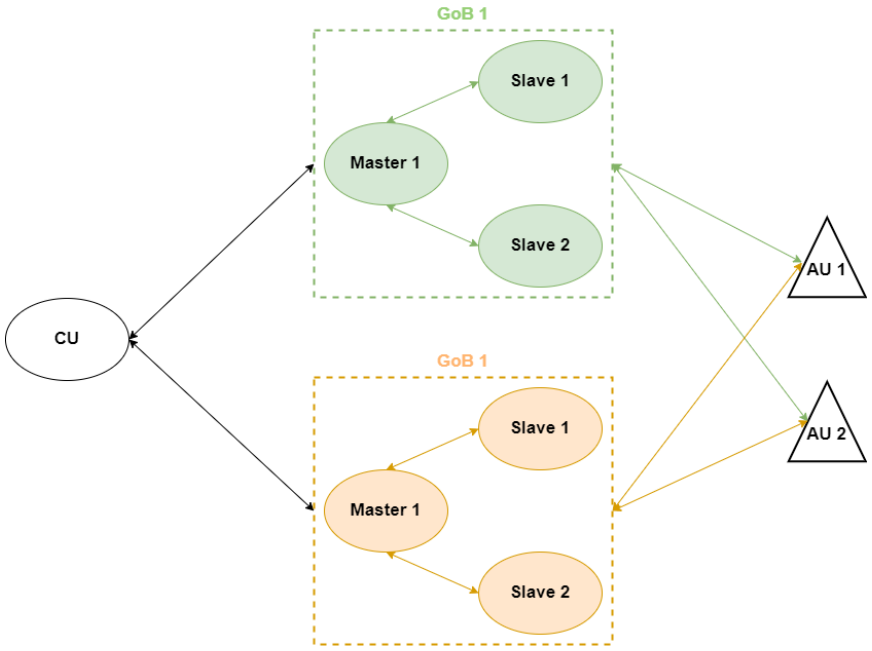

Figure 14: Architecture of *1 CU - multiple GoB - multiple AUs* communication

5 Potential areas of improvement

As the development of the system is not fully completed, some software problems remain. Here is an exhaustive listing of them:

- **Syntax of ping requests sent from the CU:** As explained in the subsection 4.1, sending the ping request *"$G, <NUM>, RNG"* from the CU is subject to syntax constraints. No spaces should be added between characters in the message, otherwise the ping command will not be taken into account completely.

- **Manual change of the IP address of the broker:** This point is also explained in the section 4.1. In order to establish the MQTT connection between the 3 buoys, they must know the IP address of the broker filtering the messages. As the broker is executed by the master buoy, its IP address corresponds to the IP address of this buoy: it does not matter on which board the code of the master buoy will be executed, the broker's address will be automatically recorded.

However, this is not the case for the 2 slave buoys. With the current implementation, they have no way of automatically retrieving the IP address of the broker: they must therefore be told manually, by changing the value of the variable corresponding to the IP address of the broker, in their respective code. Setting up a way to know and retrieve this address automatically would make the system more adaptive. No matter on which board the master and slave buoy codes would be executed, the MQTT connection would be established directly without any software modification.

- **Reboot of the master buoy:** It may happen that the ESP32 of the master buoy reboots alone and randomly. When the ESP32 is rebooted, the following message is displayed:

  task_wdt: Task watchdog got triggered. The following tasks did not reset the watchdog in time:
  task_wdt: - IDLE0 (CPU 0)
  task_wdt: Tasks currently running:
  task_wdt: CPU 0: Broker task
  task_wdt: CPU 1: Master Buoy task
  task_wdt: Aborting.
  abort() was called at PC 0x400e37d3 on core 0

  This problem is in fact due to the execution of 2 tasks in parallel: the one of the broker and the one of the main buoy program. ESP-IDF creates a watchdog timer for idle tasks in addition to the 2 other tasks stated above. It means that it creates two extra tasks IDLE0 and IDLE 1 (one for each core), whose sole purpose is to do nothing but feed the watchdog whenever its respective core is working. Whenever any task, in Arduino or in ESP-IDF, run with a higher priority than the IDLE tasks without delaying or blocking sufficiently long/often enough (such as when executing in a tight while loop), it triggers the watchdog, because that core is now fully busy. Thus, it is no longer idling often enough, meaning the idle "tasks" get starved, so they can't reset the watchdog. So basically, the task watchdog gets triggered when your code blocks the OS from switching tasks.

  This bug can cause data loss: by the time the connection of the buoy to the WiFi AP and the MQTT communication are re-established, the ping requests sent by the CU are no longer taken into account.

- **Unwanted sending of data to the CU :** When one of the buoys receives a ping request *"$G,* <NUM>*, RNG"* but its pinger does not respond, because it has no battery or is not connected to the ESP32 for instance, the buoy is supposed to not send anything to the CU. However, instead, it sends the ping request *"$G,* <NUM>*, RNG"* that it has received. On the other hand, it also happens that random characters are sent when the pinger does not respond.