

Compte rendu

Projet Embarquai2



BENAITION Loris, DORIATH Arthus

ICSSA – BTS SNIR 2021/2022

- **Attention, certaines parties du rapport ont été rédigé et réalisé par un seul étudiant. Par conséquent, le nom sur le sommaire y est précisé ou bien une sous partie « auteurs » y a été rajouté. Si aucun de ces cas de figures n'est présent, alors on peut considérer que ces parties sont communes à tous les étudiants.**
- **Etant donné que Baptiste Bossuet a quitté le BTS pendant le projet, certaines parties qui touchent de près ou de loin au cahier des charges de cet étudiant seront incomplètes voire inexistantes.**

Remerciements

Tout d'abord, nous tenons à remercier personnellement Monsieur Bodin Dominique, enseignant en section SN, de nous avoir confié ce projet et de nous avoir fait confiance.

Je tiens également à remercier l'équipe éducative et administrative de l'ICSSA pour nous avoir aidé et accompagné au long de ce projet notamment Mr SALLE, Mr ARNAULT et Mme CANDILLIER.

INTRODUCTION	9
REPARTITION DES TACHES	10
1. Etudiant 1 DORIATH Arthus	10
2. Etudiant 2 BENAITIER Loris	10
3. Etudiant 3 BOSSUET Baptiste.....	10
PLANNING PREVISIONNEL	11
SPRINT	12
BASE DE DONNÉES.....	15
ANALYSE UML	16
1. Diagramme des exigences	16
2. Diagramme de cas d'utilisation.....	17
3. Diagramme de classe.....	18
4. Diagramme de séquence	19
5. Diagramme de définition de bloc.....	20
MATERIELS UTILISES	21
1. Présentation du M5Stack Core2	23
LOGICIELS ET LIBRAIRIES UTILISES.....	26
1. Logiciels	26
2. Librairies	27
SYSTEME D'EXPLOITATION UTILISE	28
LANGAGES DE PROGRAMMATIONS UTILISES	29
RECUPERATIONS DONNEES CAPTEURS VIA LE M5STACK CORE2.....	30
1. Auteur.....	31
2. Objectif	31
3. Cahier des charges	31
4. Matériels nécessaires.....	31
4.1 la carte M5Stack Core 2	31
4.2 le module CAN Unit	32
4.3 Le matériel NMEA2000 SIMRAD	33
5. Présentation du standard NMEA2000.....	34
6. Développement du système	34
6.1 Environnement de développement	34
6.2 Librairie initiale.....	35
6.3 Présentation de la librairie NMEA2000	36

6.4 Présentation de la classe c_Navi	37
6.5 système de flag	39
6.6 Configuration	40
ENVOIE DES DONNEES DEPUIS LE M5STACK CORE2.....	41
1. Auteur.....	42
2. Objectif	42
3. Cahier des charges	42
4. Matériels nécessaires	42
4.1. Module UNIT LoRaWan868.....	43
4.2. Module Lorawan : Xbee Waveshare + Carte ATIM LoRaWan	44
5. Problème rencontré	45
6. Environnement de développement	45
7. Présentation de Lora et Lorawan	46
7.1. Introduction	46
7.2. Qu'est-ce que LoRa ?	46
7.3. Qu'est-ce que LoRaWAN ?	46
7.4. Quels sont les avantages des protocoles LoRa et LoRaWAN ?.....	47
8. Liaison physique : module Lorawan/M5Stack Core2	47
9. Choix du format de la trame contenant les données.....	49
9.1. Caractères au début et à la fin de la trame de données.....	49
9.2. Caractères de séparations entre chaque champ de la trame	49
9.3. FLAG	49
9.4. Données	51
9.5. CRC.....	52
10. Envoi de la trame contenant les données.....	57
11. Test unitaire pour l'envoi de la trame contenant les données	58
12. Test unitaire pour la génération de fausse valeur.....	59
13. Intégration avec l'étudiant 1.....	61
14. Conclusion.....	61
RECUPERATION DES DONNEES ENVOYEEES PAR LE M5STACK CORE2.....	62
1. Auteur.....	63
2. Objectif	63
3. Cahier des charges	63
4. Matériels et logiciels nécessaires.....	63
4.1. Routeur multitech MTCAP 868.....	64

4.2.	Raspberry pi	65
5.	Configuration routeur multitech Lorawan (MTCAP 868)	66
5.1.	Introduction	66
5.2.	Configuration des paramètres réseau du routeur.....	66
5.3.	Démarrage du NETWORK SERVEUR.....	67
5.4.	Ajouts des identifiants d'un module Lorawan (END-DEVICE)	67
5.5.	Ajout d'un appareil pour le lier avec un module Lorawan.....	69
5.6.	Transferts des données en MQTT.....	70
5.7.	Paramètre en fonction de notre localisation	71
6.	Configuration du serveur Mosquitto sur le PC administrateur.....	71
7.	Test du serveur Mosquitto	72
8.	Test serveur Mosquitto + routeur Lorawan + module Lorawan.....	73
9.	Conclusion.....	74
	ENREGISTREMENT DES DONNEES DANS LA BDD	75
1.	Auteur.....	76
2.	Objectif	76
3.	Cahier des charges	76
4.	Matériel et logiciels Nécessaires.....	76
5.	Décodage de la trame	76
5.1.	Récupérer le paquet qui contient la donnée	76
5.2.	Extraire la trame donnée du paquet Lorawan	77
5.3.	Extraire les différentes informations contenues dans la trame.....	77
5.4.	Traitement du CRC.....	77
5.5.	Traitement du flag	78
6.	Enregistrement dans la base de données.....	79
7.	Gestion performance du serveur	80
8.	Gestion erreurs du serveur	80
9.	Test unitaire	81
10.	Intégration avec l'étudiant 3.....	83
11.	Conclusion.....	83
	TRAITEMENT DES DONNEES VIA UNE IHM	84
1.	Auteur.....	85
2.	Situation.....	85
	IHM DU M5STACK CORE2.....	86
1.	Auteur.....	87

2. Objectif	87
3. Cahier des charges	87
4. Développement	87
4.1 librairie utilisée.....	87
4.2 Architecture de mon IHM	87
4.3 Différents états	88
4.4 Affichage d'une page.....	88
4.4 Cacher une page.....	89
4.5 Transition vers une autre page	89
4.6 Ajout d'une nouvelle page.....	90
5 images des différentes pages de l'IHM	90
DOSSIER MAINTENANCE ENVOI/RECEPTION/ENREGISTREMENT DONNEES..	93
1. Auteur.....	94
2. Objectif	94
3. Debug	94
INSTALLATION MODULES : ENVOI/RECEPTION/ ENREGISTREMENT DONNEE	96
1. Auteur.....	97
2. Objectif	97
3. Carte ATIM Lorawan + Xbee Waveshare	97
4. Routeur Multitech Lorawan.....	98
5. Serveur Mosquitto	105
6. Serveur python.....	106
UTILISATION DES MODULES : ENVOI/RECPETION/ENREGISTREMENT.....	108
1. Auteur.....	109
2. Objectif	109
3. Mise en œuvre	109
4. Debug	111
TEST FINAL DE L'ENSEMBLE DU PROJET	113
1. Test qui rassemble les parties de chaque étudiant.....	114
CONCLUSION	117
1. Synthèse et optimisations possibles	118
ANNEXES	119
1. Schéma	120
1.1. M5Stack Core2 + Carte ATIM LoRaWan + Xbee Waveshare	120
2. Bibliographie.....	120

3. Licences	120
-------------------	-----

INTRODUCTION

Dans le cadre de notre projet informatique de BTS SNIR, on nous a confié la réalisation du projet nommé « embarquai2 ».

Le but du projet consiste à développer un système de suivi de courses nautiques. Le public pourrait visualiser en temps réel l'état de la course, des bateaux sur le principe de l'interface graphique "Virtual Regatta".

La création et la suppression des participants et des courses seront accessibles par un administrateur via l'IHM. Toutes les courses ainsi que les participants seront enregistrées dans la base de données et rejouable par l'administrateur et visible par tout le monde.

Ce système ne concernerait que les courses proches des côtes de manière à pouvoir utiliser le système LoRaWan comme support de transmission des données.

Le système "mobile" embarqué sur le bateau doit récolter les informations issues des différents capteurs (vitesse, position, vent, direction du vent) et les envoyer à intervalles réguliers au système "fixe". Ce dernier stocke dans une base de données toutes les informations reçues et les affiche sur l'écran de télévision afin que le public puisse suivre la course en temps réel.

Afin de mener à bien ce projet, nous serons 3 étudiants à travailler dessus avec des tâches spécifiques pour chacun. Pour le versionning du projet, nous avons utilisé Github qui rassemble le travail de chaque étudiant. Le projet Github est accessible à l'aide du lien suivant : <https://github.com/Sorann753/embarquai>

Le but, à terme, est de rassembler le travail de toute l'équipe pour avoir un projet fonctionnel qui réponde aux attentes et aux exigences du cahier des charges. Les moyens qui seront alloués pour la mise en place de ce projet seront de 100€.

REPARTITION DES TACHES

1. Etudiant 1 DORIATH Arthus

- Installer et câbler la carte M5Stack ou RPI, ainsi que les modules SIMRAD
- Configurer le BusCAN et les modules SIMRAD
- Installer l'environnement de développement
- Développer le module CAN de récupération des données issues du bus CAN (NMEA 2000)
- Tester le module CAN de récupération des données issues du bus CAN
- Intégrer le module CAN au module LoRaWan de l'étudiant 2
- Documenter le module CAN

2. Etudiant 2 BENAITIER Loris

- Installer et câbler la carte M5Stack ou RPI
- Configurer la carte additionnelle LoRaWan et le routeur Multitech
- Installer l'environnement de développement
- Développer le module LoRaWan d'envoi/réception/enregistrement des données
- Tester le module LoRaWan
- Intégrer le module LoRaWan avec le module CAN de l'étudiant 1 et la base de données de l'étudiant 3
- Documenter le module LoRaWan

3. Etudiant 3 BOSSUET Baptiste

- Installer et câbler, le mini-ordinateur RPi et la télévision
- Configurer le mini-ordinateur RPi et la télévision
- Installer l'environnement de développement
- Développer le module WEB de configuration, le module IHM d'affichage et le module STAT de visualisation a posteriori
- Tester les modules WEB, IHM et STAT
- Intégrer les modules WEB, IHM, STAT entre eux ainsi qu'avec l'étudiant 2
- Documenter les modules

PLANNING PREVISIONNEL

	SPRINT 1	SPRINT 2	SPRINT 3	SPRINT 4	SPRINT 5
Arthus DORIATH	~ Installer l'environnement de développement « Visual studio code » ~ Commander matériel	~ Faire schéma UML ~ Câblé + configurer M5stack	Non défini	Non défini	Non défini
Loris BENAITIER	~ Rechercher de librairie ~ Installer l'environnement de développement « Visual studio code » ~ Commander matériel	Non défini	Non défini	Non défini	Non défini
Baptiste BOSSUET	~ Installer le Raspberry PI ~ Installer PhP ~ Installer SQLite ~ Installer Bootstrap ~ Commander matériel	Non défini	Non défini	Non défini	Non défini

SPRINT

Sprint 1	
Arthus DORIATH	<ul style="list-style-type: none">• Installer l'environnement de développement « Visual studio Community »• Etude du projet
Loris BENAITION	<ul style="list-style-type: none">• Recherche de librairie décodage Lorawan + M5Stack• Installer l'environnement de développement « Visual studio Community »• Etude du projet
Baptiste BOSSUET	<ul style="list-style-type: none">• Etude du projet

Sprint 2	
Arthus DORIATH	<ul style="list-style-type: none">• Tester matériel M5stack• Câblé Bus NMEA2000 et Capteur
Loris BENAITION	<ul style="list-style-type: none">• Programme de test / simuler fausses valeurs des capteurs• Faire programme génération fausse valeur
Baptiste BOSSUET	<ul style="list-style-type: none">• Installer PHP (v7) et Installer SQLite• Installer Bootstrap et Installer Raspberry PI

Sprint 3

Arthus DORIATH	<ul style="list-style-type: none">• Câbler + configurer M5Stack avec le NMEA2000 (CAN)• Recherche et test de librairie décodage CAN + M5Stack• Faire module NAVI• Tester module NAVI• Faire programme Principal MStack• Décodage CAN NMEA
Loris BENAULTIER	<ul style="list-style-type: none">• Faire programme pour prouver l'envoi• Faire un client de test pour le serveur• Programme de test / fausse base de données• Faire module Lorawan (envoie)• Câblé + configurer MStack & Lorawan• Faire module Lorawan Réception/ Enregistrement• Configurer le routeur Lorawan
Baptiste BOSSUET	<ul style="list-style-type: none">• Faire IHM configuration• Faire IHM viewer• Tester IHM

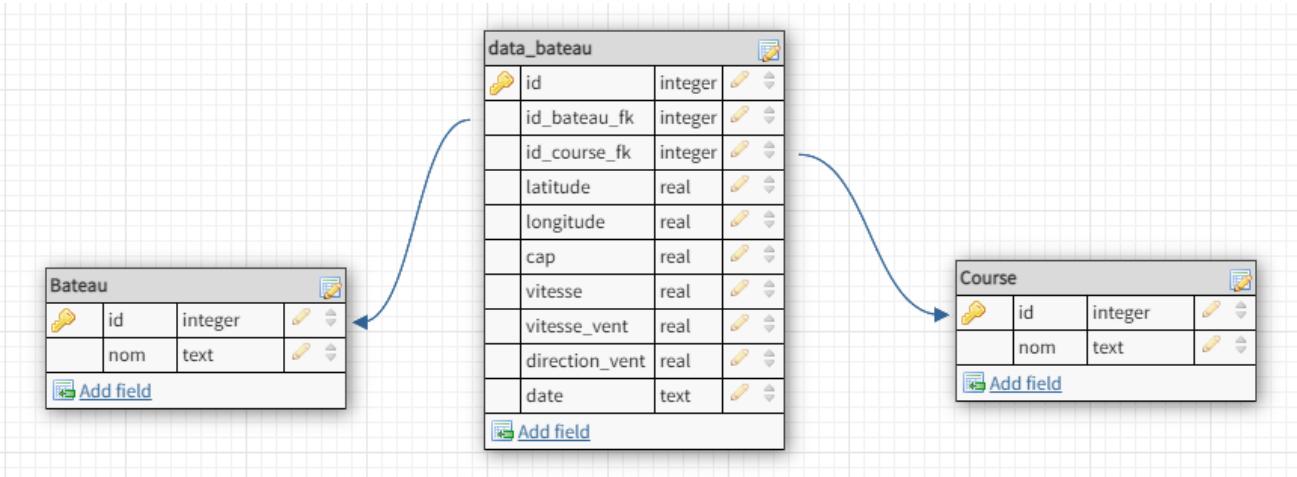
Sprint 4

Arthus DORIATH	<ul style="list-style-type: none">• Faire IHM M5Stack• Faire le boîtier + carte électronique pour l'intégration• Intégrer le module NAVI au Lorawan (software)
Loris BENAITIER	<ul style="list-style-type: none">• Intégrer le Lorawan Réception/Enregistrement avec BDD• Intégrer le module Lorawan, envoi avec le module NAVI• Faire script Bash pour l'installation des modules python nécessaires• Faire script Bash pour l'installation et l'utilisation de mosquitto• Test pour savoir si mes productions répondent au cahier des charges
Baptiste BOSSUET	<ul style="list-style-type: none">• Faire IHM STAT (à posteriori)• Intégrer IHM avec BDD

Sprint 5

Arthus DORIATH	<ul style="list-style-type: none">• Documentation(s) technique(s)
Loris BENAITIER	<ul style="list-style-type: none">• Documentation(s) technique(s)
Baptiste BOSSUET	<ul style="list-style-type: none">• Documentation(s) technique(s)

BASE DE DONNÉES



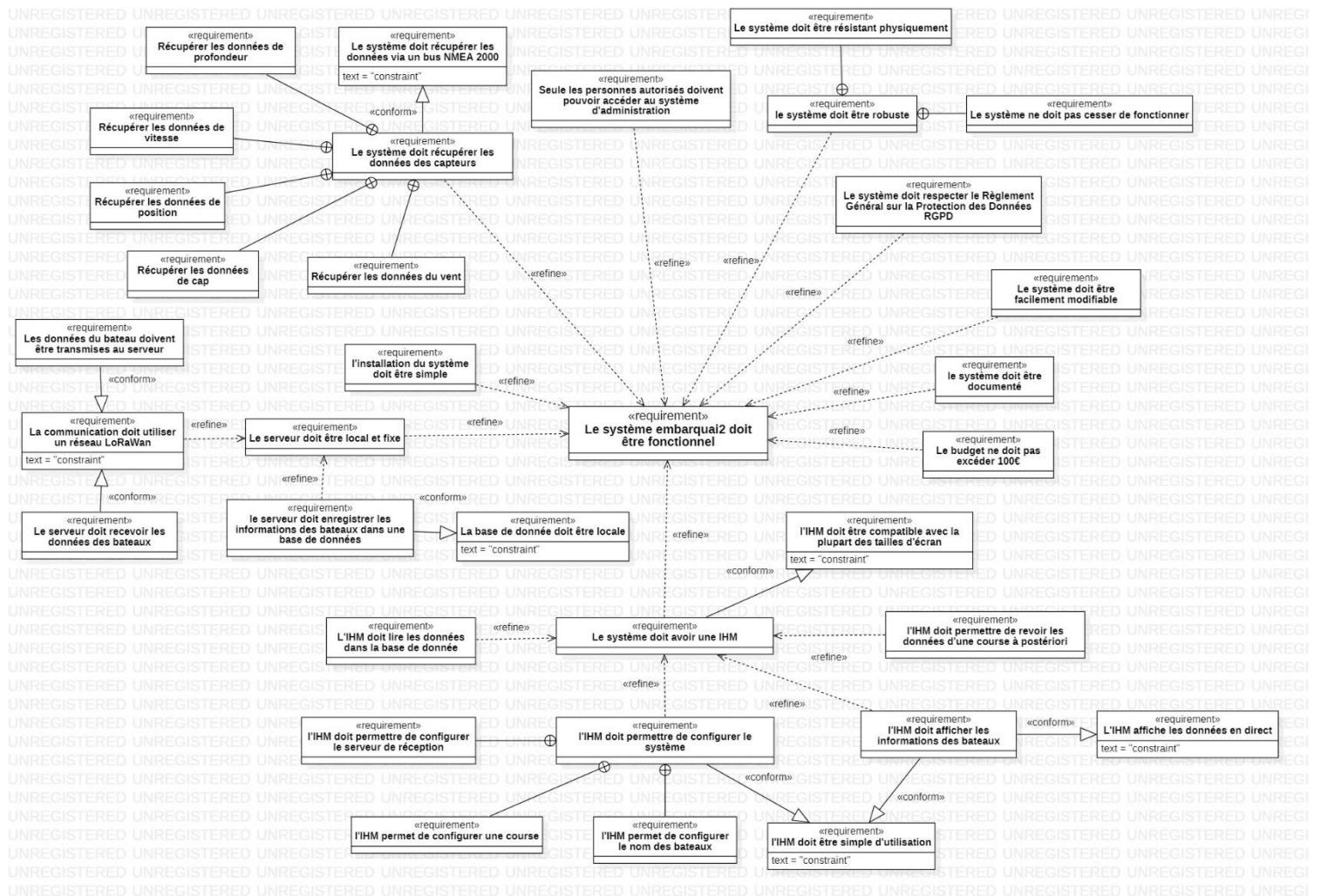
La base de données nous permet de stocker toutes les informations nécessaires à la réussite du projet. Nous avons par exemple l'identifiant du bateau ou encore l'identifiant de la course qui permettent de savoir à qui appartiennent les données que nous recevons et à quelle course elles correspondent. SQLite nous a été imposé comme outil pour créer notre base de données.

La base de données comporte 3 tables :

- Bateau : La liste des noms de chaque bateau.
- Course : La liste des noms de chaque course
- Data_bateau : Stocke l'ensemble des informations récoltées par les capteurs et les paramètres entrés par l'utilisateur.

ANALYSE UML

1. Diagramme des exigences

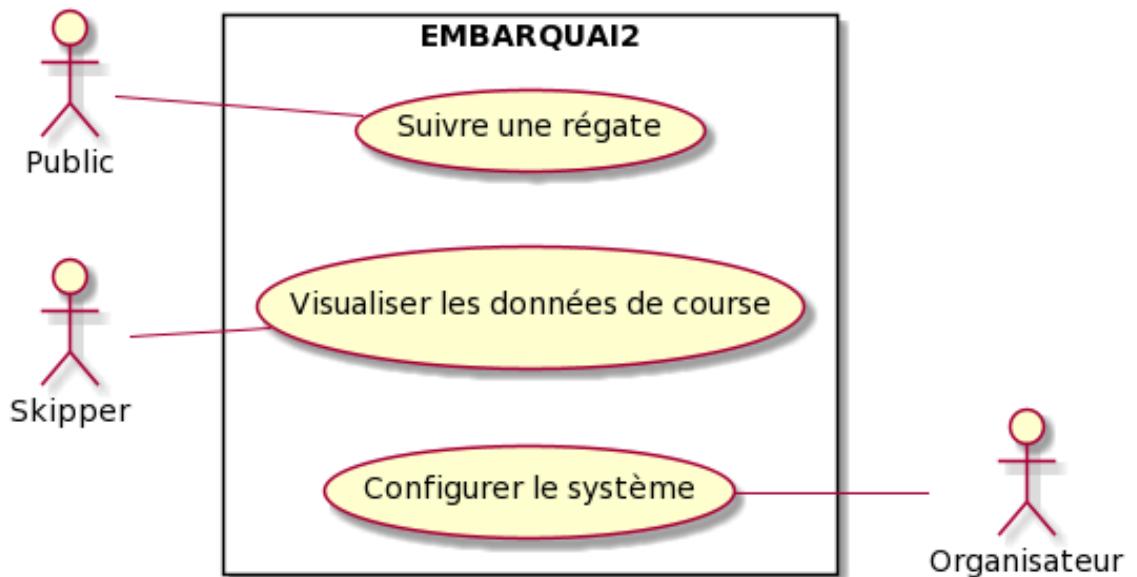


Le diagramme d'exigence est un diagramme spécialement utilisé en SYSML. Il précise les fonctions à réaliser ainsi que le niveau de performance à atteindre. Une exigence comporte un intitulé, un identifiant et une description.

Le diagramme permet de mettre en évidence toutes les exigences attendues sur le projet. De façon générale, notre système doit pouvoir récolter les données des capteurs en utilisant la technologie Lorawan. Et enfin, toutes ces données doivent être exploitées afin de créer une IHM.

- IHM : IHM signifie interface homme machine et fait référence à un tableau de bord qui permet à un utilisateur de communiquer avec une machine, un programme informatique ou un système. En l'occurrence, notre IHM affichera les différentes données récupérées depuis une base de données afin de répondre aux exigences du client.

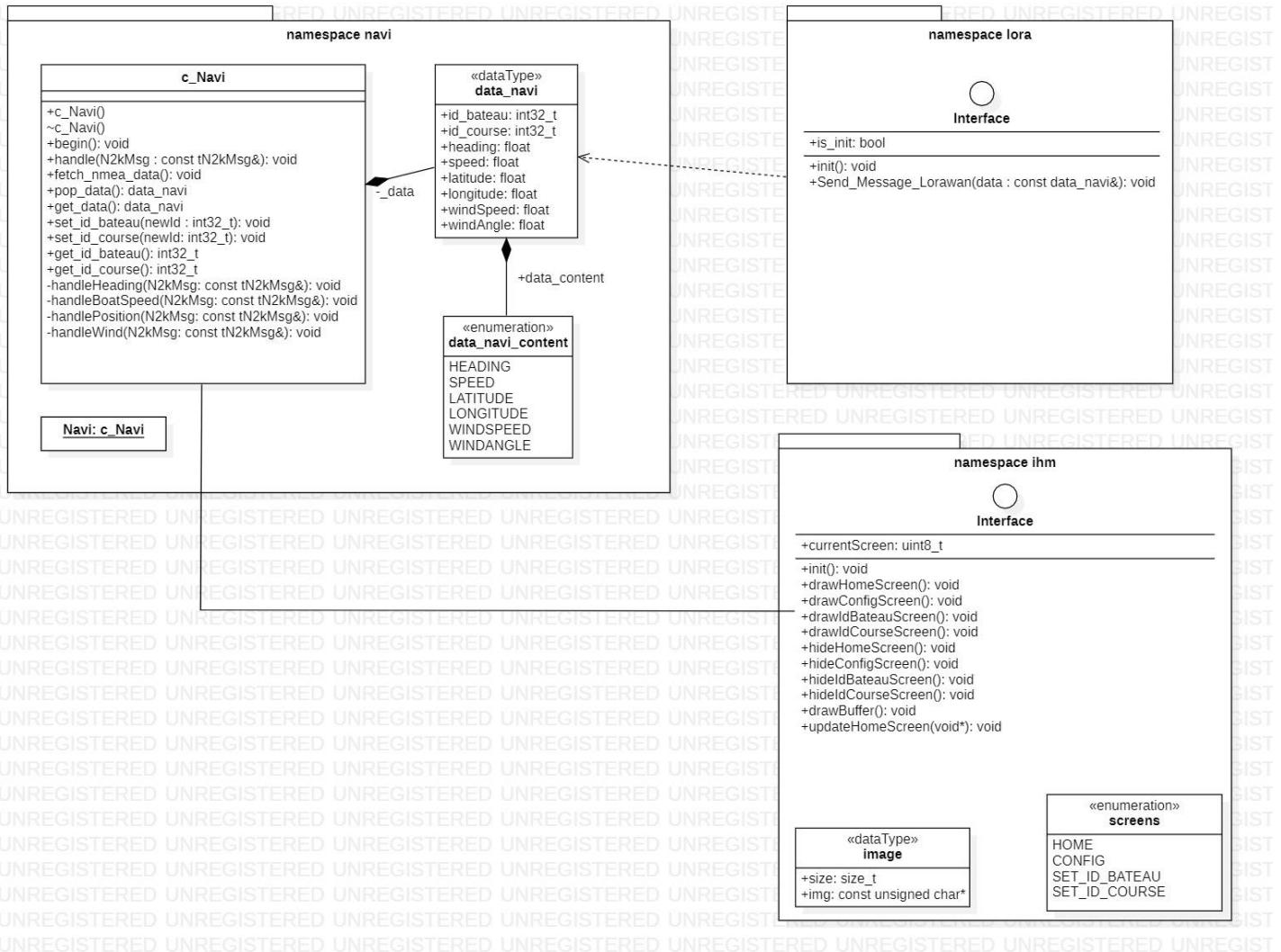
2. Diagramme de cas d'utilisation



Le diagramme de cas d'utilisation est un diagramme UML utilisé pour une représentation du comportement fonctionnel d'un système logiciel. On va pouvoir y modéliser toutes les interactions entre le système et ses utilisateurs.

Sur ce diagramme, nous pouvons observer que le public doit pouvoir suivre le parcours d'une régate (bateau) en temps réel. Le skipper pourra visualiser les données de la course en direct depuis son bateau. Et enfin, l'organisateur pourra effectuer des réglages afin de configurer la course.

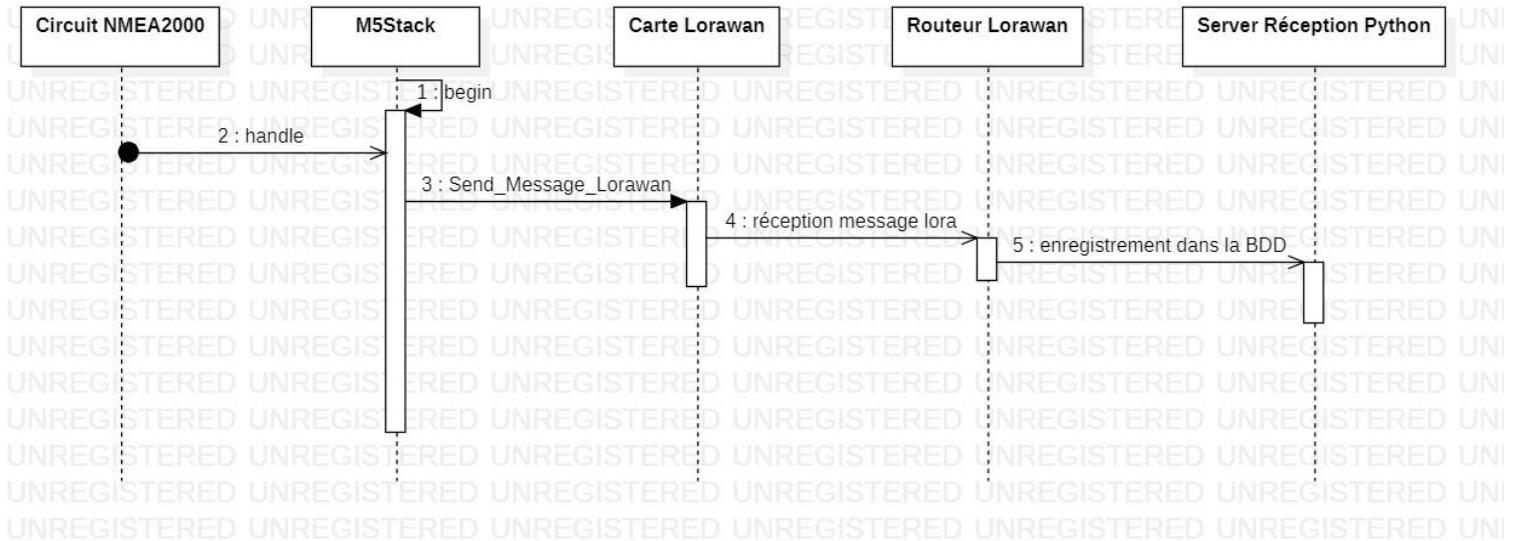
3. Diagramme de classe



Le diagramme de classes est un schéma utilisé en génie logiciel pour présenter les classes et les interfaces des systèmes ainsi que leurs relations.

Le diagramme de classe ci-dessus permet de modéliser et de décrire la classe **c_Navi** qui est utilisée pour récupérer les données des capteurs NMEA2000 depuis le M5stack ainsi que ses interactions avec les autres modules du programme.

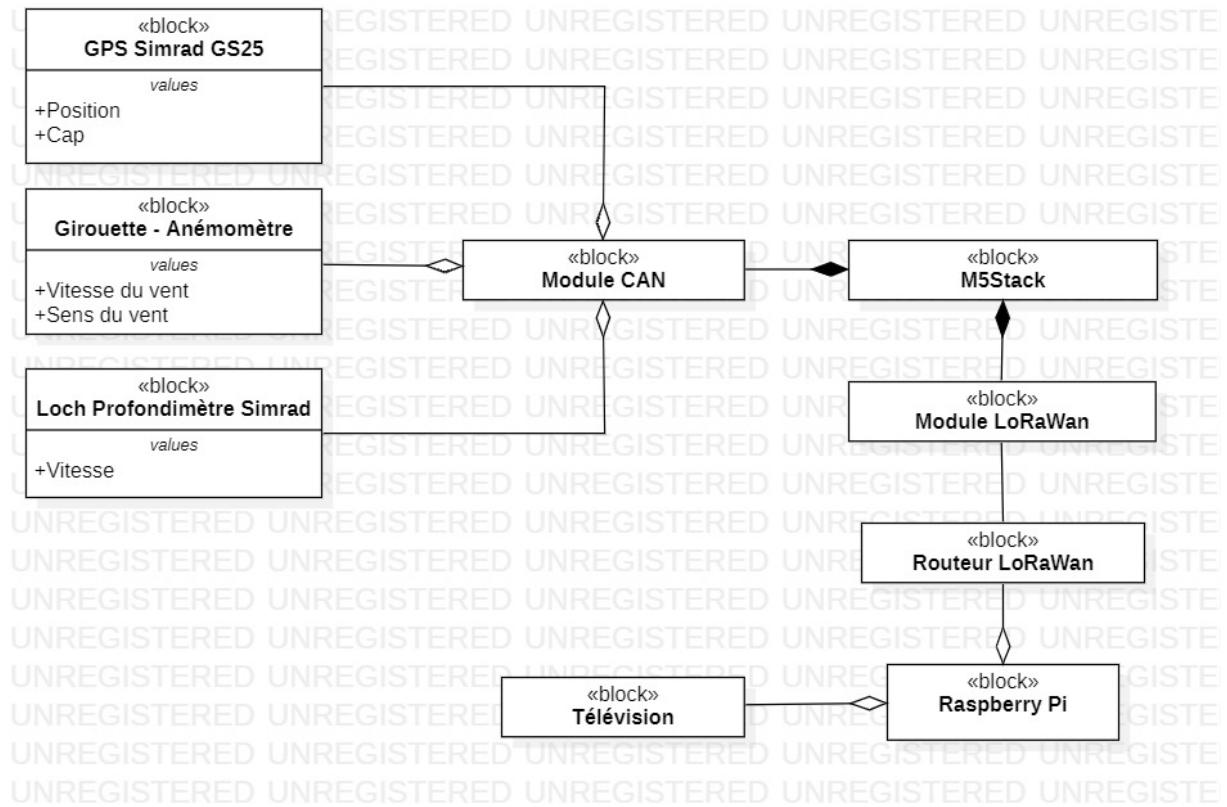
4. Diagramme de séquence



Le diagramme de séquence est la représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique.

Ici, on peut observer toutes les interactions entre le système embarquai2 et les acteurs comme les capteur NMEA2000, le M5Stack ou encore le serveur de réception python dans l'ordre chronologique.

5. Diagramme de définition de bloc



Le diagramme de bloc permet d'exprimer la structure d'un système, d'un sous-système ou d'un composant. Les blocs peuvent représenter des entités physiques où logiques. Ils sont décomposables, ils peuvent posséder des propriétés et un comportement.

Notre diagramme de bloc permet de représenter la structure des composants de notre système embarquai2. Il modélise les composants électroniques présents pour récupérer les données et les envoyer jusqu'au M5stack. Il y a également les composants nécessaires au transfert des données en MQTT du M5stack jusqu'au Raspberry Pi pour être affichés sur une télévision via une IHM.

MATERIELS UTILISES

	Disponibilité	Prix en euro (s)	Description	Quantité	Respect du cahier des charges
M5Stack Core2 : Réf : ESP32	Acheté	40 €	M5Stack est un système de modules composé d'ordinateurs à cartes uniques à destination de l'Internet des objets	2	Oui
Module Lorawan : Réf : UNIT LoRaWan868, ASR6501 868MHz, UART interface pour M5Stack	Acheté	8,50 €	Module permettant de communiquer et de transférer des données avec la norme Lorawan	1	Oui
Module CAN : Ref : CAN (Controller Area Network) Unit SKU:U085	Acheté	9.50 €	Le module Grove CAN bus est un transceiver isolé pour bus CAN (Controller Area Network) pouvant être utilisé pour réaliser de complexes réseaux de communication tels qu'on les trouve dans les voitures.	1	Oui
Routeur Multitech Lorawan : Réf : MTCAP 868	Existant	0 €	Routeur capable de récupérer les données émises par les appareils équipés de modules Lorawan	1	Oui
Raspberry Pi	Existant	0 €	Le Raspberry Pi est un nano-ordinateur monocarte à processeur ARM	1	Oui

Ecran compatible HDMI : Réf : Acer KA2 ecran KA222QA Noir	Existant	0 €	Dalle pour afficher du contenu...	1	Oui
Simrad (Modules et cables NMEA2000) : Ref :.....	Existant	0 €	Matériels nautique utilisés sur les bateaux pour la récupération des données environnantes.	1	Oui
Module Lorawan : Réf : Xbee Waveshare + Carte ATIM LoRaWan	Existant	0 €	Module permettant de communiquer et de transférer des données avec la norme Lorawan	1	Oui

Le cahier des charges et ses contraintes ont été respectés au niveau du matériel acheté et utilisé. Le budget final est inférieur à 100€ donc le cahier des charges a été respecté concernant le budget maximal alloué pour notre projet.

- Budget final -> 98 euros
- Budget maximal -> 100 euros
- $98 < 100 \rightarrow$ Cahier des charges respecté

1. Présentation du M5Stack Core2





M5Stack Core2, un module de développement basé sur l'ESP32 riche en fonctionnalités qui permet de prototyper des idées IoT dès la sortie de la boîte. Ce kit de développement comprend une batterie intégrée et de nombreux autres capteurs et modules intéressants. Et en plus de tout cela, il peut être facilement programmé avec Arduino IDE, Platformio, ou micro-python.

L'unité principale est équipée d'un écran tactile capacitif de 2,0 pouces qui fournit une interface homme-machine fluide et réactive. Le moteur de vibration intégré peut être utilisé pour fournir un retour haptique ou des alertes. Le module RTC intégré fournit une heure précise de la journée. L'alimentation est fournie via une puce de gestion de l'alimentation AXP192, pour surveiller et contrôler les attributs d'alimentation de l'appareil. L'emplacement pour carte TF inclus prend en charge les cartes microSD jusqu'à 16 Go. Le haut-parleur intégré est associé à une puce d'amplificateur de puissance d'interface audio numérique I2S pour réduire la distorsion du signal et fournir une sortie audio plus claire. Il y a des boutons physiques indépendants d'alimentation et de réinitialisation (RST) sur les côtés de Core2, avec 3 boutons tactiles programmables à l'avant de l'écran.

spécification

Ressources	Paramètre
ESP32-D0WDQ6-V3	Double cœur 240 MHz, 600 DMIPS, 520 Ko de SRAM, Wi-Fi
Éclat	16 Mo
PSRAM	8 Mo
Puce de cryptage matériel	ATECC608B-TNGTLSU-G (adresse 0x35)
Tension d'entrée	5V @ 500mA
Interface hôte	TypeC x1, broche POGO x1, I2C x1, GPIO x1, UART x1
Lumière LED programmable	SK6812*10
Bouton	Bouton d'alimentation, bouton RST, bouton d'écran virtuel * 3
Rappel de vibration	Moteur vibrant
Écran LCD IPS	2.0" @320*240 ILI9342C
IC d'écran tactile capacitif	FT6336U
Conférencier	1W-0928
Microphone	SPM1423
Amplificateur de puissance I2S	NS4168
IMU	MPU6886
RTC	BM8563
UGP	AXP192
Puce USB	CP2104
Amplification CC-CC	SY7088
Fente pour carte TF	Prise en charge jusqu'à 16G
Batterie au lithium	500 mAh à 3,7 V
Antenne	Antenne 3D 2.4G
Température de fonctionnement	32°F à 104°F (0°C à 40°C)
Poids net	101g
Taille du produit	54x54x24mm
Taille du paquet	90x60x27mm
Matériau de la coque	Plastique (PC)

LOGICIELS ET LIBRAIRIES UTILISES

1. Logiciels

	Disponibilité	Prix en euro (s)	Description	Respect du cahier des charges
Visual Studio Code	Existant	0 €	Visual Studio Code est un éditeur de code extensible développé par Microsoft pour Windows, Linux et macOS. Les fonctionnalités incluent la prise en charge du débogage, la mise en évidence de la syntaxe, la complétion intelligente du code, les snippets, la refactorisation du code et Git intégré.	Oui
Platformio	Existant	0 €	PlatformIO est un écosystème open source dédié au développement IoT. PlatformIO IDE est l'environnement de développement C/C++ pour les systèmes embarqués supportés. Il est multi-plateformes (Windows, Mac et GNU/Linux) et il fournit une extension à un éditeur de texte existant : soit Atom soit Visual Studio Code.	Oui
Pycharm	Existant	0 €	PyCharm est un environnement de développement intégré utilisé pour programmer en Python	Oui

2. Librairies

	Disponibilité	Prix en euro (s)	Description	Respect du cahier des charges
Librairie M5Stack	Existant	0 €	Librairie réutilisant les librairies déjà existantes du c/c++ adaptées pour l'embarquée	Oui
Librairies Python : <code>from socket import timeout, gaierror</code>	Existant	0 €	Cette librairie est utilisée pour détecter et gérer les timeouts et les problèmes de chaîne de caractères incorrecte sur un fichier	Oui
Librairie Python : <code>from threading import Thread</code>	Existant	0 €	Cette librairie est utilisée pour gérer les Threads et notamment le multithreading	Oui
Librairie Python : <code>import paho.mqtt.client as mqtt</code>	Existant	0 €	Cette librairie permet aux applications de se connecter à un courtier MQTT pour publier des messages, s'abonner à des rubriques et recevoir des messages publiés.	Oui
Librairie Python : <code>import re</code>	Existant	0 €	Cette librairie a été utilisée pour récupérer des informations précises dans des chaînes de caractères.	Oui
Librairie Python : <code>import base64</code>	Existant	0 €	Ce module fournit des fonctions pour encoder des données binaires en caractères ASCII imprimables et décoder ces encodages en données binaires.	Oui

Librairie Python : <code>import sqlite3</code>	Existant	0 €	Cette librairie a été utilisée pour manipuler les bases de données SQLite	Oui
Librairie Python : <code>import configparser</code>	Existant	0 €	Cette librairie a été utilisée pour la création de fichiers	Oui
Librairie Python : <code>import sys</code>	Existant	0 €	Ce module fournit un accès à certaines variables utilisées et maintenues par l'interpréteur, et à des fonctions interagissant fortement avec ce dernier.	Oui
Librairie Python : <code>import logging</code>	Existant	0 €	Cette librairie a été utilisée pour écrire des messages d'erreur dans un fichier	Oui
Librairie Python : <code>import datetime</code>	Existant	0 €	Cette librairie a été utilisée pour récupérer l'heure et la date à l'instant t.	Oui
Librairie Python : <code>import time</code>	Existant	0 €	Ce module fournit différentes fonctions liées au temps.	Oui

SYSTEME D'EXPLOITATION UTILISE

	Disponibilité	Prix en euro (s)	Description	Respect du cahier des charges
Raspbian	Existant	0 €	Système d'exploitation pour Raspberry PI	Oui

LANGAGES DE PROGRAMMATIONS UTILISES

	Disponibilité	Prix en euro (s)	Description	Respect du cahier des charges
C++	Existant	0 €	C++ est un langage de programmation compilé permettant la programmation sous de multiples paradigmes, dont la programmation générique et d'objet.	Oui
Python	Existant	0 €	Python est un langage de programmation interprété, multi-paradigmes et multiplateformes.	Oui
HTML	Existant	0 €	Le HyperText Markup Language, généralement abrégé HTML est le langage de balisage conçu pour représenter les pages web.	Oui
Bash	Existant	0 €	Langage de programmation utilisé sur Linux qui est beaucoup utilisé pour faire des scripts.	Oui
Shell	Existant	0 €	Langage de programmation utilisé notamment pour faire des scripts sur des systèmes UNIX	Oui
SQL	Existant	0 €	Langage de programmation utilisé pour gérer les bases de données en ligne de commande.	Oui

RECUPERATIONS DONNEES CAPTEURS VIA LE M5STACK CORE2

1. Auteur

Cette partie a été rédigée et réalisée par Doriath Arthus.

2. Objectif

L'objectif est de lire et décoder les données du circuit NMEA2000 afin de les rendre utilisables

3. Cahier des charges

- ✓ Les informations des capteurs du bateau doivent être récupérées
- ✓ Le système embarqué doit être compatible avec le standard NMEA2000
- ✓ La carte M5Stack, ainsi que les modules SIMRAD doivent être installés et câblés
- ✓ Le Bus CAN et les modules SIMRAD doivent être configurés

4. Matériels nécessaires

Afin de lire sur le réseau NMEA2000 j'ai évidemment besoin des différents capteurs SIMRAD mais j'aurais également besoin de la carte M5Stack qui constituera la pièce principale de notre système embarqué ainsi que d'un module de décodage CAN.

4.1 la carte M5Stack Core 2

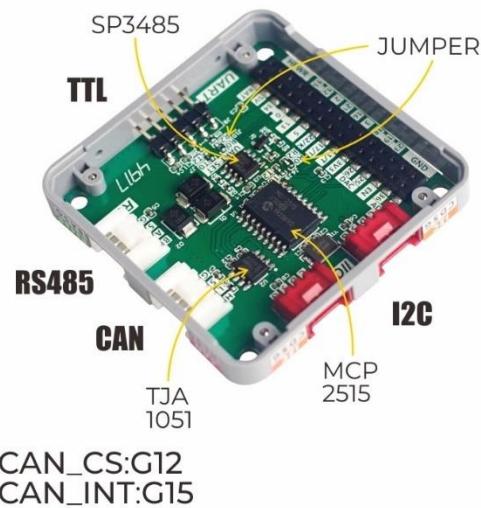
Comme présenté précédemment nous avons décidé de réaliser la partie embarquée du projet sur une carte du type M5Stack Core 2, ce choix était notamment motivé par le fait que les produits de la marque M5Stack sont équipés par défaut, entre autres, d'un écran tactile et de librairies qui simplifient sa programmation. Cela en faisait donc un bien meilleur candidat que les systèmes tels qu'Arduino ou nous aurions eu à entièrement refaire beaucoup de choses. De plus la carte M5Stack est basée sur un ESP32 qui est une puce 32bit équipée de deux coeurs. Nous avons donc bien plus de puissance qu'avec de nombreux autres microcontrôleurs, ce qui nous sera utile pour la création de notre programme.

4.2 le module CAN Unit

Afin de pouvoir lire des données sur le BUS NMEA2000 il nous faudra d'abord être capable d'interfacer le M5Stack avec ce circuit, afin de faire cela il nous fallait un module CAN. Nous avions décidé de nous limiter aux produits de la marque M5 afin de garder un maximum de compatibilités avec notre microcontrôleur, nous avions donc le choix entre le module CAN Unit



Et le module COMMU qui est un module plus complexe et qui est capable de convertir le signal dans encore plus de standards différents, de plus il a l'avantage de pouvoir être connecté en s'empilant sur le M5Stack ce qui aurait facilité son installation



Finalement, le choix qui aura été retenu sera le module CAN Unit car c'est un module plus simple et plus direct d'utilisation, il nous simplifiera donc la tâche.

4.3 Le matériel NMEA2000 SIMRAD

Ce matériel nous était déjà fourni préconfigurer. Il est composé de 3 équipements, un GPS SIMRAD GS25, une Girouette Anémomètre de la même marque ainsi qu'un Profondimètre qui nous intéressera ici pour sa capacité à mesurer la vitesse du bateau.



Le GPS Simrad



L'anémomètre Simrad

BESOIN D'IMAGE DU PROFONDIMETRE

Afin de connecter les différents équipements entre eux nous utiliserons des câbles NMEA2000 de la marque Simrad prévus pour ces différents équipements.

5. Présentation du standard NMEA2000

Le protocole NMEA2000 est un protocole de transmission standardisé basé sur les bus CAN et utilisé pour connecter des capteurs maritimes et des affichages sur un bateau. Les communications s'effectuent à environ 250Kbit / sec et permet à n'importe quel capteur de communiquer avec n'importe quel autre machine compatible NMEA2000.

6. Développement du système

6.1 Environnement de développement

Comme indiqué plus haut afin de programmer nous avons utilisé le combo Visual Studio Code + Platformio ce qui nous a permis d'avoir rapidement un système fonctionnel afin de reprogrammer le microcontrôleur de la carte, en effet platformio installe par lui-même tout le nécessaire afin de compiler et envoyer facilement notre programme sur la carte de notre choix. De plus platformio est équipé d'un système de gestion des dépendances ce qui nous a permis de chercher des librairies à travers son moteur de recherche et de le laisser installer par lui-même celles dont nous avions besoin, cela nous a sans aucun doute permis de gagner énormément de temps.

6.2 Librairie initiale

Lorsque j'ai démarré la programmation de ce module j'ai remarqué qu'il y avait des codes d'exemples fournis par M5Stack, j'ai donc pensé m'en servir afin de développer le code qui serais capable de lires les données entrantes par le module CAN. Cependant je me suis très rapidement confronté à un grand problème, les codes d'exemples fournis n'était pas adaptés à mon problème, en effet il s'agissait de codes de trop bas niveau d'abstractions qui n'implémentaient guère plus que la lecture et écriture basique sur le bus CAN. Bien que cela soit théoriquement plus que suffisant pour réaliser l'application je fus rapidement conscient que si je partais d'aussi peu je risquais de ne pas finir le projet à temps.

```
77 if(xQueueReceive(CAN_cfg.rx_queue,&rx_frame, 3*portTICK_PERIOD_MS)==pdTRUE){  
78     M5.Lcd.fillRect(0, 60, 320, 180, BLACK);  
79     M5.Lcd.setCursor(0, 60, 4);  
80     //do stuff!  
81     if(rx_frame.FIR.B.FF==CAN_frame_std){  
82         printf("New standard frame");  
83         M5.Lcd.printf("New standard frame");  
84     } else{  
85         printf("New extended frame");  
86         M5.Lcd.printf("New extended frame");  
87     }  
88     if(rx_frame.FIR.B.RTR==CAN_RTR){  
89         printf(" RTR from 0x%08x, DLC %d\r\n",rx_frame.MsgID, rx_frame.FIR.B.DLC);  
90         M5.Lcd.printf(" RTR from 0x%08x, DLC %d\r\n",rx_frame.MsgID, rx_frame.FIR.B.DLC);  
91     } else{  
92         printf(" from 0x%08x, DLC %d\r\n",rx_frame.MsgID, rx_frame.FIR.B.DLC);  
93         M5.Lcd.printf(" from 0x%08x, DLC %d\r\n",rx_frame.MsgID, rx_frame.FIR.B.DLC);  
94         for(int i = 0; i < 8; i++){  
95             printf("%c\t", (char)rx_frame.data.u8[i]);  
96             M5.Lcd.printf("%c\t", (char)rx_frame.data.u8[i]);  
97         }  
98         printf("\n");  
99     }  
100 }  
101 }
```

Extrait code Example CAN M5Stack : lecture sur le bus CAN

Fort heureusement pour moi notre environnement de développement, platformio, intègre un système de recherche et d'installation de librairie. J'ai donc recherché sur platformio s'il y avait une librairie qui implémenterait le décodage du bus NMEA2000 et j'ai trouvé.

6.3 Présentation de la librairie NMEA2000

La librairie NMEA2000 est une librairie orientée objet faite pour les microcontrôleurs du type ESP , Arduino ou encore Raspberry. Elle intègre directement des fonctions capables d'écouter puis de décoder un message NMEA2000 et de rendre les différents champs accessibles. Comme nous travaillons sur un ESP32 (le microcontrôleur du M5Stack est un ESP32) l'auteur de la librairie nous conseillait d'utiliser une seconde librairie en plus de la première à savoir NMEA2000_esp32 qui est une librairie qui intègre un driver CAN et hérite des objets de la première librairie et les spécialise pour l'ESP32.

Afin de réagir à la réception de messages NMEA2000 la librairie nous permet de définir notre propre gestionnaire de messages sous la forme d'un callback (ici implémenté avec un pointeur de fonction) nous pouvons alors définir à l'intérieur de ce callback un switch qui se chargera d'appeler les bonnes fonctions de décodage du message en fonction du type de donnée contenue dans la trame, ce qui peut être récupéré à l'aide du PGN (parameter group number).

```
switch(NmeaMessage.PGN){  
    case 127250L:  
        this->handleHeading(NmeaMessage);  
        break;  
  
    case 128259:  
        this->handleBoatSpeed(NmeaMessage);  
        break;  
  
    case 129025L:  
        this->handlePosition(NmeaMessage);  
        break;  
  
    case 129026L:  
        this->handleCogSog(NmeaMessage);  
        break;  
  
    case 130306L:  
        this->handleWind(NmeaMessage);  
        break;  
}
```

Extrait du handler NMEA2000 défini dans la classe c_Navi

6.4 Présentation de la classe c_Navi

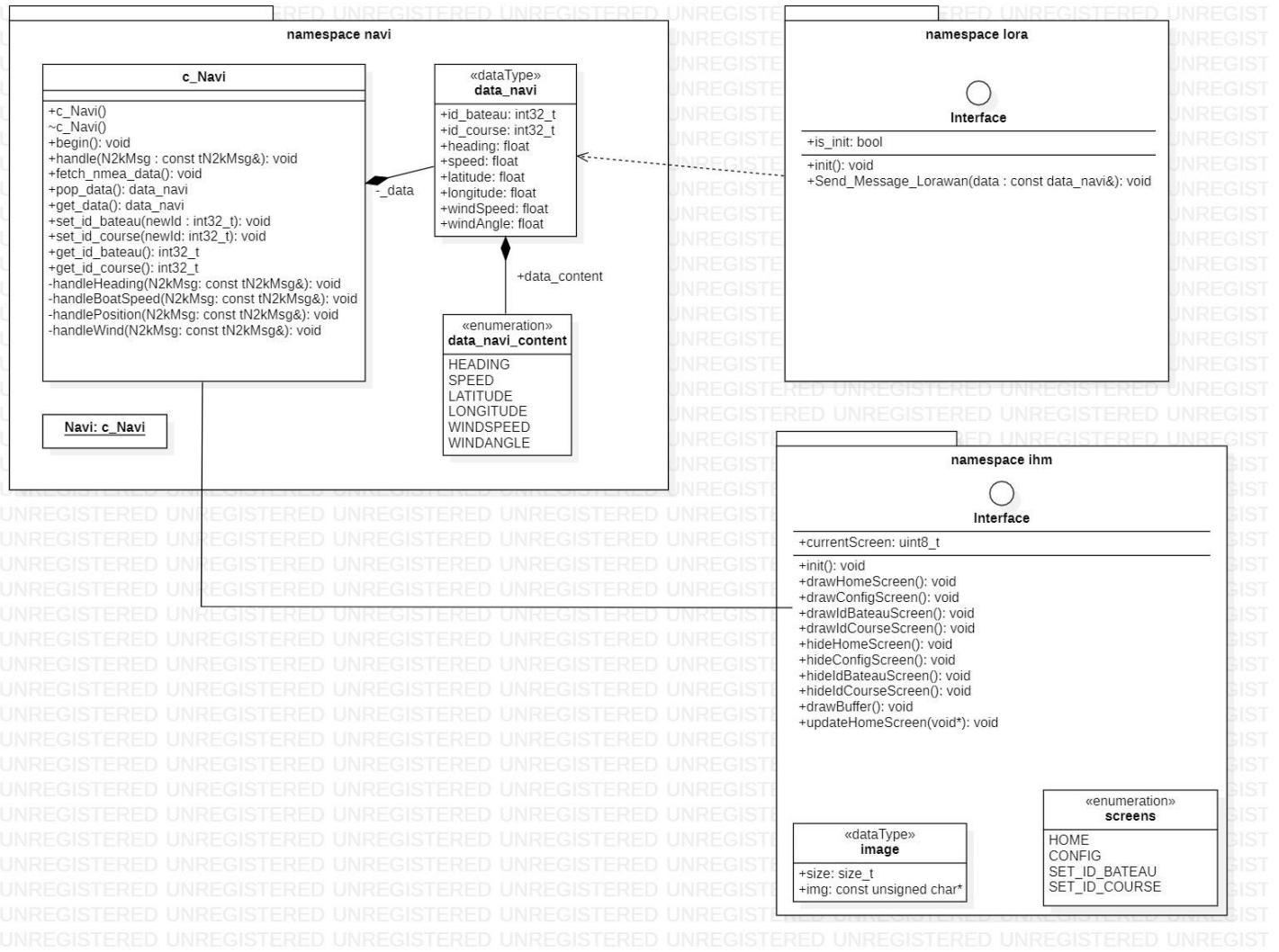


Diagramme de classe du code sur le M5Stack

La classe **c_Navi** est une classe qui a été conçue comme une couche d'abstraction supplémentaire qui utilise la classe NMEA2000, en effet avec la méthode **begin()** on va initialiser le callback de la classe NMEA2000 comme étant la méthode **handle()** d'un objet de la classe **c_Navi** puis on va ouvrir le CAN afin que l'objet NMEA2000 commence à écouter le bus CAN et à transmettre ces messages à la classe **c_Navi**.

Une fois que la classe **c_Navi** a reçu le message NMEA elle va se servir de fonctions définies dans la librairie NMEA afin d'en décodé le contenu puis elle va mettre à jours la structure **data_navi** contenue dans l'objet afin de mémoriser les dernières valeurs.

Lorsque les données sont mises à jours la structure **data_navi** le flag est également mis à jours ce qui permettra plus tard au module Lorawan de savoir quels données sont initialisés ou non.

```

void c_Navi::begin(){

    //s'assure qu'on n'ai pas déjà appellé begin
    if(started) return;

    NMEA2000.EnableForward(false);
    NMEA2000.SetMsgHandler([](const tN2kMsg &N2kMsg){
        Navi.handle(N2kMsg);
    });

    Serial.println();
    while(!NMEA2000.Open()){
        Serial.println("CAN ERROR : can't Open");
        delay(1000);
    }
    Serial.println("CAN OPEN : NAVI STARTED");

    started = true;
}

```

Méthode begin() de la classe c_Navi

```

void c_Navi::handlePosition(const tN2kMsg& N2kMsg) noexcept{

    double latitude = 0;
    double longitude = 0;

    if(ParseN2kPositionRapid(N2kMsg, latitude, longitude) ){

        if(!N2kIsNA(latitude)){
            this->_data.latitude = latitude;
            this->_data.data_content |= navi::data_navi_content::LATITUDE;
        }

        if(!N2kIsNA(longitude)){
            this->_data.longitude = longitude;
            this->_data.data_content |= navi::data_navi_content::LONGITUDE;
        }
    }
}

```

Méthode qui décode un message qui contiens une position et l'enregistre

```


/**
 * @brief une structure qui stocke les données des capteurs d'un bateau
 */
struct data_navi{

    int32_t id_bateau = -1;
    int32_t id_course = -1;

    float heading = 0.0f;
    float speed = 0.0f;
    float latitude = 0.0f;
    float longitude = 0.0f;
    float windSpeed = 0.0f;
    float windAngle = 0.0f;

    //flags pour indiquer le contenu valide de la data
    byte data_content = 0b00000000;
};



```

Le code de la structure data_navi

La structure data_navi sert de conteneur pour les différentes données récupéré par la classe c_Navi mais elle sert aussi de moyen standard pour transférer des données entre le module navi et le module lora, ce qui facilite la fusion des deux.

Une fois que les données sont enregistrées dans la classe Navi l'utilisateur peut y accéder de deux manières différentes. Soit il utilise la méthode get_data() qui va simplement renvoyé les données actuellement présentes dans Navi mais sans changer quoi que ce soit. Soit l'utilisateur peut utiliser la méthode pop_data() qui en plus de renvoyé les données va remettre à zéro le champ data_content du conteneur.

Cela est essentiel au bon fonctionnement du flag et c'est utile afin de détecter lorsqu'un équipement est déconnecté du circuit NMEA

6.5 système de flag

Le système de flag a été fait afin de pouvoir avoir une trace de ce qui est contenu dans la structure data_navi et donc de quels champs sont valides, cela sera très utile lors de l'envoi par le module lora. Mais le flag pourrait également être utilisé par l'ihm afin de prévenir l'utilisateur lorsqu'aucune données n'est reçue de la part d'un certains capteurs ce qui pourraient indiquer que ces derniers sont défectueux ou au moins mal connectés

6.6 Configuration

Le système Navi doit être configurable, en effet l'utilisateur doit être en mesure de configurer l'identifiant du bateau ainsi que l'identifiant de la course à la quel il participe. Cela est implémenté très simplement dans la classe c_Navi en effet cette classe contiens des méthodes mutateurs et accesseur afin de permettre ces accès et modifications. Ces modifications se font dans les écrans de configurations de l'ihm.

ENVOIE DES DONNEES DEPUIS LE M5STACK CORE2

1. Auteur

Cette partie a été rédigée et réalisée par l'étudiant Loris Benaitier.

2. Objectif

L'objectif est de faire transiter les données des capteurs entre le M5Stack (Partie mobile) jusqu'au PC administrateur par le biais du réseau Lorawan.

3. Cahier des charges

- ✓ La communication des données doit utiliser le réseau LoRaWan
- ✓ L'envoi des données doit se faire soit par le biais d'un M5Stack ou d'un Raspberry Pi
- ✓ Envoyer les informations récoltées au poste administrateur pour exploitation à intervalle fixe et configurable
- ✓ Permettre une installation/configuration aisée pour les enseignants ainsi qu'une utilisation facile et documentée pour les élèves.

4. Matériels nécessaires

Pour réaliser cette partie, nous avons choisi d'utiliser le M5Stack Core 2 avec son module Lorawan (UNIT LoRaWan868).

Pour des raisons pratiques et financières, nous avons acheté 2 M5Stack Core 2 afin que l'étudiant 1 (Arthus Doriath) et moi puissions travailler chacun de notre côté en effectuant nos tests.

4.1. Module UNIT LoRaWan868



Specification

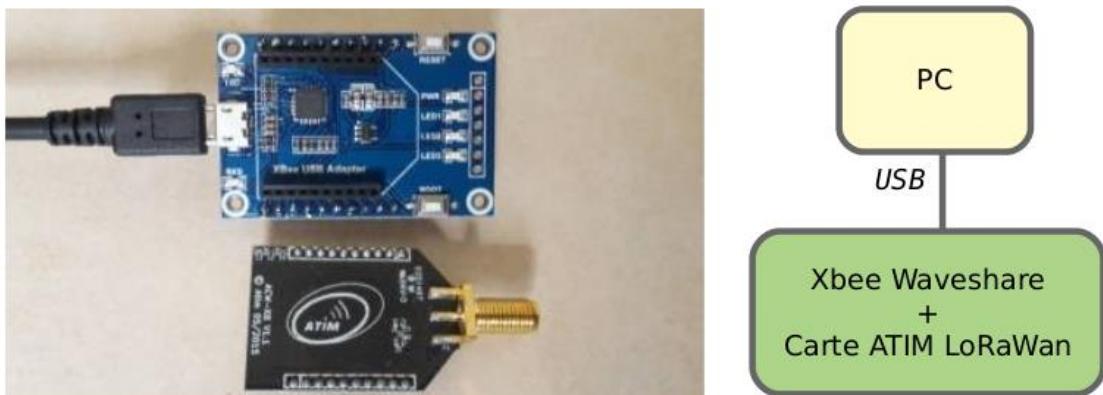
Specifications	Parameters
UART baud rate	115200
Net weight	12.8g
Gross weight	45g
Product Size	71.5*24*8mm
Package Size	95*65*25mm

L'unité LoRaWAN868 est un module de communication LoRaWAN adapté à la fréquence 868 MHz lancé par M5Stack. Le module adopte le schéma ASR6501, qui prend en charge la communication longue distance et a à la fois une consommation d'énergie ultra-faible et une sensibilité élevée.

Le module intègre la pile de protocoles LoRaWAN et adopte une interface de communication série (utilisant le jeu de commandes AT pour le contrôle). Lorsqu'il est utilisé, il peut être utilisé comme nœud de collecte pour accéder à un grand nombre de passerelles pour la collecte et la gestion des données.

Ce module convient aux applications de communication IoT longue distance à faible consommation, telles que le déploiement de nœuds de surveillance environnementale.

4.2. Module Lorawan : Xbee Waveshare + Carte ATIM LoRaWan



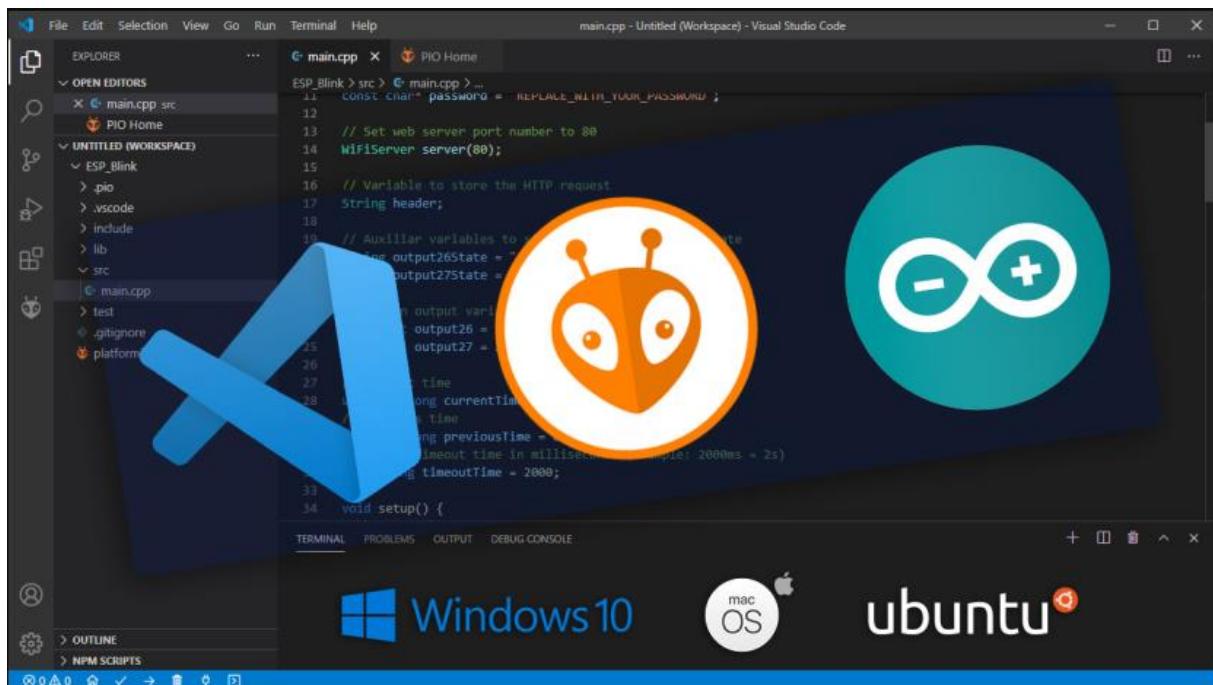
La carte ATIM Lorawan (module Lorawan) est un module permettant de communiquer en Lorawan. La carte XBE Waveshare est un support utilisé pour transférer les données à envoyer sur le module Lorawan (Carte ATIM LoRaWan).

5. Problème rencontré

Après beaucoup de tests, je n'ai pas réussi à faire fonctionner le module Lorawan (UNIT LoRaWan868). Il faut savoir que ce module est extrêmement peu documenté, même après avoir suivi le peu d'instructions que j'avais sur la documentation technique, le module ne fonctionnait toujours pas. J'ai donc envoyé un message au support technique des créateurs de ce module (M5Stack), mais sans succès puisqu'eux-mêmes mon certifié qu'ils ne l'avaient jamais testé.

Par conséquent, pour ne pas prendre de retard sur le projet, j'ai utilisé la Carte ATIM LoRaWan avec son support (Xbee Waveshare) qui était disponible dans mon établissement sans frais supplémentaires.

6. Environnement de développement



```
ESP_Blink > src > main.cpp > ...
11  const char* password = "REPLACE_WITH_YOUR_PASSWORD";
12
13 // Set web server port number to 80
14 WiFiServer server(80);
15
16 // Variable to store the HTTP request
17 String header;
18
19 // Auxilliary variables to
20 // store the output state
21 String output26state = "0";
22 String output27state = "0";
23
24 // Variables to
25 // store the current time
26 long currentTime;
27
28 // Previous time
29 long previousTime = 0;
30
31 // Timeout time in milliseconds
32 long timeoutTime = 2000;
33
34 void setup() {
```

Pour pouvoir envoyer un programme sur le M5Stack Core2, il fallait un IDE capable d'interagir avec lui. Par conséquent, j'ai choisi Plateformio qui est un outil qui peut s'installer sur Visual studio code en tapant son nom dans la barre de recherche des extensions installables. Après l'avoir installé, un logo apparaîtra sur le côté gauche de Visual studio code ou on pourra créer son propre projet selon notre matériel.

L'avantage est qu'une fois le projet créé, toutes les bibliothèques du matériel choisi seront directement importées et disponibles dans l'IDE.

Etant donné qu'on a utilisé un M5Stack Core2, toutes les bibliothèques liées à cette carte ont été importées. Les librairies de cette dernière étant codées en c++, tout mon code sur le M5Stack Core2 sera en c++.

7. Présentation de Lora et Lorawan

7.1. Introduction

Développés par la start-up grenobloise Cycleo, LoRa et LoRaWAN sont des protocoles majeurs de l'internet des objets. Véritable révolution dans le domaine, cette technologie facilite la communication radio entre les appareils connectés.

7.2. Qu'est-ce que LoRa ?

LoRa (*Long Range*) est une technologie de communication radio bas débit et très longue portée. C'est la couche physique d'un réseau étendu à faible puissance et longue distance (LPWAN). Améliorant le système de modulation par déplacement de fréquence (FSK), LoRa se base sur la modulation à spectre étalé (*chirp*).

Ce protocole propriétaire offre ainsi une meilleure portée de la communication, avec une puissance tout aussi faible que les autres systèmes et une bonne résistance aux interférences.

7.3. Qu'est-ce que LoRaWAN ?

LoRaWAN (*Long Range Wide-area network*) est l'architecture du système, le réseau étendu à faible puissance et longue distance. Ce protocole permet la communication à longue portée et faible débit entre des objets connectés ayant une faible consommation électrique. Plus précisément, LoRaWAN aide ces appareils à communiquer avec les serveurs d'applications.

En amont, tous les appareils connectés au réseau possèdent une puce LoRa qui établit la communication avec des passerelles (gateways) en utilisant la modulation par étalement de spectre. Ces passerelles sont des stations de base pourvues d'antennes et connectées à internet. Elles servent alors d'intermédiaires pour contacter le serveur.

Tandis que la couche physique de modulation LoRa établit la communication à longue portée avec les passerelles, le protocole IP prend le relais entre les gateways et le serveur applicatif.

Ainsi, LoRaWAN ne crée pas directement la communication entre des objets intelligents. Il fournit une architecture leur permettant de dialoguer avec le serveur applicatif – via les passerelles et la technologie LoRa.

7.4. Quels sont les avantages des protocoles LoRa et LoRaWAN ?

L'avantage principale de LoRa et LoRaWAN est leur grande portée, associée à une puissance aussi faible que les protocoles concurrents. De ce fait, une seule passerelle ou station de base peut couvrir des villes entières ou des centaines de kilomètres carrés. Avec un minimum d'infrastructure, il est alors possible de couvrir facilement des pays entiers.

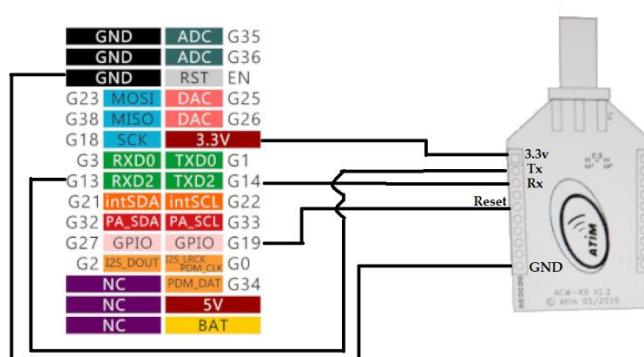
8. Liaison physique : module Lorawan/M5Stack Core2

Afin de relier le module Lorawan avec le M5Stack, il faut brancher la carte ATIM LoRaWan sur les pins de la partie supérieure de la Xbee Waveshare (Du côté où se trouve la prise pour une liaison sur un ordinateur). Puis, relier les pins qui se trouvent en dessous de la carte Xbee Waveshare sur les pins disponibles dans le M5Stack Core2 comme sur le schéma :

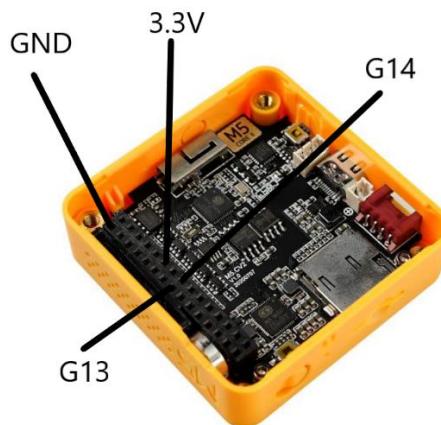
Montage provisoire pour les tests :

PINS M5Stack Core 2 :

Carte ATIM LoRaWan + Xbee Waveshare :



Les pins dans le M5Stack sont accessibles en séparant les deux parties qui le compose qui sont maintenues par des vis.



Montage Finale :



9. Choix du format de la trame contenant les données

Le format de la trame de données est extrêmement important puisque ça va déterminer la taille finale du message et la façon dont on pourra récupérer les données par la suite. On verra également que tout ça aura une incidence sur le taux d'erreur des messages récupéré par le serveur python.

Format trame : b'@Flag;id_course;id_bateau;Donnee_1;Donnee_2;!CRC~'

9.1. Caractères au début et à la fin de la trame de données

Afin que le message soit envoyé, il est impératif de mettre au début de la trame le caractère **b** et le caractère **'** à la suite. Puis à la fin de la trame, il faut rajouter le caractère **'**. Ces caractères sont obligatoires car cela va avoir pour effet d'encoder la trame en UTF8 qui est le format d'envoi standard des messages utilisant le protocole Lorawan. Si on n'utilise pas ce format d'envoi, les messages envoyés seront complètement corrompus et inexploitables.

9.2. Caractères de séparations entre chaque champ de la trame

Le choix des caractères de séparations entre les différents champs ne sont pas obligatoires pour l'envoi d'un message. Par conséquent, nous avons choisi les caractères les plus propices afin d'avoir un message clair et facile à décoder pour mon serveur python.

- **@** et **!** sont les deux caractères qui séparent le flag et les données avec le CRC
- Le caractère **;** sera ajouté juste après le champ flag et chaque champ de données
- Le caractère **~** sera ajouté juste après le champ CRC

9.3. FLAG

Le flag a été mis en place car étant donné que les capteurs ne donnent pas toujours de valeurs, on peut avoir seulement deux valeurs sur les 6 qu'on souhaite envoyer. La deuxième raison est que si jamais par exemple, la donnée_1 est la même que l'ancienne qui a été envoyée, alors ça ne sert à rien de renvoyer cette valeur. Et enfin, le fait de n'envoyer que les données disponibles et utiles réduit considérablement la taille de la trame de données. Par conséquent, le taux d'erreur et de corruption diminue lui aussi drastiquement.

Flag : 0b00000000;

- Le Flag est composé de bit à 0 ou à 1 avec 8 champs donc 2 champs inutiles à la fin. Si le champ est à 1, alors la donnée a été mise à jour, sinon, la donnée n'est pas disponible donc pas la peine d'envoyer cette donnée.
- Le '0b' dans le flag veut dire que le flag est en binaire (0x-> en hexadécimale...)
- Le flag est contenu dans un byte qu'on va récupérer dans la structure data_navi du namespace navi qui est mis à jour par mon coéquipier Arthus Doriath en fonction de ce qu'il reçoit.
- L'ordre de lecture se lit de droite à gauche et pour savoir quelle donnée correspond à quel bit dans le flag, on se réfère aux champs de l'énumération data_navi_content qui est dans le namespace navi.
- Les champs donnés sont récupérés dans la structure data_navi qui est dans le namespace navi. Les champs donnés sont mis dans la trame dans l'ordre de lecture du flag, comme ça, on peut, en réutilisant la méthode inverse, retrouver à quoi correspondent chaque valeur sur le serveur python qui traitera la trame de données.
- Afin que le serveur python puisse savoir à son tour quelle donnée correspond à quels champs (altitude, vitesse...), on doit lui passer le flag dans la trame message. Par conséquent, pour réduire la taille de la trame, on va passer la valeur décimale du flag au lieu de passer le flag brut.
- Pour la vérification des bits dans le flag, on effectue un traitement bit à bit.

Extrait code qui gère le flag « fonction_send_lora.cpp » :

```
/*
* @brief Fonction qui va remplir une chaîne de caractère contenant les infos mise à jour en fonction de la valeur du flag.
* @param data structure qui contient les différents champs qui contiennent les données des capteurs
* @return String, Chaîne de caractère contenant la data (flag + donnée(s) capteur(s))
*/
String lora::Traitement_flag_data(const navi::data_navi& data)
{
    // Variable locale
    String message{""};

    //Traitement avec un 'ET logique' pour voir si il y a un bit à 1 ou pas sur chaque bit du flag
    if(data.data_content & navi::data_navi_content::HEADING)
    {
        message += String(data.heading, 2) + ";";
    }

    if(data.data_content & navi::data_navi_content::SPEED)
    {
        message += String(data.speed, 2) + ";";
    }

    if(data.data_content & navi::data_navi_content::LATITUDE)
    {
        message += String(data.latitude, 5) + ";";
    }

    if(data.data_content & navi::data_navi_content::LONGITUDE)
    {
        message += String(data.longitude, 5) + ";";
    }

    if(data.data_content & navi::data_navi_content::WINDSPEED)
    {
        message += String(data.WindSpeed, 2) + ";";
    }

    if(data.data_content & navi::data_navi_content::WINDANGLE)
    {
        message += String(data.WindAngle, 2) + ";";
    }
}
```

Extrait du code qui gère la conversion base 2 à Base 10 « fonction_send_lora.cpp » dans la fonction « Traitement_flag_data » :

```
String(static_cast<int>(data.data_content), 10)
```

9.4. Données

La trame peut contenir au maximum 6 champs de données récupérées grâce aux capteurs. Il y a id_course et id_bateau qui seront configurables et placés juste avant les données capteurs pour savoir à qui appartiennent ces données.

Les données capteurs sont : Heading, Speed, Altitude, Longitude, WindSpeed, WindAngle.

Toutes ces données seront récupérées depuis la structure data_navi du namespace navi en fonction du flag.

Extrait code qui gère la concaténation des données « fonction_send_lora.cpp » dans la fonction « Traitement_flag_data » :

```
//Formation de la data
String config = String(data.id_course, 10) + ";" + String(data.id_bateau, 10);
String trame = String(static_cast<int>(data.data_content), 10) + ";" + config + ";" + message;
```

- Message -> Données bateau récupérées en fonction du flag
- Data_content -> Nom de la variable de type 'byte' qui contient le flag (Transformation base 2 -> base 10)

9.5. CRC

Pour que le serveur python qui sera charger de traiter les données qui ont été envoyé puisse savoir si elles sont complètes ou non, j'ai mis en place un algorithme de CRC qui permet en autre de vérifier si le message reçu est conforme par rapport à l'original (avant son envoi).

Le principe du CRC est très simple, on prend les données sur lesquelles on veut appliquer le CRC, on les convertit en binaire, on passe la valeur binaire + une clé binaire qu'on a défini au préalable dans la fonction generation_crc qui va générer comme son nom l'indique un CRC en binaire. Et enfin, on met ce CRC dans la trame donnée. Une fois que le serveur python qui traite les données à reçu cette trame, il va convertir les données ou le CRC y est appliqué en binaire et les passer dans la même fonction 'generation_crc' avec la même clé binaire que celle utilisée pour l'envoi. Pour finir, on va comparer le CRC trouver et le CRC contenu dans la trame et voir s'ils sont égaux. S'ils ne sont pas égaux, alors les données sont corrompues, sinon, elles sont conformes et peuvent être utilisées.

Extrait code qui génère un CRC « fonction_send_lora.cpp » :

```
/**
 * @brief Generation d'un string contenant le message combinant une clé pour donner une valeur d'identification en cas de perte de données
 * @param key
 * @param data
 * @return String
 */
String lora::generation_CRC(const String& data, const String& key)
{
    int l_key = key.length();

    // Ajoute n-1 zéros à la fin des données
    String appended_data = data + add_bit(l_key - 1);

    String remainder = mod2div(appended_data, key);

    return remainder;
}
```

- Donnée en binaire + clé binaire définies au préalable en rajoutant des bits à 0 selon la taille de la clé choisie à l'aide de la fonction add_bit.

- On fait appel à la fonction mod2div qui va prendre en paramètre le résultat précédent et la clé en binaire pour générer le remainder (reste == CRC)
- Les données passées dans cette fonction seront les suivantes : **flag;id_course;id_bateau;donnee_1;donnee_2...;** le tout convertit en binaire.

Extrait code -> Fonction qui va diviser les données binaires par la clé et stocker le reste de la division « fonction_send_lora.cpp » :

```
/*
 * @brief Fonction qui va diviser les données binaires par la clé et stocker le reste de la division
 * @param divident
 * @param divisor
 */
String lora::mod2div(const String& divident, const String& divisor)
{
    // Nombre de bits à XORer à la fois.
    int i = divisor.length();

    // Découpage du diviseur pour s'approprier
    // longueur pour une étape particulière
    String tmp = divident.substring(0, i);

    int length = divident.length();

    while (i < length)
    {
        if (tmp[0] == '1')

            // Remplace le diviseur par le résultat
            // de XOR et tirez 1 bit vers le bas
            tmp = xor1(divisor, tmp) + divident[i];
        else

            // Si le bit le plus à gauche est '0'.
            // Si le bit le plus à gauche du dividende (ou le
            // partie utilisée à chaque étape) est 0, l'étape ne peut pas
            // utiliser le diviseur normal ; nous devons utiliser un
            // diviseur composé uniquement de 0.
            tmp = xor1(add_bit(i), tmp) + divident[i];

        // Incrémenter la sélection pour aller plus loin
        i += 1;
    }

    // Pour les n derniers bits, il faut l'exécuter
    // normalement, car l'augmentation de la valeur de i entraînera
    // Index hors limites.
    if (tmp[0] == '1')
        tmp = xor1(divisor, tmp);
    else
        tmp = xor1(add_bit(i), tmp);

    return tmp;
}
```

Extrait code -> Fonction qui va faire un ou exclusif entre deux types string « fonction_send_lora.cpp » :

```
/**  
 * @brief Fonction qui va faire un xor(ou exclusif) entre deux String  
 * @param a  
 * @param b  
 * @return String  
 */  
String lora::xor1(const String& a, const String& b)  
{  
    // Initialise le resultat  
    String result = "";  
  
    int n = b.length();  
  
    // Parcourt tous les bits, si les bits sont  
    // pareil, alors XOR vaut 0, sinon 1  
    for(int i = 1; i < n; i++)  
    {  
        if (a[i] == b[i])  
            result += "0";  
        else  
            result += "1";  
    }  
    return result;  
}
```

Extrait code -> Fonction qui va s'occuper de faire la conversion binaire d'une chaîne de caractère « fonction_send_lora.cpp » :

```
/**  
 * @brief Conversion d'une chaîne de caractère en binaire  
 * @return String  
 */  
String lora::conversion_binaire(const String& data)  
{  
    int N = data.length();  
    String bin = "";  
  
    for(auto i = 0; i <= N; i++)  
    {  
        //Convertir chaque char en ASCII  
        int val = int(data[i]);  
  
        //Convertir ASCII en binaire  
        String value_Bin = "";  
        while(val > 0)  
        {  
            (val % 2)? value_Bin += '1':  
            |           | value_Bin += '0';  
            val /= 2;  
        }  
        std::reverse(value_Bin.begin(), value_Bin.end());  
        bin += value_Bin;  
    }  
    return bin;  
}
```

Extrait code qui s'occupe d'appeler les fonctions nécessaires pour générer le CRC et le placer dans la trame « fonction_send_lora.cpp » dans la fonction « Send_Message_Lorawan » :

```
message = Traitement_flag_data(data);

//Convertir le message en binaire
String value_bin = conversion_binaire(message);

//Generation CRC
String crc_bin = generation_CRC(value_bin, key);

//Concatener le crc à la fin du message
Borne += message + "!";
Borne += crc_bin + "~"; // Fin message
```

- On place crc_bin juste après les données (flag + id_course + id_bateau) suivi du caractère ~.

10. Envoi de la trame contenant les données

On envoie la trame à l'aide de la commande Serial2.println qui correspond à l'envoi de données pour le port C (G13 = RXD2 et G14 = TXD2), le tout dans un type string.

Extrait code qui envoie message « fonction_send_lora.cpp » :

```
/*
 * Fonction qui va envoyer une chaîne de caractère sur un module lorawan.
 * @param struct donnee structure ui contiendra les différents champs qui contiendront les données des capteurs
 * @return void
 */
void lora::Send_Message_Lorawan(const navi::data_navi& data)
{
    //si le serial2 n'a pas été initialisé on ne fait rien
    if(!is_init) return;

    String message="";
    String Borne="";
    String key = "101010001"; // Choisir cle en binaire

    // Concaténation des différentes valeurs du capteurs
    Borne += "b'@"; // Début message(obligatoire pour l'envoie)
    message = Traitement_flag_data(data);

    //Convertir le message en binaire
    String value_bin = conversion_binaire(message);

    //Generation CRC
    String crc_bin = generation_CRC(value_bin, key);

    //Concatener le crc à la fin du message
    Borne += message + "!";
    Borne += crc_bin + "~"; // Fin message

    Serial.println(Borne); // DEBUG
    Serial2.println(Borne); // Trame message -> Serial2.println("b'&Loris;Loris&'");
    //Serial2.println("b'@9;1;0;9.0;48.25868;!01000101~"); //DEBUG, exemple de données qui respectent le format de la trame
    delay(2000);
}
```

- Le delay choisi entre chaque envoi est de 2 secondes car en dessous de cette valeur, il y a beaucoup trop de messages qui sont corrompus et donc inexploitable.
- Après avoir rencontré le client, il m'a dit que ce n'était pas nécessaire que le temps entre l'envoie de chaque message soit configurable donc cette fonctionnalité n'a pas été implémenté.

11. Test unitaire pour l'envoi de la trame contenant les données

Grace au dossier test qui est fourni avec Platformio, en exécutant une commande spécifique dans le terminal, on peut lancer ses propres test sans utiliser la fonction Setup() et la fonction loop() principale.

Dans ce cas précis, on va pouvoir tester l'envoi des données grâce à cet outil.

Commande de base : pio test -e m5stack-core2 -f nom_dossier_a_tester --verbose

Commande : pio test -e m5stack-core2 -f test_LoRaWan --verbose

Extrait code source pour tester l'envoi des données « test_lora.cpp » :

```
* @brief fonction d'initialisation du programme
* @param none
* @return none
*/
void setup() {
    M5.begin();
    Serial.begin(9600);

    UNITY_BEGIN();
    RUN_TEST([](){
        TEST_ASSERT_EQUAL(lora::is_init, false); //test qui vérifie que is_init est bien initialisé à false
    });

    lora::init();

    RUN_TEST([](){
        TEST_ASSERT_EQUAL(lora::is_init, true); //test qui vérifie que is_init passe bien à true après init()
    });
}

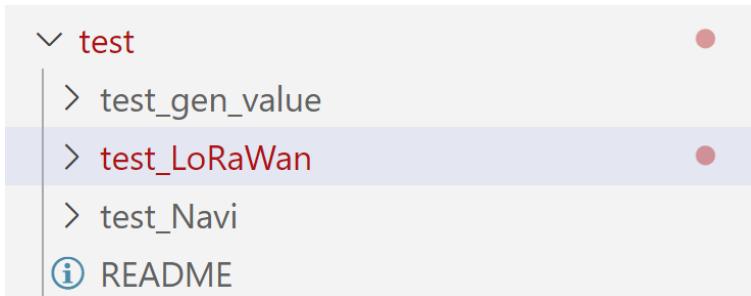
/**
* @brief boucle d'exécution du programme
* @param none
* @return none
*/
void loop() {
    static int compteur = 0;
    navi::data_navi data; // Structure de donnée capteur
    lora::Send_Message_Lorawan(data);

    Serial.println("ok1");
    delay(100);
    compteur++;

    if(compteur > 1000){UNITY_END();}
}
```

- On appelle 1000 fois la fonction Send_Message_Lorawan

Extrait de l'arborescence du projet avec le dossier test_LoRaWan de Platformio :



Résultat :

Après le lancement du test, aucune erreur de compilation n'est apparue et les tests n'ont révélé aucune erreur. Le message ok1 envoyé sur le Serial pour s'assurer que la boucle fonctionnait était présent.

De plus, l'affichage du message complet dans la fonction Send_Message_Lora à l'aide de la commande Serial.println(born) s'affichait correctement et était conforme à la structure de la trame voulue.

✓ Test validé

12. Test unitaire pour la génération de fausse valeur

J'ai mis en place ce test pour deux raisons :

- Si l'étudiant 1 (Arthus Doriath) ne parvenait pas à récupérer les valeurs des capteurs
- Faire des tests d'envoi avec des valeurs qui varient pour être en condition réelle sans dépendre de l'étudiant 1 (Arthus Doraith) pour avancer.

Code pour la génération de fausse valeur « fonction_gener_value.h » :

```
/** Cette fonction est generique, va generer des valeurs aleatoire pour simuler les valeurs des capteurs.
 * @param typename-T choix du type pour l'utilisateur
 * @param size_t nombre valeur a generer
 * @return std::array<T> tableau contenant les donnees capteurs
 */
template<typename T, size_t capacity>
std::array<T, capacity> generate_test_values_sensors_number()
{
    //Creation tableau
    std::array<T, capacity> tab;

    //Creation valeur random entre 1 et 200(div par 100 == float)
    for(int i = 0; i<capacity; i++)
    {
        tab[i] = random(100,1000);
        tab[i] /= 10;
    }

    //Retourner tableau
    return tab;
}
```

Code pour tester si ma fonction me donne bien les résultats attendus « test_gen_value.h » :

```
/*Fonction pour tester si la generation de valeur est correcte
*@param : void
*@return: void
*/
void test_gen_number()
{
    //Ajout valeurs generées par les fonctions dans tableau pour le test
    std::array<float, 10> tab;
    std::array<float, 10> tab1;
    tab = generate_test_values_sensors_number<float, 10>();
    tab1 = generate_test_values_sensors_number<float, 10>();//2

    //Boucle de test
    for(uint8_t i = 0; i<tab.size(); i++)
    {
        //Si first == second la valeur est fausse, le test échouera.
        TEST_ASSERT_NOT_EQUAL(tab[i], tab1[i]);

        Serial.print(tab[i]);
        Serial.print(" ");
    }

    //Test pour savoir si la taille demande est correcte
    //Si first != second la valeur est fausse, le test échouera
    TEST_ASSERT_EQUAL(tab.size(), tab1.size());

    //Test pour savoir si le type est correcte
    bool type = false;
    if(sizeof(tab[1]) == sizeof(float) && sizeof(tab1[1]) == sizeof(float)){type = true;}
    TEST_ASSERT_TRUE_MESSAGE(type, "Type correcte");
}
```

Extrait code pour lancer le test de ma fonction de génération de nombre « main_test_genValu.cpp » dans la fonction loop :

```
RUN_TEST(test_gen_number);
delay(500);
```

Extrait résultat compilation test fausse valeur :

```
52.1 12.1 58.2 45.5 68.5 54.1 82.1 98.2 75.5 18.5
```

- On obtient bien des valeurs aléatoires sans aucune erreur de compilation, tous les tests sont passés.
- test_gen_value = PASSED
- Si jamais il y avait un test qui ne passait comme celui du type par exemple, on obtiendrait une erreur « test_gen_value = Error »
- Commande utilisée : pio test -e m5stack-core2 -f test_gen_value --verbose

Grace à ce test, je peux maintenant utiliser ma fonction de génération de fausses valeurs en condition réelle pour simuler les valeurs des capteurs en étant sur quelle génèrent des résultats conformes.

- ✓ Test validé

13. Intégration avec l'étudiant 1

Pour fusionner mon module « Envoi des données depuis le M5Stack Core 2 » et le module « récupération données capteurs depuis le M5Stack Core2 » de l'étudiant 1, il suffira que l'étudiant 1 appelle ma fonction Send_Message_Lorawan en lui passant la structure data_navi qu'il aura mise à jour préalablement. Etant donné que le premier module Lorawan (UNIT LoRaWAN 868) ne fonctionnait pas, j'ai utilisé le module ATIM Carte Lorawan + Xbee Waveshare. Cependant, la liaison se fait seulement avec les pins qui sont situés à l'intérieur du M5Stack Core2. Par conséquent, l'étudiant 1 ne pouvait plus accéder à son port pour connecter son module Bus CAN. De ce fait, il a fallu concevoir une carte électronique pour que nos deux modules puissent s'intégrer ensemble proprement au niveau hardware.

14. Conclusion

Le cahier des charges sur cette partie est validé entièrement car le module « Envoi des données depuis le M5Stack Core2 » est totalement fonctionnel et répond aux attentes du client et du cahier des charges.

RECUPERATION DES DONNEES ENVOYEEES PAR LE M5STACK CORE2

1. Auteur

Cette partie a été rédigée et réalisée par l'étudiant Loris Benaitier.

2. Objectif

L'objectif de cette partie est de réussir à récupérer les trames contenant les données envoyées depuis le M5Stack Core2 sur le pc administrateur (Poste fixe).

3. Cahier des charges

- ✓ Envoyer les informations récoltées au poste "fixe" donc le pc administrateur

4. Matériels et logiciels nécessaires

Afin de pouvoir récupérer les trames contenant les données, nous avons besoin d'un routeur qui est capable d'intercepter et de récupérer les trames Lorawan émises par des appareils qui sont équipés d'un module Lorawan. Nous avons également besoin d'un Raspberry pi pour respecter le cahier des charges avec l'OS Raspbian. Cependant, n'importe quel autre ordinateur avec un os différent marcherait aussi puisque le système est multiplateforme.

- Raspberry pi avec OS Raspbian
- Routeur multitech Lorawan (MTCAP 868)
- Pour réaliser le code en python, j'ai utilisé l'IDE Pycharm qui est spécialisé pour le langage de programmation python.

4.1. Routeur multitech MTCAP 868

MTCAP Specifications

Category	Description
General	
Standards	LoRaWAN 1.0.2 specifications
	LTE 3GPP Release 9
	HSPA+ with GPRS fallback
RAM	256MB
Flash	256MB
Radio Frequency	
ISM Band	915 MHz ISM band for US and Canada
4G/LTE	1900 (B2) / AWS 1700 (B4) / 850 (B5) / 700 (B12/13)
3G	1900 (B2) / 850 (B6)
Physical Description	
Weight	1.36 kg
Dimensions	Refer to Mechanical Drawings for Dimensions.
Chassis Type	PC-ABS
Environment	
Operating Temperature ¹	0° C to +70° C
Storage Temperature	-40° C to +85° C
Humidity	20%-90% RH, non-condensing
Power Requirements	
Operating Voltage	5Vdc, 1.4A
Certifications and Compliance	
EMC and Radio Compliance	FCC Part 15 Class B
	EN 300-220
	EN 300-440

Le routeur MTCAP 868 est un routeur qui a été conçu pour pouvoir réceptionner les données venant des différents modules lorawan dans sa zone (10 km de portée).

Il possède beaucoup de configurations différentes, on peut notamment comme n'importe quel routeur le mettre sur notre réseau, il possède un port Ethernet et des petites LED vertes qui indiquent si oui ou non le service Lorawan est actif.

4.2. Raspberry pi



Raspberry Pi est fondamentalement un ordinateur. Malgré sa petite carte, son faible coût et son système d'exploitation open source Raspbian, il s'agit toujours d'un ordinateur adapté aux besoins de base en matière de programmation.

Carte MicroSD = 16 Go

OS = Raspbian (image iso à récupérer sur internet)

5. Configuration routeur multitech Lorawan (MTCAP 868)

5.1. Introduction

Pour que le routeur puisse intercepter et récupérer les trames de données, déchiffrer (besoins des identifiants du module Lorawan) envoyées depuis le M5Stack Core 2, il faut le configurer pour faire en sorte qu'il reconnaisse notre module Lorawan. Puis par la suite, rediriger les trames contenant les données en MQTT sur l'ordinateur fixe (administrateur).

Le principe de fonctionnement est le suivant : le routeur une fois allumé va être en écoute constante sur un périmètre de 10 km. Il aura la capacité de capter les trames de données émises par n'importe quel appareil équipé d'un module Lorawan mais ne pourra pas les déchiffrer puisqu'il n'aura pas leurs identifiants.

5.2. Configuration des paramètres réseau du routeur

On part du principe que le pc fixe (administrateur) sera sur un réseau local, par conséquent, pour que le routeur puisse transférer les paquets en MQTT, on doit le mettre sur le même réseau que le pc fixe (administrateur).

Cette démarche est nécessaire car pour le transfert en MQTT, on verra après qu'on doit rentrer une adresse IP qui correspond à l'appareil qui recevra les données du routeur. Par conséquent, afin que le routeur puisse rediriger les données sur cette adresse IP, il doit la connaître et donc être sur le même réseau.

Configuration routeur :

- a) **SETUP-> DHCP Configuration**
- b) Cocher : **enable**
- c) **SETUP -> Network Interface**
- d) Cliquer sur la ligne de **br0** avec le crayon
- e) **IPV4 Setting -> Mode = static && IP_disponible** sur le réseau local voulu
- f) Le routeur sera accessible depuis le réseau local en question en tapant son IP : **IP_disponible** et pourra donc transférer les données en MQTT.

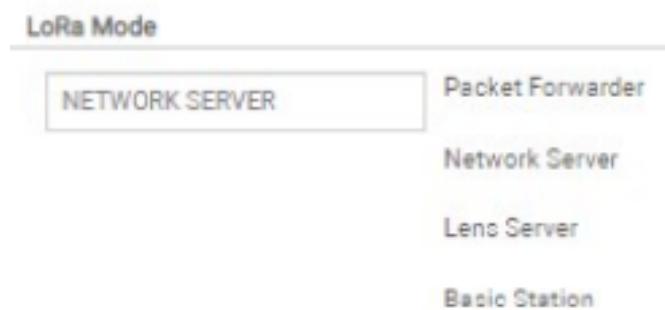
5.3. Démarrage du NETWORK SERVEUR

Afin que le routeur puisse gérer la réception des paquets, il faut activer le service Lora en mode NETWORK SERVER.

Configuration routeur :

- a) LoRaWAN -> Network interface
- b) Dans le menu LoRa Mode, il y a une case qui nous permet de choisir NETWORK SERVER

Image de ma configuration :



5.4. Ajouts des identifiants d'un module Lorawan (END-DEVICE)

- Chemin paramètres routeur = LoRaWAN -> Key Management :
- Location = Local Keys

Afin qu'il récupère seulement les trames de notre appareil et puisse les déchiffrer, il suffit juste dans les paramètres du routeur qu'on appelle également filtre, de rentrer les identifiants de notre module Lorawan en allant dans : Local End-Device Credentials -> Add new et pareil pour la partie Local Network Settings. Ces identifiants sont donnés par le constructeur lors de l'achat de l'appareil (notice d'utilisation). Si jamais ces identifiants ne sont pas disponibles, on peut pour certains les générer ou les modifier.

On doit également choisir le type de jonction (Mode de communication entre routeur et module Lorawan) qu'on souhaite utiliser (OTA ou ABP). La différence entre ces deux types de jonction est que OTA est une jonction sécurisée cryptée en AES 128 (cryptage sécurisé) car il fait ce qu'on appelle une jointure alors que ABP n'est pas crypté et est relativement plus lent. De plus, ce type de connexion n'est quasiment plus utilisé de nos jours.

- ABP : Activation By Personnalization
- OTAA : Over The Air

Il faut également choisir le type de connexion en fonction de notre localisation puisqu'en Europe, la fréquence est de 868 Mhz donc on utilise LW102-OTA-EU868

Pour le choix de la classe, il faut se référer aux spécifications de l'appareil en question mais la classe A est la classe par défaut de tous les modules Lorawan.

Signification des identifiants et paramètres pour OTA :

- Dev EUI : c'est un identifiant qui rend unique chaque objet, programmé en usine. Ce paramètre n'est théoriquement pas modifiable.
- App EUI : C'est un identifiant unique d'application qui permet de regrouper les objets. Cette adresse, sur 64 bits, permet de classer les périphériques par application. Ce paramètre est modifiable.
- App Key : Il s'agit d'un secret partagé entre le périphérique et le réseau, utilisé pour dériver les clefs de session. Ce paramètre peut être modifié.
- Device profile = Profil de l'appareil qu'on configure (LW102-OTA-EU868)
- Network Profile = Classe par default (Classe A)

Image de ma configuration :

EDIT END-DEVICE KEY

Dev EUI	70b3d59ba000659e
App EUI	70b3d59ba0000004
App Key	0F9C5AF61133BBDEF8D11345B81DFA4
Class	A
Device Profile	LW102-OTA-EU868
Network Profile	DEFAULT-CLASS-A

OK Cancel

5.5. Ajout d'un appareil pour le lier avec un module Lorawan

- Chemin paramètres routeur = **LoRaWAN -> DEVICES**

Pour finir, il est nécessaire d'ajouter un appareil dans les paramètres du routeur en allant dans **End Devices -> Add New**. Il faut rentrer le DEV EUI qu'on a rentré précédemment, le nom de notre appareil pour le reconnaître, le Device Profil qui est identique à celui rentré juste avant et enfin le Network Profile en Classe A. Cette configuration servira notamment à faire que si on reçoit des données de tel module Lorawan, on sait que ça correspond à un thermomètre par exemple.

Image de ma configuration :

ADD DEVICE

Dev EUI	70-b3-d5-9b-a0-00-65-9e
Name	EklorTest
Device Profile	LW102-OTA-EU868
Network Profile	DEFAULT-CLASS-A
Serial Number	
Product ID	
Hardware Version	
Firmware Version	
OK Cancel	

5.6. Transferts des données en MQTT

Etant donné que le routeur à la possibilité de transférer les paquets qu'il a reçu en MQTT, je vais utiliser cette méthode afin de rediriger les trames de données sur l'ordinateur administrateur.

MQTT : MQTT est un protocole standardisé reposant sur TCP/IP. Il est particulièrement utilisé pour transporter des données des objets connectés sur le cloud. Pour communiquer avec MQTT, les objets connectés utilisent un broker, c'est-à-dire un programme en charge de la réception des informations publiées afin de les transmettre aux clients abonnés. Le broker joue un rôle de relais. Il existe plusieurs types de brokers : ActiveMQ, JoramMQ, Mosquitto ou encore, RabbitMQ.

Paramètres routeur à modifier :

- a) Aller dans les paramètres du routeur
- b) Aller dans **LoRaWAN** puis **Network Setting**
- c) Aller dans Payload Broker
- d) Modifier ces champs :
 - o Username = ne rien mettre (sauf si on veut un identifiant)
 - o Password = ne rien mettre (sauf si on veut un mot de passe)
 - o Port = 1883 (Port par default)
 - o Hostname = IP de l'appareil ou l'on veut rediriger les paquets du routeur

Image de ma configuration :

The screenshot shows the 'Network Server Logging' and 'Payload Broker' sections of the configuration interface. In the 'Payload Broker' section, the 'Local Only' checkbox is unchecked, 'Network Lead Time' is set to 500, and 'Enabled' is checked. The 'Upstream Port' is set to 1780 and 'App Port Up' is 1784. The 'Downstream Port' is set to 1782 and 'App Port Down' is 1786. The 'Hostname' is 127.0.0.1, 'Username' is admin, and 'Password' is admin.

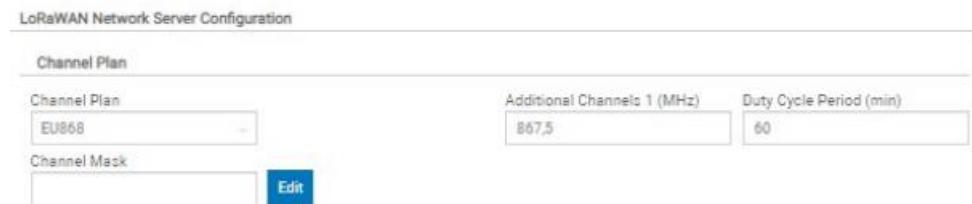
5.7. Paramètre en fonction de notre localisation

Etant donné qu'on se trouve en Europe, il est nécessaire de modifier le paramètre suivant :

Paramètre routeur à modifier :

- a) Aller dans les paramètres du routeur
- b) Aller dans **LoRaWAN** puis **Network Setting**
- c) Aller voir dans **LoRaWAN Network Serveur Configuration** et **Channel Plan**
 - Channel Plan = EU868

Image de ma configuration :



6. Configuration du serveur Mosquitto sur le PC administrateur

Afin de pouvoir réceptionner les paquets que le routeur redirige sur le pc administrateur, il est nécessaire d'avoir un serveur sur le pc administrateur qui puisse les recevoir. Pour cela, j'ai choisi d'utiliser un serveur Mosquitto car c'est simple d'utilisation et multiplateforme.

Ouvrir terminale commande sur Raspbian (Raspberry pi) :

- a) Installer mosquitto : `sudo apt-get install mosquitto`
- b) Création fichier conf afin de pouvoir écouter tous les appareils (cond.d est le dossier où l'on doit mettre les fichiers de configurations pour un mosquitto) : `sudo nano /emplacement_mosquitto/mosquitto/conf.d/nom_new_fichier_conf`
- c) Dans le fichier de conf on met : `allow_anonymous true` (on accepte les appareils anonyme...)
Listener 1883 (Port par default)
- d) Lancer serveur : `sudo service mosquitto start`
- e) Arrêter le serveur : `sudo service mosquitto stop`

7. Test du serveur Mosquitto

Grace à un client de test en python, j'ai pu tester et savoir si mon serveur mosquitto était fonctionnel avant d'envoyer de vraies données avec le routeur et le module Lorawan.

Extrait code python du client de test pour mosquitto « Client_Python_test.py » :

```
# Appel pour voir si il y a des messages publié ou non
client = mqtt.Client()

# Rentrer l'IP et le port ou le client doit écouter
client.connect("192.168.1.156", 1883, 60)

# Appel bloquant qui traite le trafic réseau, distribue les rappels et
# gère la reconnexion.
# D'autres fonctions loop*() sont disponibles qui donnent une interface threadée et un
# interface manuelle.
"""Boucle qui envoie à l'infini la data suivante avec un délai de 10 seconde"""
while True:
    your_data = "/YidAMTAuMDsxNS4wOzIwLjUhMTAxMDEwMDB+Jw==/up"
    client.publish("lora/",your_data,)
    time.sleep(10)
```

Après avoir testé, j'ai pu confirmer que le serveur mosquitto était fonctionnel car j'ai bien reçu les paquets publiés visibles sur le serveur mosquitto. De plus, mon client était sur un autre ordinateur donc cela veut dire que le serveur mosquitto accepte bien les connexions extérieures. Par conséquent, il ne devrait pas y avoir de problèmes en conditions réelles.

Exemple paquet visible sur serveur mosquitto : Received'lora/...' (44 bytes)

- ✓ Test validé

8. Test serveur Mosquitto + routeur Lorawan + module Lorawan

Pour faire le test en condition réelle, j'ai créé un serveur qui va récupérer et afficher tous les paquets de type lora qui seront publiés sur le serveur mosquitto.

Extrait code serveur python test lora « Serveur_Python_test.py » :

```
# Le rappel lorsque le client reçoit une réponse lora du serveur.
def on_connect(client, userdata, flags, rc):
    """! Fonction qu'on appelle lorsque le client reçoit une réponse lora du serveur
    @param client Identité du client
    @param userdata Données définies par l'utilisateur de tout type qui sont transmises en tant que paramètre userdata aux rappels
    @param flag objet dictionnaire utilisé pour vérifier si vous avez défini une session propre sur True ou False pour l'objet client
    @param rc code de résultat , est utilisé pour vérifier l'état de la connexion
    @return rien
    """
    print("Connected with result code " + str(rc))
    # S'abonner à on_connect() signifie que si nous perdons la connexion et
    # se reconnecter puis les abonnements seront renouvelés.
    client.subscribe("lora/#")

# Le rappel lorsqu'un message PUBLISH est reçu du serveur.
def on_message(client, userdata, msg):
    """! Fonction qu'on appelle lorsqu'un message PUBLISH est reçu du serveur
    @param client Identité du client
    @param userdata Données définies par l'utilisateur de tout type qui sont transmises en tant que paramètre userdata aux rappels
    @param msg Contient la donnée
    @return rien
    """
    topic = str(msg.topic)
    message = str(msg.payload.decode("utf-8"))
    print(topic + " " + message)
```

Après avoir lancé le serveur mosquitto et le serveur python sur le pc administrateur + lancé le programme de test de fausses données avec le module Lorawan depuis le M5Stack Core2 et configurer le routeur, on obtient le résultat attendu qui est qu'on reçoit bien des paquets lora sur le serveur mosquitto.

Exemple : serveur mosquitto en activité :

```
Received PUBLISH from 70b3d59ba0000004 (d0, q1, r0, m136, 'lora/70-b3-d5-9b-a0-00-65-9e/packet_recv', ... (250 bytes))
Sending PUBACK to 70b3d59ba0000004 (Mid: 136)
Received PUBLISH from 70b3d59ba0000004 (d0, q1, r0, m80, 'lora/70-b3-d5-9b-a0-00-65-9e/packet_recv', ... (250 bytes))
Sending PUBACK to 70b3d59ba0000004 (Mid: 137)
Received PUBLISH from 70b3d59ba0000004 (d0, q1, r0, m138, 'lora/70-b3-d5-9b-a0-00-65-9e/geolocation', ... (139 bytes))
Sending PUBACK to 70b3d59ba0000004 (Mid: 138)
Received PUBLISH from 70b3d59ba0000004 (d0, q1, r0, m81, 'lora/70-b3-d5-9b-a0-00-65-9e/geolocation', ... (139 bytes))
Received PUBLISH from 70b3d59ba0000004 (d0, q1, r0, m139, 'lora/70-b3-d5-9b-a0-00-65-9e/mac_recv', ... (124 bytes))
Sending PUBACK to 70b3d59ba0000004 (Mid: 139)
Received PUBLISH from 70b3d59ba0000004 (d0, q1, r0, m82, 'lora/70-b3-d5-9b-a0-00-65-9e/mac_recv', ... (124 bytes))
Received PUBLISH from 70b3d59ba0000004 (d0, q1, r0, m140, 'lora/70-b3-d5-9b-a0-00-65-9e/up', ... (494 bytes))
Sending PUBACK to 70b3d59ba0000004 (Mid: 140)
Received PUBLISH from 70b3d59ba0000004 (d0, q1, r0, m83, 'lora/70-b3-d5-9b-a0-00-65-9e/up', ... (494 bytes))
Received PUBLISH from lora-app-05dbc4a3186 (d0, q1, r0, m20, 'lora/70-b3-d5-9b-a0-00-65-9e/up', ... (494 bytes))
Received PUBLISH from 70b3d59ba0000004 (d0, q1, r0, m141, 'lora/70-b3-d5-9b-a0-00-00-04/70-b3-d5-9b-a0-00-65-9e/up', ... (494 bytes))
Sending PUBACK to 70b3d59ba0000004 (Mid: 141)
Received PUBLISH from 70b3d59ba0000004 (d0, q1, r0, m142, 'lora/70-b3-d5-9b-a0-00-00-04/70-b3-d5-9b-a0-00-65-9e/5/up', ... (494 bytes))
Sending PUBACK to 70b3d59ba0000004 (Mid: 142)
```

- On peut observer ici qu'un envoi est constitué de toutes ces étapes, la donnée se trouve dans le /up.
- Format paquet contenant la donnée : lora/DEV EUI/up
- Toutes ces étapes constituent la procédure nécessaire à l'envoi d'un message de type Lorawan.

Exemple : serveur mosquitto en activité mais sans paquet :

```
Received PUBLISH from 70b3d59ba000004 (d0, q1, r0, m135, 'lora/net_keepalive', ... (0 bytes))
```

- Si aucune donnée n'est envoyée, alors le routeur envoie ce type de paquet.

Exemple : serveur python qui récupère les messages lora publiés sur le serveur mosquitto :

```
lora/70-b3-d5-9b-a0-00-65-9e/up
```

- On voit bien ici que j'ai reçu un paquet de mon module Lorawan car il possède bien l'identifiant contenu entre les 2 « / » en tant que DEV EUI.

✓ Tests validés

9. Conclusion

Pour conclure, toutes les fonctionnalités attendues sont présentes et répondent au cahier des charges car je reçois bien les paquets lora envoyés du M5Stack Core2 jusqu'au serveur mosquitto sur le pc administrateur.

ENREGISTREMENT DES DONNEES DANS LA BDD

1. Auteur

Cette partie a été rédigée et réalisée par l'étudiant Loris Benaitier.

2. Objectif

L'objectif de cette partie est de réussir à enregistrer les données reçues sur le pc administrateur dans une base données envoyée depuis le M5Stack Core2.

3. Cahier des charges

- ✓ Enregistrer les informations des bateaux dans une base de données
- ✓ Faire en sorte que le système soit simple d'utilisation et robuste

4. Matériel et logiciels Necessaires

- Aucun matériel n'est nécessaire pour cette partie.
- Pour réaliser le code en python, j'ai utilisé l'IDE Pycharm qui est spécialisé pour le langage de programmation python.
- DB Browser qui est un logiciel pour visualiser et construire ses bases de données.

5. Décodage de la trame

5.1. Récupérer le paquet qui contient la donnée

Pour récupérer le paquet qui contient nos données, il faut que le serveur récupère tous les messages avec ce format : lora/DEV EUI/up

Extrait code qui récupère ce type de paquet « Serveur_Python.py » dans la fonction « on_connect » :

```
# Abonnement aux messages du topic "general"
"""S'abonner à d'autres appareils:
lora/identifiant_lora_appareil/up,
Cet identifiant sera visible sur le serveur qui reçoit les données"""
client.subscribe("lora/70-b3-d5-9b-a0-00-65-9e/up",) # Vrai données
```

5.2. Extraire la trame donnée du paquet Lorawan

Afin d'extraire la trame du paquet, on va récupérer en premier ce qui se trouve entre « "data":" » et « " » puis le décoder car il est en base64.

Extrait code qui décode et extrait la donnée du paquet « Serveur_Python.py » dans la fonction « on_message » :

```
# Decode message base64 ->text et filtrage pour récupérer seulement la donnée
string = message
found = re.search('"data": "(.+?)"', string).group(1)
print(f"base64 = {found}")
logging.debug(f"base64 = {found}")
message_bytes = base64.b64decode(found)
print(f"text = {message_bytes}")
logging.debug(f"text = {message_bytes}")
```

Une fois que la donnée est extraite on obtient notre trame :

b'@9;1;0;9.0;48.25868;!01000101~'

5.3. Extraire les différentes informations contenues dans la trame

Extraction des 2 id et des données capteurs, puis nous allons extraire le CRC et le flag.

Extrait code qui extrait les données et le CRC « Serveur_Python.py » dans la fonction « on_message » :

```
found_2 = re.search('@(.+?)!', message_bytes_str).group(1) # Récupération des données utiles
found_3 = re.search('!(.+?)~', message_bytes_str).group(1) # Récupération du CRC
```

5.4. Traitement du CRC

Afin de vérifier que le message n'est pas corrompu, on va comparer le CRC dans la trame et le nouveau CRC générer en passant les données suivantes converties en binaire :

flag;id_course;id_bateau;donnee_1;donnee_2...;

Code qui génère le nouveau CRC « Serveur_Python.py » :

```
# Fonction utilisée pour encoder les
# données en ajoutant le reste de la division modulaire
# à la fin des données.
def encodeData(data, key):
    """! Fonction qui sera utilisée pour encoder les données en ajoutant le reste de la division modulaire à la fin des données (CRC).
    @param data Donnée sur laquelle on souhaite effectuer l'opération
    @param key Clé pour le CRC définie au préalable
    @return remainder CRC
    """
    l_key = len(key)

    # Ajoute n-1 0 à la fin de la data
    appended_data = data + '0' * (l_key - 1)
    remainder = mod2div(appended_data, key)

    return remainder
```

Code qui convertit la donnée en binaire « Serveur_Python.py » :

```
# Fonction pour une transformation binaire
def strToBinary(s):
    """! Fonction qui effectue une transformation binaire.
    @param s Chaine de caractère qu'on souhaite transformer
    @return Variable Resultat de la transformation (valeur binaire)
    """
    bin_conv = []

    for c in s:
        # Convertie chaque char en
        # valeur ASCII
        ascii_val = ord(c)

        # Conversion de la valeur ASCII en binaire
        binary_val = bin(ascii_val)
        bin_conv.append(binary_val[2:])

    return (' '.join(bin_conv))
```

5.5. Traitement du flag

Pour traiter le flag, même opération que le module « envoie données depuis le M5Stack Core 2 », il faut parcourir le flag envoyé et en fonction du modèle de correspondance des bits qui est connu du serveur python et du M5Stack Core2, on peut déterminer la correspondance des données reçues (exemple : si donnee_1 == au cap ou bien à la vitesse...).

Extrait code qui gère le flag « Serveur_Python.py » dans la fonction « traitement_donnee » :

```
# Parcours du flag pour enregistrer données en conséquence...
# Principe de l'algorithme :
"""Je prend la première valeur de ma liste 'list' contenant les données que j'ai reçu,
puis je parcours mon flag et des que je tombe sur un '1', alors en connaissant l'ordre prédefinis de mon flag et l'ordre
je peux donc savoir par correspondance que la première valeur reçue correspond à tel champs."""
compteur = 0
for key in list[3:len(list)]: # champs obligatoire : flag, id_course, id_bateau
    for key_2 in flag[compteur:len(flag)]: # je commence à parcourir mon str à partir de 'l'itération + 1' ou je me suis
        if key_2 == "1":
            if compteur == 0: # cap
                list_data_bdd[4] = key
                compteur += 1
                break
            elif compteur == 1: # vitesse
                list_data_bdd[5] = key
                compteur += 1
                break
            elif compteur == 2: # latitude
                list_data_bdd[2] = key
                compteur += 1
                break
            elif compteur == 3: # longitude
                list_data_bdd[3] = key
                compteur += 1
                break
            elif compteur == 4: # vitesse vent
                list_data_bdd[6] = key
                compteur += 1
                break
            elif compteur == 5: # direction vent
                list_data_bdd[7] = key
                compteur += 1
                break
        compteur += 1
```

6. Enregistrement dans la base de données

Pour l'enregistrement des données dans une base de données, je vais faire appel à la bibliothèque sqlite3 de python. La fonction « Ajout_Base_Donnee » dans le fichier « Serveur_Python.py » va enregistrer chaque donnée dans le champ correspondant de la table.

Extrait code enregistrement base de données « Serveur_Python.py » dans la fonction « Ajout_Base_Donnee » :

```
# Préparation commande pour l'ajout de données
Name_commande = "INSERT INTO"
Name_table = Nom_table
sql = Name_commande + " " + Name_table + " (id_course, id_bateau, latitude, longitude, cap, vitesse, vitesse_v"
val = list_data
```

7. Gestion performance du serveur

Pour faire en sorte que le serveur récupère et traite les données rapidement et efficacement, j'ai imaginé un système 'Producteur consommateur' à l'aide de la bibliothèque 'thread'.

Le principe est le suivant : Un thread va gérer la récupération des données et l'autre thread va s'occuper de traiter la donnée (extraction, décodage, enregistrement BDD).

Extrait code qui gère les threads « Serveur_Python.py » dans la fonction « main » :

```
# Créations des deux threads pour la réception et le traitement des données(Modèle 'producteur consommateur').
t1 = Thread(target=reception_trame, args=(client,))
t2 = Thread(target=reception_message, args=(client,))
```

8. Gestion erreurs du serveur

Pour faire en sorte que le serveur soit robuste, qu'en cas d'erreur ou de situation imprévue, le serveur continue de fonctionner ou affiche un message d'erreur, j'ai mis en place une gestion intelligente des erreurs en utilisant notamment le 'try except' pour toutes les situations à risques.

Je me suis également servi de la bibliothèque 'logging' pour la création d'un fichier log qui enregistrera en temps réel tout ce qui se passe sur le serveur en précisant l'heure, la date, et la ligne exacte.

Extrait code pour la création et la gestion d'un fichier log « Serveur_Python.py » dans la fonction « log_init » :

```
try:
    logging.basicConfig(level=logging.DEBUG,
                        filename="historique.log",
                        filemode="a",
                        format='%(asctime)s - %(levelname)s - %(message)s')
except Exception as e:
    print(f'log : {type(e).__name__}')
    print("Erreur inattendue, impossible de créer le fichier log...")
else:
    print("Fichier log prêt!")
```

Extrait code pour la gestion d'erreur du serveur « Serveur_Python.py » dans la fonction « main » :

```
except timeout as time: # Trop de temps écoule sans aucune action(valeurs champs réseau incorrecte...)
    print(
        "@Connexion impossible!@\nVerifier les différents paramètres dans le fichier(param.ini)...")
    logging.debug(
        "@Connexion impossible!@\nVerifier les différents paramètres dans le fichier(param.ini)...")
main()
sys.exit(0) # Fin programme à cause de mauvais paramètre de base
```

9. Test unitaire

Afin de tester le système d'enregistrement dans une BDD, j'ai envoyé de fausses données au niveau du M5 Stack Core2 puis j'ai lancé le serveur mosquitto et le serveur python sur le pc administrateur.

Protocole de mise en service du serveur python (Serveur_Python.py) :

- Mettre le serveur python (Serveur_Python.py), le créateur de fichier conf (creation_fichier_conf.py), et le script bash (Installation_module_mqtt.bash) dans le même dossier
- Le fichier bdd_embarquai.sqlite contenant la base de données peut être placé n'importe où car il suffit de rentrer le chemin du fichier dans le fichier de configuration (param.ini créée grâce au programme « creation_fichier_conf.py »). Si jamais le fichier de configuration est mauvais, le serveur python prendra les configurations par défaut : nom_bdd = bdd_embarquai.sqlite. Par conséquent, il faudra placer le fichier bdd_embarquai.sqlite dans le même dossier que le serveur python (Serveur_Python.py). Si jamais le fichier bdd_embarquai.sqlite est supprimé par erreur, il faudra créer une nouvelle base de données avec la même structure que notre diagramme de base de données présenté un peu plus haut et suivre les mêmes étapes qu'au-dessus pour la lier au serveur python.

1/ Installer module paho-mqtt :

- a) Lancer le script bash nommé : Installation_module_mqtt.bash
- b) Accordé droit execution : sudo chmod +x Installation_module_mqtt.bash
- c) Exécution du Script : ./Installation_module_mqtt.bash

2/ Lancer le créateur de fichier conf (creation_fichier_conf.py) pour rentrer l'adresse ip, le port et le nom de la base de données que le serveur python doit exploiter afin qu'il puisse se connecter et enregistrer ses données :

- a) Python3 nom_fichier.py
- b) Génération d'un fichier :param.ini
- c) Rentrer les configurations voulu dans les champs disponibles et enregistrer le fichier
- d) Si les champs sont invalides, alors le serveur utilisera ses propres configurations

3/ Lancer le serveur python :

- a) Python3 nom_fichier.py
- b) Rentrer chemin du fichier de conf (param.ini)
- c) Si les paramètres étaient valides dans le fichier conf et que ces paramètres ne permettaient pas au serveur python de se connecter, alors le serveur redemandera de rentrer une configuration valide

Extrait des messages en sortie du serveur python pour l'enregistrement des données dans une base de données « Serveur_Python.py » :

```
Données lisible en cours d'enregistrement dans la base de donnee(crc valide)...!
Opérations effectuées correctement sur la base de donnée, nom : bdd_embarquai.sqlite
base64 = YidAnjM7LTE7LTE7NC4zNDswLjAw0zQ2LjMyODMy0y0wLjQ3NDUz0zAuMDA7MS410DshMDExMTAwMTB+Jw0K
text = b"b'@63;-1;-1;4.34;0.00;46.32832;-0.47453;0.00;1.58;!01110010~'\r\n"
Données lisible en attente de vérification du crc!
message = 63;-1;-1;4.34;0.00;46.32832;-0.47453;0.00;1.58;
crc = 01110010
Données lisible en cours d'enregistrement dans la base de donnee(crc valide)...!
Opérations effectuées correctement sur la base de donnée, nom : bdd_embarquai.sqlite
```

Extrait du fichier de conf (param.ini) « creation_fichier_conf.py » :

```
1 [RESEAU]
2 port = Remplacer champ(uniquement chiffres ou nombres)
3 ip = Remplacer champ(uniquement chiffres ou nombres)
4
5 [BDD]
6 nom_bdd = Remplacer champ(uniquement chiffres ou nombres)
7
```

Extrait du fichier log (historique.log) « Serveur_Python.py » :

```
2022-05-13 21:21:25,960 - DEBUG - |||||||Nouvelles valeurs de log ci dessous|||||||
2022-05-13 21:21:48,346 - DEBUG - Argument valide!
2022-05-13 21:21:48,347 - DEBUG - -----Attention, apres s'etre connecte avec succes sur le serveur mos
-1/ Serveur mosquito déconnecté ou dysfonctionnelle
-2/ Problème réseau ou matériel au niveau du routeur et ou m5stack
-----
2022-05-13 21:21:53,360 - DEBUG - -----Connexion impossible!-----
Verifier les différents paramètres dans le fichier(param.ini)...
```

Extrait des messages en sortie du serveur python lors de la connexion du serveur python « Serveur_Python.py » :

```
INITIALISATION LANCE(En attente de connexion au broker...)
Rentrer le chemin complet du fichier de configuration
gh
Argument valide!
Récuperation configuration personnalise...
erreur : KeyError
Configuration personnalise invalide(voir fichier param.ini...)
Récuperation configuration par default...
Fichier de configuration dans les normes!
-----Attention, apres s'etre connecte avec succes sur le serveur m
-1/ Serveur mosquito déconnecte ou dysfonctionnelle
-2/ Probleme reseau ou materiel au niveau du routeur et ou m5stack
-----
ip : 127.0.0.1 port : 1883 nom_bdd : bdd_embarquai.sqlite fichier : gh
Connexion réussies au broker (code=0)
```

Extrait des données récupérées dans un fichier sqlite « DB Browser » :

4364	4364	1	1	10.0	20.0	15.0	16.5	17.5	18.5	2022-05-22 01:35:04.382121
4365	4365	1	1	10.0	20.0	15.0	16.5	17.5	18.5	2022-05-22 01:36:44.484789
4366	4366	1	1	10.0	20.0	15.0	16.5	17.5	18.5	2022-05-22 01:37:24.524684

Après avoir lancé tous les programmes nécessaires pour le test, toutes les fonctionnalités voulues et attendues sont fonctionnelles et répondent au cahier des charges car on peut voir qu'on enregistre bien des données dans une base de données.

- ✓ Test Validé

10. Intégration avec l'étudiant 3

Afin d'assurer l'intégration avec l'étudiant 3, il suffira que je rentre le nom ou le chemin du fichier contenant la base de données à l'intérieur du fichier de configuration (param.ini générer par le programme creation_fichier_conf.py). Faire attention à ce que la structure de la base de données rentrée dans le fichier de configuration soit conforme à la norme établie (voir diagramme de base de données).

11. Conclusion

Toutes les exigences et les attendus du client sont présentes et fonctionnelles. La partie enregistrement dans une base de données répond au cahier des charges.

TRAITEMENT DES DONNEES VIA UNE IHM

1. Auteur

Cette partie aura dû être rédigé par Baptiste Bossuet.

2. Situation

Baptiste Bossuet ayant quitté le BTS pendant la période du projet, cette partie ne sera pas complétée.

IHM DU M5STACK CORE2

1. Auteur

Cette partie a été rédigée et réalisée par Doriath Arthus.

2. Objectif

L'objectif est de créer une interface graphique claire et simple d'utilisation pour le M5Stack

3. Cahier des charges

- ✓ Rendre les informations des capteurs NMEA lisible par le skipper
- ✓ Le système doit être facilement configurable

4. Développement

4.1 librairie utilisée

Afin de réalisés l'interface graphique du M5Stack je me suis servi de la librairie fournie par M5Stack dans la quel on trouve tous les outils pour créer des boutons, afficher des images ou encore pour afficher du texte. Cette librairie ce base sur l'appel de méthode sur l'objet M5, instance de la classe M5Core2, qui sert a représenté le M5Stack lui-même.

J'ai choisi d'utilisé cette librairie en priorité car il s'agit de celle fournis par le constructeur et ce fus un bon choix car cette librairie m'a grandement facilité la vie.

4.2 Architecture de mon IHM

Lorsque j'ai appris à me servir de la librairie M5Stack je me suis rendu compte que les boutons fonctionnent à l'aide de callback que l'on configure à la construction du bouton ainsi je me suis rendu compte qu'il m'était possible de faire l'ihm avec de la programmation évènementielle.

La programmation évènementielle est une manière de programmé qui ce base sur un grand nombre de callback, l'idée étant de n'agir qu'en réponse d'un événement, on se débarrasse donc des boucles d'exécutions et on obtient un modèle qui est bien plus facilement

modulable. J'ai essayé au maximum d'appliquer ce principe dans la programmation de mon module IHM et je suis arrivé à un résultat plus que satisfaisant.

Mon module IHM ce présente alors sous forme d'une sorte de machine à état dont les transitions sont dictées par les différents événements écoutés à un certain instant.

4.3 Différents états

Les différents états de l'IHM sont définis dans l'énumération screens. Chaque état représente une des différentes pages qui peuvent être affichées par l'IHM, ajouter une nouvelle page est donc facile.

```
enum screens{  
    HOME,  
    CONFIG,  
    SET_ID_BATEAU,  
    SET_ID_COURSE  
};
```

L'énumération qui représente les différentes pages affichables

4.4 Affichage d'une page

Les différentes pages de l'IHM sont affichées à l'aide d'une fonction du namespace ihm qui porte un nom de la forme « void drawPageScreen() » par exemple la fonction pour dessiner la page de configuration s'appelle drawConfigScreen().

```
void drawConfigScreen(){  
    boutonHome.draw();  
    boutonIdBateau.draw();  
    boutonIdCourse.draw();  
    M5.Lcd.drawJpg(img_boat.img, img_boat.size, boutonHome.x, boutonHome.y, 0,0,0,0, JPEG_DIV_2);  
    currentScreen = screens::CONFIG;  
}
```

La fonction qui permet d'afficher le menu de configuration

On peut voir dans cette fonction que l'on appelle les méthodes draw() des différents boutons de la page. Cette méthode va à la fois afficher les boutons à l'écran mais également les activer afin qu'ils réagissent aux inputs de l'utilisateur. On peut également voir que j'utilise la méthode drawJpg afin de dessiner une image à l'écran. Cette image sert d'affichage pour le boutonHome, cela permet de définir des images cliquables. Enfin une fois que la page a été affichée on change l'état de l'IHM pour indiquer que l'on est maintenant sur la page CONFIG.

4.4 Cacher une page

Lorsque l'on souhaite faire disparaître une page, notamment pour passer d'une page à une autre, on va avoir besoin de cacher la page en cours. Pour ce faire, de la même manière que pour afficher une page, on va utiliser une fonction spécialement définie pour cela. Ici elle sera nommée « void hidePageScreen() » par exemple la fonction pour cacher la page de config que l'on a vus précédemment ce nomme hideConfigScreen() .

```
void hideConfigScreen(){  
    boutonHome.hide();  
    boutonIdBateau.hide();  
    boutonIdCourse.hide();  
    M5.Lcd.clear();  
}
```

La fonction qui cache le menu de configuration

On constate dans cette fonction que l'on appelle la méthode hide() sur chacun des boutons que l'on avait précédemment draw, cela sert à désactiver les boutons de telle manière que l'on ne puisse pas actionner un bouton de l'écran de configuration alors qu'on est sur l'écran principal par exemple. On peut également constater l'appel de la méthode clear() qui nous sert ici à nettoyer l'écran, dans ce cas-ci cela sert à effacer l'image du boutonHome.

4.5 Transition vers une autre page

Lorsque l'on souhaite pouvoir passer d'une page à une autre on va devoir définir un bouton de transition, ce bouton aura comme rôle de cacher la page actuelle puis de dessiner la nouvelle page.

```
boutonHome.addHandler([], Event& e){  
    hideConfigScreen();  
    drawHomeScreen();  
}, E_RELEASE);
```

Lambda qui sert de callback au boutonHome

Dans la capture d'écran précédente l'on peut voir que l'on appelle la méthode addHandler() du boutonHome, cela sert à lui ajouter un callback qu'il appellera lorsqu'un certains évènement est déclenché. On peut également constater que le callback que l'on ajoute au boutonHome n'est pas une fonction classique mais il s'agit ici d'une fonction anonyme, aussi appelé lambda.

A l'aide de cette capture d'écran nous pouvons voir que le callback du boutonHome appelle en premier la fonction hideConfigScreen afin de cacher l'écran qui est affiché à ce moment-là (le

bouton home n'est utilisé que dans le menu de configuration). Puis la fonction drawHomeScreen est exécuté, ce qui vas avoir comme intérêt de dessiner la nouvelle page

4.6 Ajout d'une nouvelle page

Afin d'ajouter une nouvelle page il nous suffit simplement de définir une nouvelle page dans l'énumération « screens » puis de définir une fonction drawNouvellePageScreen() et la fonction hideNouvellePageScreen() et en fin il ne nous reste plus qu'à définir des boutons qui permettent de transitionner d'une page vers notre nouvelle page, et inversement d'autres bouton qui nous permettrais de sortir de cette nouvelle page pour allez a un autre endroit, et voilà on a ajouté une nouvelle page avec succès!

5 images des différentes pages de l'IHM



Page Principale de l'IHM

Si il n'y a pas de valeur sur cette page c'est parceque pour la photo le système n'était pas connecté au bus NMEA2000



Menu de configuration de l'IHM



Page de configuration de l'ID Course



Page de Configuration de l'ID Course avec une entrée de l'utilisateur



Page de Configuration de l'ID Bateau

DOSSIER MAINTENANCE
ENVOI/RECEPTION/ENREGISTREMENT
DONNEES

1. Auteur

Cette partie a été rédigée et réalisée par l'étudiant Loris Benaitier.

2. Objectif

- ✓ Résolutions de potentiels problèmes sur le système d'envoi, réception et enregistrement

3. Debug

Par default, les messages de debugs renvoyés par python sont suffisants pour comprendre d'où vient le problème. Si on souhaite revoir en détail ce qu'il s'est passé, il suffit d'aller voir dans le fichier de log (historique.log generer par le serveur python, qui contient toutes les activités du serveur)

- Si le serveur python (Serveur_Python.py) ne se connecte pas et redemande en permanence le chemin complet du fichier de configuration (param.ini générer par creation_fichier_conf.py »), et que les sorties de debugs précédentes indiquent -> Récupération configuration personnalisée puis fichier de configuration dans les normes et connexion impossible :
 - Vérifier les champs (IP et port) dans le fichier de configuration personnalisé
- Si le serveur python (Serveur_Python.py) ne se connecte pas et redemande en permanence le chemin complet du fichier de configuration (param.ini générer par creation_fichier_conf.py »), et que les sorties de debugs précédentes indiquent -> Récupération configuration personnalisé puis fichier de configuration personnalisée invalide et connexion impossible :
 - Le port et l'IP choisi sont donc ceux par defaut donc IP = 127.0.0.1 port = 1883 donc cela veut dire que l'IP et le port par defaut sont mauvais

- S'il y a le moindre problème pour lire ou récupérer les données du fichier de configuration (param.ini « Instalation_fichier_conf.py »), le serveur python ira chercher sa configuration par défaut: IP = 127.0.0.1 et port = 1883. Et nom_bdd = bdd_embarquai.sqlite
 - Exemple : Le chemin du fichier de conf est mauvais
 - Exemple : Modification d'un champ qui ne doit pas être modifié dans le fichier conf (param.ini)
- Si le serveur python c'est connecté avec succès (connexion Broker réussi) et qu'il ne se passe rien, alors cela veut dire que soit le serveur mosquitto ne reçoit rien ou bien il s'est déconnecté entre temps.
- Si on réussit à se connecter au Serveur (Connexion Broker réussi) et qu'il n'y a pas de message de debugs qui dit : Opération effectuée correctement sur la base de données après le message suivant (Donnée lisible en cours d'enregistrement crc valide), alors cela veut dire qu'il y a un problème au niveau de la base de données
 - Vérifier nom_bdd rentrer dans fichier configuration ou bien ça peut être la structure de la base de données qui ne correspond pas à la norme établie (voir diagramme base de données)

INSTALLATION MODULES :
ENVOI/RECEPTION/
ENREGISTREMENT DONNEE

1. Auteur

Cette partie a été rédigée et réalisée par l'étudiant Loris BENAFTER.

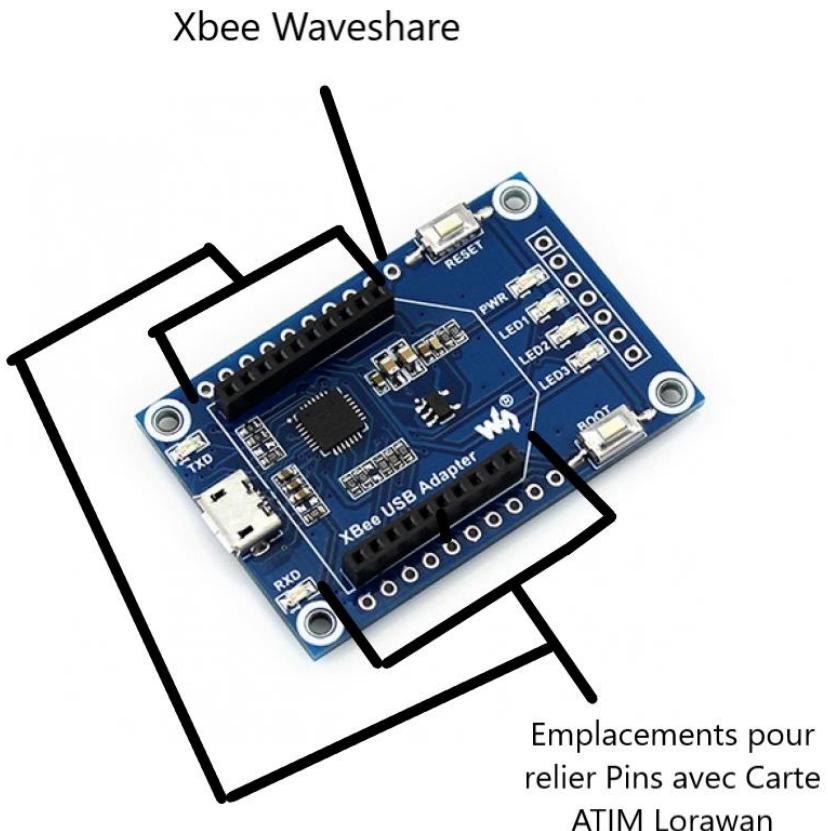
2. Objectif

Tutoriel qui va expliquer en détails les étapes nécessaires pour la mise en service du module Lorawan (Carte ATIM Lorawan + Xbee Waveshare), du routeur Lorawan, du serveur Mosquitto et du serveur python pour l'enregistrement des données.

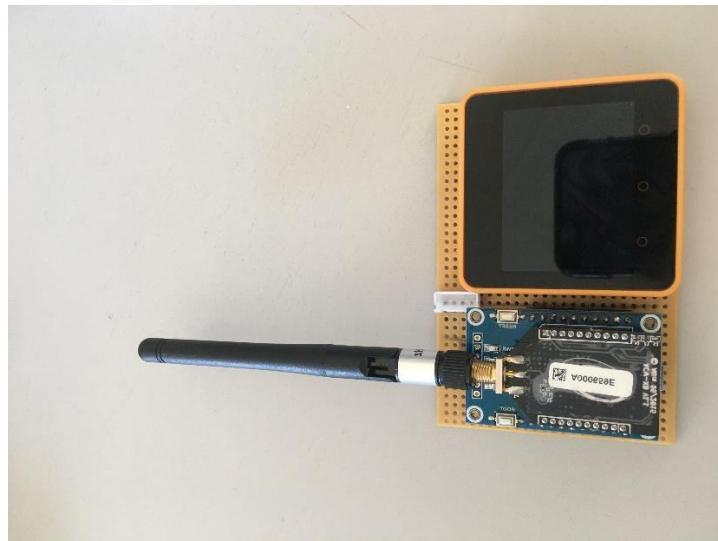
3. Carte ATIM Lorawan + Xbee Waveshare

Intégration Carte ATIM Lorawan + Xbee Waveshare sur carte électronique :

- S'assurer que la carte ATIM Lorawan est emboîtée dans les emplacements dédiés de la carte Xbee Waveshare avec l'antenne qui doit être du côté ou il n'y a pas la prise vers USB:



- Relier la carte Xbee Waveshare sur les emplacements prévus sur la carte électronique :



4. Routeur Multitech Lorawan

ATTENTION : Les configurations du routeur pour la partie Lorawan seront les configurations qui correspondent au module Lorawan utilisé pour le projet (configuration adaptée pour faire fonctionner le module Lorawan : Carte ATIM Lorawan + Xbee Waveshare). Par conséquent, si un autre module Lorawan est acheté, les configurations seront probablement différentes.

- 1) Brancher son alimentation et Appuyer sur le bouton Reset à l'arrière du routeur
- 2) Attendre quelques secondes puis relier un pc avec le routeur à l'aide d'un câble Ethernet
- 3) Tapez l'adresse suivante dans n'importe quel navigateur internet du pc relié par le câble ETHERNET : <http://192.168.2.1>
 - Si ça ne marche pas du premier coup, c'est normal, il est assez lent...
 - Dans ce cas, éteindre le routeur et le rallumer et refaire les étapes précédentes
 - Attention, si on tombe sur l'avertissement comme quoi ce n'est pas sécurisé, appuyé sur plus d'options et accepter d'aller quand même sur cette adresse.
- 4) Taper : admin dans la partie username et password puis mettre les identifiants de notre choix
- 5) Configuration Réseau en DHCP (accès au routeur sur votre réseau local) :
 - Setup -> Network Interfaces -> cliquer sur le stylo en face de la ligne br0 (BRIDGE)

Home
Save and Apply
LoRaWAN ®
Setup
Network Interfaces
Global DNS
DDNS Configuration
DHCP Configuration
SMTP Configuration
SNMP Configuration
Time Configuration

NETWORK INTERFACES CONFIGURATION ?

Name	Direction	Type	IP Mode	IP Address	Bridge	Options
eth0	LAN	ETHER	--	--	br0	
br0	LAN IPv4	BRIDGE	Static	192.168.1.222/24	br0	

CLIQUER ICI

- Mode = Static && IP == IP disponible sur votre réseau local (Cet ip servira à accéder au routeur après la fin de la configuration Réseau) && Mask = Mask correspondant à la classe de son IP (A/B/C) && Submit (enregistrer)

NETWORK INTERFACE CONFIGURATION - BR0 ?

Direction	LAN
<input type="checkbox"/> Enable IPv6 Support	
IPv4 Settings	
Mode	Static
IP Address	192.168.1.222
Mask	255.255.255.0
Gateway	
Primary DNS Server	
Secondary DNS Server	
Submit	Cancel

- DHCP CONFIGURATION : Cliquer sur stylo qui se trouve dans le sous menu « IPv4 DHCP Servers »

DHCP SERVERS AND DHCPV6/RA CONFIGURATION ?

CLIQUER ICI

IPv4 DHCP Servers						Add IPv4 DHCP Server
Status	Interface	Gateway	Domain	Lease Range Start	Lease Range End	Options
Enabled	br0	192.168.1.1		192.168.1.100	192.168.1.254	

DHCPv6 and Router Advertisement					Add DHCPv6/RA
Status	Interface	RA Mode	Lease Time	Options	
Enabled	br0	STATELESS	01-00-00		

- Cocher Enabled puis submit puis Save and Apply (Attendre que Save and Apply soit en rouge avant d'appuyer dessus...)

DHCP CONFIGURATION 

DHCP	
<input checked="" type="checkbox"/> Enabled	
Interface	Subnet
br0	192.168.1.0
Gateway	Mask
192.168.1.1	255.255.255.0
Domain	Lease time (dd-hh-mm)
	01-00-00
Lease Range Start	Lease Range End
192.168.1.100	192.168.1.254
<input type="button" value="Submit"/> <input type="button" value="Cancel"/>	

6) Configuration LoraWAN pour se connecter avec le module : Carte ATIM LoRaWan + Xbee Waveshare

- Maintenant que le routeur est disponible sur votre réseau, vous pourrez le configurer de n'importe quel ordinateur. Par conséquent, taper l'ip donnée au routeur dans la rubrique « Network Interfaces » dans un navigateur de recherche et se connecter à l'aide des identifiants définis au préalable.
- Une fois qu'on est sur la page de config du routeur : LoRaWAN-> Network Setting
 - LoRa Mode = NETWORK SERVER
 - Channel Plan = EU868 (Europe)
 - Server Ports-> Username = laisser vide, Password = Laisser vide, Hostname = IP de l'appareil ou l'on veut que les paquets reçus par le routeur soient redirigés (PC Administrateur), Port = 1883 par default, Enabled = Cochez la case && faire un submit

Server Ports = Envoyer les données en MQTT sur un autre appareil

Home

Save and Apply

LoRaWAN ®

Network Settings

- Key Management
- Gateways
- Devices
- Device Groups
- Profiles
- Packets
- Downlink Queue
- Operations

Setup

Firewall

Tunnels

LORAWAN NETWORKING ③

LoRa Mode

NETWORK SERVER	Packet Forwarder	
Network Server	4.0.1-r32.0	RUNNING
Lens Server	2.4.12	RUNNING
Basic Station	2.4.12	DISABLED
	2.0.5-1-r2.0	DISABLED

LoRa Card Information

Gateway EUI	00-80-00-00-00-01-8C-A0
Frequency Band	868
FPGA Version	31

LoRaWAN Network Server Configuration

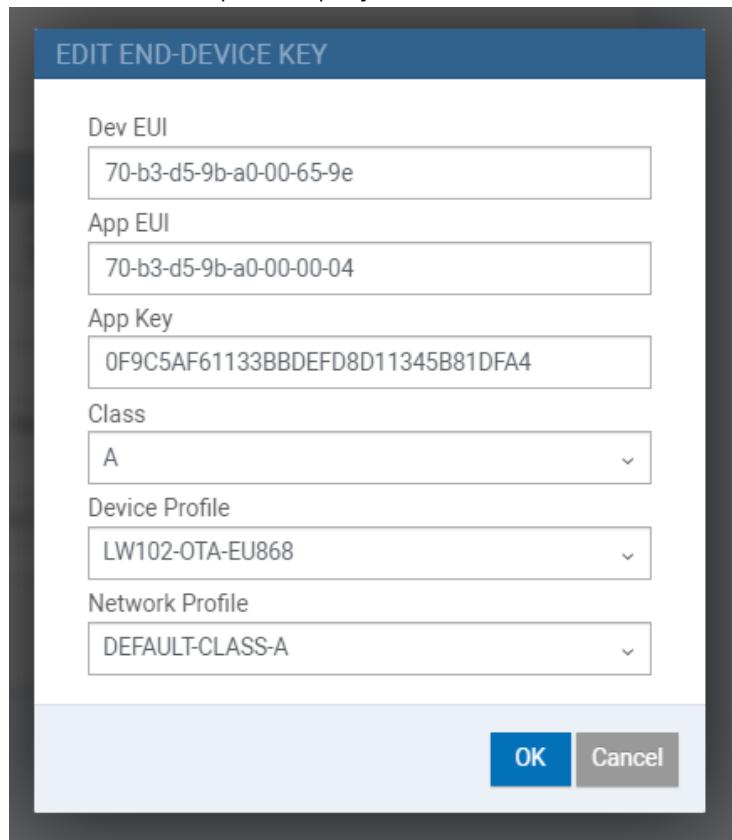
Channel Plan

Channel Plan	Additional Channels 1 (MHz)	Duty Cycle Period (min)
EU868	867,5	60

Server Ports		Hide ↑	Payload Broker		Hide ↑
<input type="checkbox"/> Local Only	Network Lead Time	500	<input checked="" type="checkbox"/> Enabled		
Upstream Port	App Port Up	1784	Hostname	192.168.1.219	Username
Downstream Port	App Port Down	1786	Port	1883	Password
1780					
1782					

- Aller dans Key Management : location = local Keys, cliquer sur le crayon dans le sous menu « Local End-Device Credentials » pour ajouter un appareil final en recopiant les identifiants suivants et faire « ok ».
(Si jamais il s'agissait d'une autre carte ATIM LoRaWaN, alors toutes les étapes avant et après seront identiques. Cependant, les valeurs rentrées dans les deux sous-menus appartenant à key management et la config dans DEVICES différerait car ces valeurs sont propres à chaque appareil. Ces valeurs sont données par le constructeur ou dans la notice du produit).

- Recopier ces identifiants qui correspondent à notre carte ATIM LoRaWaN utilisé pour le projet :



- Dans le sous-menu « Local Network setting » Recopier les différents identifiants et paramètres affichés sur ce second screen (Réécrire à l'identique car ces identifiants correspondent à notre carte ATIM LoRaWaN utilisée pour le projet) et cocher la case Enabled puis faire Submit

Join Server

Device EUI	App EUI	App Key	Class	Device Profile	Network Profile	Options
a0-00-65-9e	a0-00-00-04	****5B81DFA4	A	LW102-OTA-EU868	DEFAULT-CLASS-A	

Local Network Settings

<input checked="" type="checkbox"/> Enabled	Network ID (AppEUI)	EUI
	EUI	70-b3-d5-9b-a0-00-00-04
Default Profile	Network Key (AppKey)	Key
DEFAULT-CLASS-A	Key	0F9C5AF61133BBDEFD8D113

Submit **Reset To Default**

- Aller dans Devices :

Save and Apply

LoRaWAN ®

- Network Settings
- Key Management
- Gateways
- Devices**
- Device Groups
- Profiles
- Packets
- Downlink Queue
- Operations

DEVICES

End Devices

Device EUI	Class	Name	Last Seen	Created	Options
No matching records					

Sessions

Device EUI	Dev Addr	Up FCnt	Down FCnt	Last Seen	Joined	Details
No matching records						

- Cliquer sur Add New dans le sous-menu End Devices

The screenshot shows the 'Devices' section of the LoRaWAN management interface. On the left, a sidebar lists options like Network Settings, Key Management, Gateways, and Devices (which is selected). The main area displays two tables: 'End Devices' and 'Sessions'. The 'End Devices' table has columns for Device EUI, Class, Name, Last Seen, Created, and Options. The 'Sessions' table has columns for Device EUI, Dev Addr, Up FCnt, Down FCnt, Last Seen, Joined, and Details. Both tables show 'No matching records'. At the top right of the 'End Devices' table, there is an 'Add New' button. A large red arrow points from the text 'CLIQUEZ ICI' (Click here) to this 'Add New' button.

- Rentrer la configuration suivante (Rentrer les mêmes valeurs pour faire fonctionner la carte ATIM Lorawan du projet), puis faire « ok » et « submit » :

The screenshot shows the 'ADD DEVICE' dialog box. It includes the following fields:

- Dev EUI: 70-b3-d5-9b-a0-00-65-9e
- Name: EklorTest
- Device Profile: LW102-OTA-EU868
- Network Profile: DEFAULT-CLASS-A
- Serial Number: (empty)
- Product ID: (empty)
- Hardware Version: (empty)
- Firmware Version: (empty)

At the bottom are 'OK' and 'Cancel' buttons.

- Attendre que le bouton Save and Apply soit en rouge et appuyer pour sauvegarder la config et l'appliquer
- 7) Le routeur est maintenant configuré : il peut donc envoyer les paquets qu'ils reçoivent directement sur l'ordinateur ciblé en mqtt et il est capable de récupérer les messages envoyés par notre module Lorawan (carte ATIM Lorawan (C'est elle qui fournit les identifiants rentrés précédemment) + Xbee Waveshare).

Module Lorawan = Carte ATIM Lorawan + Xbee Waveshare car la Carte ATIM Lorawan ne fonctionne pas sans le Xbee Waveshare.

Carte ATIM Lorawan = C'est cette carte qui nous donne les identifiants qu'on a rentrés précédemment.

5. Serveur Mosquitto

Etant donné que le pc administrateur sera un Raspberry pi, les commandes seront adaptées pour Raspbian.

1 / Ouvrir un terminal sur Raspbian sur le pc administrateur

2 / Installer mosquitto : sudo apt-get install mosquitto

3/ Création fichier configuration personnalisée : sudo nano

/localisation_de_mosquitto/mosquitto/conf.d/nom_new_fichier_conf

4/ Remplir le nouveau fichier de configuration : allow_anonymous true

listener 1883

Explications :

- Afin que le serveur mosquitto puisse exécuter un fichier de configuration personnalisée, le fichier en question doit se trouver dans le dossier « conf.d », donc on doit mettre notre fichier dedans.
- On crée un fichier de configuration personnalisée car sur les dernières versions de mosquitto, le serveur n'accepte pas les connexions des appareils qui viennent de l'extérieur.
- allow_anonymous true -> Autoriser les appareils inconnus
- listener 1883 -> Port d'écoute par défaut
- localisation_de_mosquitto -> Chemin qui doit correspondre à la localisation de mosquitto sur notre ordinateur
- nom_new_fichier_conf -> Donner un nom au nouveau fichier de configuration

6. Serveur python

L'opération se passe sur le Raspberry pi (pc administrateur) :

- a) Crée un dossier sur le pc administrateur (PC Fixe) qui contiendra le serveur python, le script bash pour l'installation du module mqtt-paho et le créateur de fichier de configuration.
- b) Déposer le serveur python, le script Bash ainsi que le créateur de fichier de configuration dans ce même dossier (Serveur_Python.py et creation_fichier_conf.py et Installation_module_mqtt.bash)
- c) Ouvrir un terminal et se déplacer dans le dossier précédemment créé qui contient les trois programmes python et Bash.
- d) Accorder les autorisations d'exécution : sudo chmod +x Installation_module_mqtt.bash
- e) Lancer le script Bash : ./Installation_module_mqtt.bash

Désormais, le serveur python est prêt à être utilisé

- Le créateur de fichier de configuration « creation_fichier_conf.py » ne doit pas être nécessairement dans le même dossier que le serveur python. S'ils ne sont pas dans le même dossier, il suffira juste de donner le chemin complet du fichier généré (param.ini) au serveur.
Exemple : Au lieu de donner « param.ini », je vais donner « ../../param.ini »

UTILISATION DES MODULES : ENVOI/RECPETION/ENREGISTREMENT

1. Auteur

Cette partie a été rédigée et réalisée par l'étudiant Loris Benaitier.

2. Objectif

Tutoriel qui va expliquer en détails comment se servir des différents modules installés : module Lorawan (Carte ATIM Lorawan + Xbee Waveshare), routeur Lorawan, serveur Mosquitto et serveur python pour l'enregistrement des données.

3. Mise en œuvre

- 1) Une fois que le routeur est configuré, il suffit de l'allumer et de le connecter au réseau local à l'aide d'un câble ETHERNET.
- 2) Allumer le Raspberry pi qui fera office de pc administrateur sur le même réseau que le routeur
- 3) Vérifier au préalable que le système NMEA2000 soit prêt et branché au M5STACK Core2

PC ADMINISTRATEUR (RASPBERRY PI) :

- o Ouvrir un nouveau terminal, Lancer le serveur Mosquitto à l'aide de la commande : sudo service mosquitto start
- o Ouvrir un nouveau terminal et se déplacer jusqu'au dossier où se trouvent les programmes python à l'aide de la commande cd /...
- o Exécuter le programme qui génère le fichier de configuration
Commande : python3 creation_fichier_conf.py
- o Remplir le fichier de configuration généré, le nom sera « param.ini » et l'enregistrer : ip= ip du broker port=Port pour une écoute globale, nom_bdd=chemin ou nom du fichier bdd selon si il se trouve dans le même dossier que le serveur python
- o Allumer le M5StackCore2 et rentrer l'id_bateau et l'id_course puis valider
- o Lancer le serveur python :
Commande : Python3 Serveur_Python.py
Une fois lancé, il va nous demander de rentrer la localisation du fichier de configuration (param.ini) pour qu'il puisse se connecter à quelque chose.

Si le fichier de configuration personnalisée n'est pas lisible (Champ effacé ou fichier cassé, ou lien de sa localisation invalide...), alors le serveur prendra ses configurations par défaut IP=127.0.0.1 Port=1883
Nom_bdd=bdd_embarquai.sqlite

Si les données par défaut ne parviennent pas à se connecter à un serveur, alors le serveur python affichera des messages de debugs pour savoir ce qu'il se passe et il nous redemandera de donner un fichier de configuration.

Si les données par défaut sont valides, alors on verra affiché « connexion au broker »

Si le fichier de configuration personnalisée est lisible et qu'il peut récupérer des données (peu importe qu'elles soient bonnes ou pas), alors il va tenter de les utiliser pour se connecter à un Broker à proximité.

Si les données personnalisées ne parviennent pas à se connecter à un serveur, alors le serveur python affichera des messages de debugs pour savoir ce qu'il se passe et il nous redemandera de donner un fichier de configuration.

Si les données personnalisées sont valides, alors on verra affiché « connexion au broker »

- o Lorsque le serveur a réussi à se connecter (connexion broker réussie)

S'il reçoit des messages de debugs disant : « ... (crc valide) » et « opération effectuée correctement sur la base de données », alors il y a bien eu enregistrement dans la base de données

S'il ne reçoit pas ces deux messages de debugs, alors il n'y a pas de données enregistrées.

s'il reçoit : « crc invalide », « données corrompues », alors cela veut dire que le message n'était pas assez lisible pour être exploité.

Si jamais il y a un problème sur le processus d'enregistrement de la base de données, des messages de debug seront affichés pour l'explication (La base de données qui a été rentrée dans le fichier .conf n'est pas le bon nom, mauvaise structure...)

S'il ne se passe rien, alors soit le serveur mosquitto ne reçoit tout simplement rien et donc le problème vient peut-être du routeur ou du module Lorawan ou même du M5Stack ou bien il est déconnecté

- Si on souhaite arrêter le système, la commande pour l'arrêt du serveur mosquitto :
Commande : sudo service mosquitto stop
- Arrêt du serveur python : ctrl c ou on ferme le programme
- Arrêt du module Lorawan : Il suffit d'éteindre le M5Stack Core2
- Le routeur peut rester allumé sans problème...

4. Debug

Etant donné que le serveur python est bien optimisé, si jamais il y a le moindre problème, il suffira de lire ce qu'il se passe dans les messages de debugs du serveur python. On peut même retrouver ces messages dans un fichier de log qui sera mis à jour en temps réel, ce fichier est généré par le serveur.

- Astuce : Si on veut être vraiment sûr qu'il y a est bien des données dans la base de données, il suffit d'installer DB Browser et d'ouvrir la base de données qui est censée enregistrer les données et voir ce qu'il s'y passe.
- Également vérifier que le routeur est bien connecté au même réseau que le pc administrateur (problèmes de câble ETHERNET...)
- Vérifier que le Module Lorawan émet bien une lumière verte régulièrement, si c'est le cas, alors ils envoient des messages. Sinon il y a peut-être un problème au niveau des branchements de la carte électronique qui relie M5Stack Core2 et Module Lorawan.
- Vérifier également que le module Lorawan a bien une lumière rouge au niveau de son antenne ce qui signifie qu'il est alimenté

Exemple de sortie de debug avec le serveur python quand l'enregistrement est effectué correctement dans la base de données :

```
Données lisible en cours d'enregistrement dans la base de donnee(crc valide)...!
Opérations effectuées correctement sur la base de donnée, nom : bdd_embarquai.sqlite
base64 = YidANjM7LTE7LTE7NC4zNDswLjAw0zQ2LjMyODMy0y0wLjQ3NDUzOzAuMDA7MS410DshMDExMTAwMTB+JwOK
text = b"b'063;-1;-1;4.34;0.00;46.32832;-0.47453;0.00;1.58;!01110010~'\r\n"
Données lisible en attente de vérification du crc!
message = 063;-1;-1;4.34;0.00;46.32832;-0.47453;0.00;1.58;
crc = 01110010
Données lisible en cours d'enregistrement dans la base de donnee(crc valide)...!
Opérations effectuées correctement sur la base de donnée, nom : bdd_embarquai.sqlite
```

Exemple de sortie de debug avec le serveur python quand la configuration personnalisée est invalide et la configuration par défaut ne fonctionne pas non plus :

```
INITIALISATION LANCE(En attente de connexion au broker...)
Rentrer le chemin complet du fichier de configuration
f
Argument valide!
Récupération configuration personnalisée...
erreur : KeyError
Configuration personnalisée invalide(voir fichier param.ini...)
Récupération configuration par défaut...
Fichier de configuration dans les normes!
-----Attention, après s'être connecté avec succès sur le serveur mosquitto, si il n'y a aucune
-1/ Serveur mosquito déconnecté ou dysfonctionnelle
-2/ Problème réseau ou matériel au niveau du routeur et ou m5stack
-----
ip : 127.0.0.1 port : 1883 nom_bdd : bdd_embarquai.sqlite fichier : f
@Connexion impossible!@
Verifier les différents paramètres dans le fichier(param.ini)...
```

Exemple de sortie de debug avec le serveur python quand la configuration personnalisée est dans les normes mais qui ne permettent pas de se connecter :

```
Fichier log prêt!
INITIALISATION LANCE(En attente de connexion au broker...)
Rentrer le chemin complet du fichier de configuration
C:\Users\Home\Documents\dev\BTS_2\projet_fin_annee\code\deployment_client\serveur_received_create_conf\param.ini
Argument valide!
Récupération configuration personnalisée...
Fichier de configuration dans les normes!
-----Attention, après s'être connecté avec succès sur le serveur mosquitto, si il n'y a aucune nouvelles
-1/ Serveur mosquito déconnecté ou dysfonctionnelle
-2/ Problème réseau ou matériel au niveau du routeur et ou m5stack
-----
ip : 192.168.1.156 port : 1883 nom_bdd : bdd_embarquai.sqlite fichier : C:\Users\Home\Documents\dev\BTS_2\projet_t
@Connexion impossible!@
Verifier les différents paramètres dans le fichier(param.ini)...
```

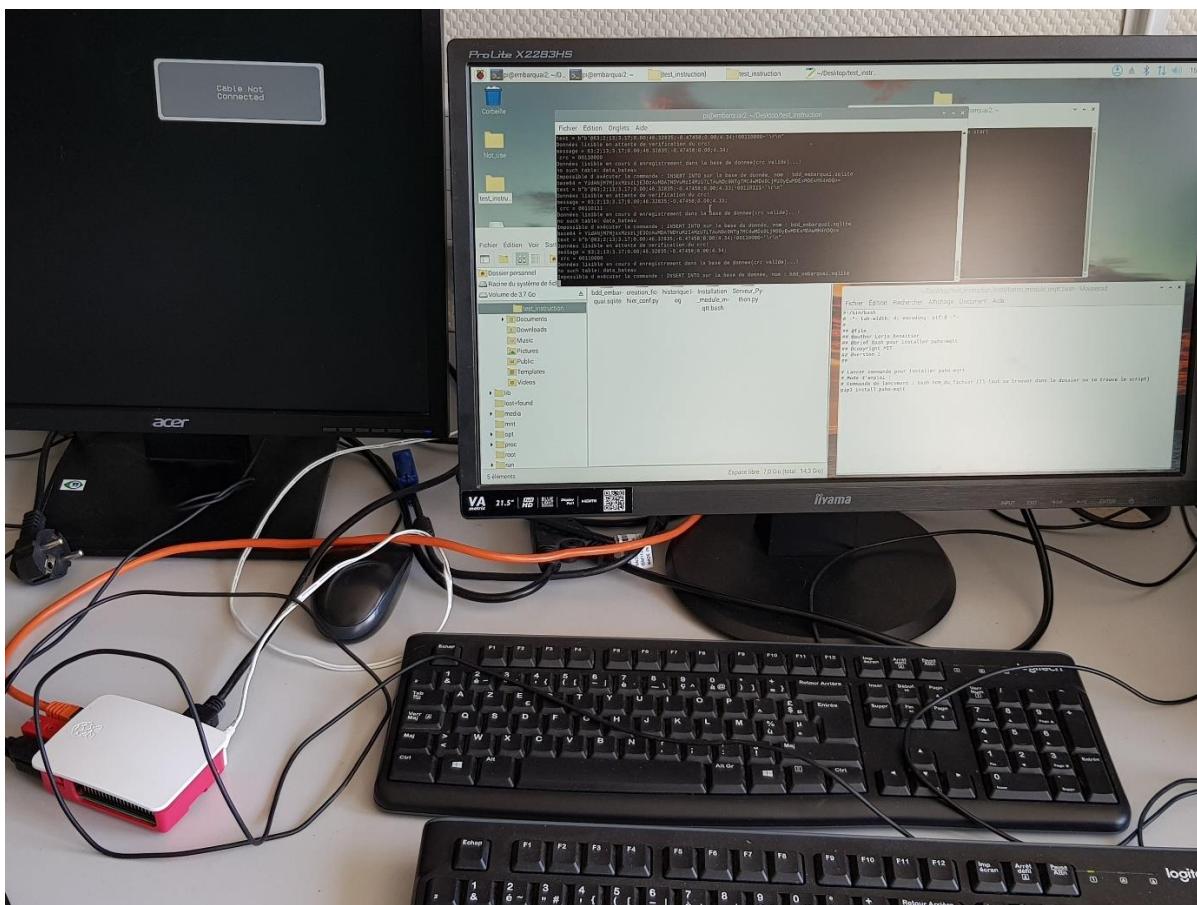
TEST FINAL DE L'ENSEMBLE DU PROJET

1. Test qui rassemble les parties de chaque étudiant

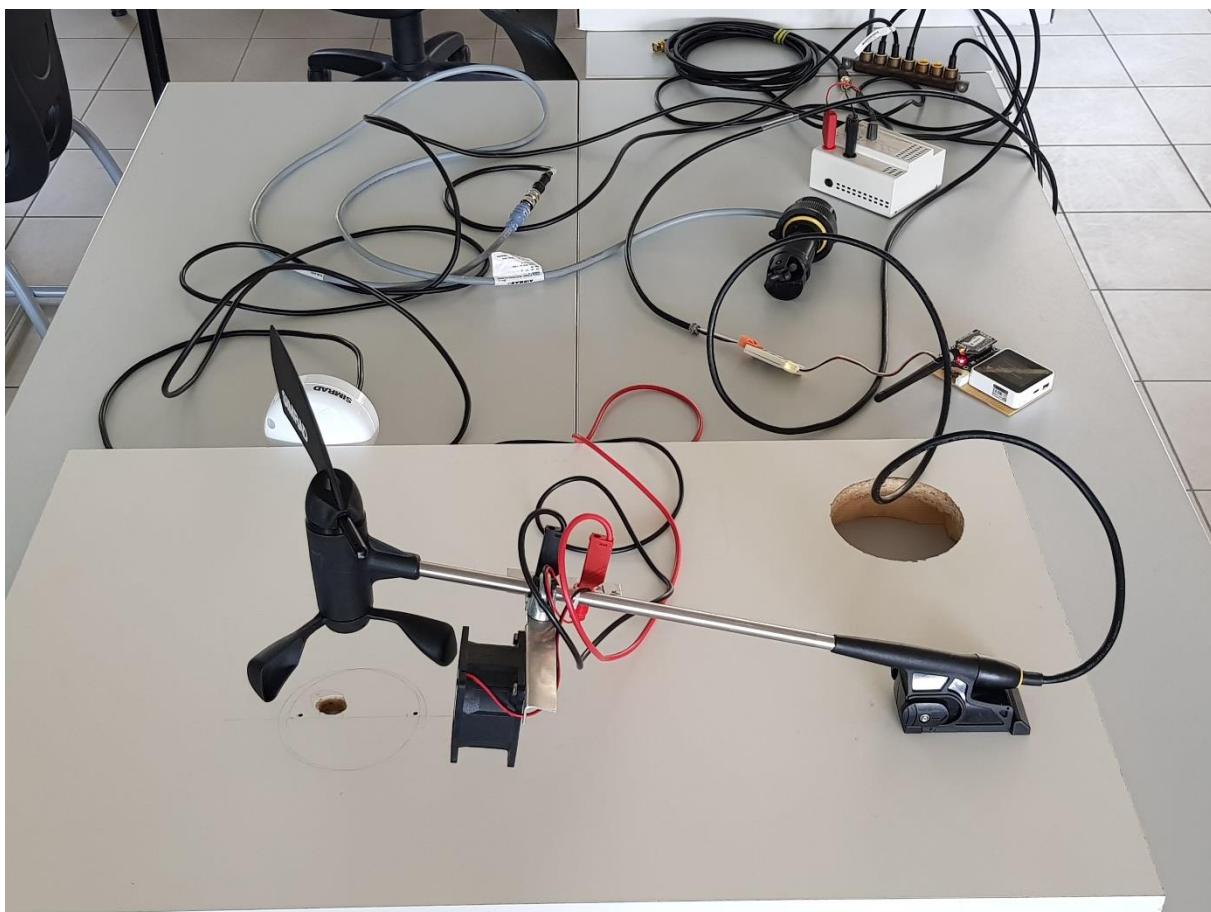
Nous avons installé tout le matériel nécessaire pour récupérer les données capteurs (Système NMEA 2000). Puis nous avons alimenter le système embarqué (Carte ATIM LoraWan + Xbee Waveshare + M5Stack Core2 le tous relié grâce à la carte électronique) via un câble USB depuis un PC.

Système final pour le test :

Coté PC Administrateur : terminal qui affiche les données reçues :



Système NMEA2000 Branché avec la partie embarqué (M5Stack Core2 + Module Lorawan) :



Partie embarquée avec l'affichage des données sur l'écran M5Stack Core2 :



On peut constater que le système complet qui rassemble le travail de toute l'équipe interagie bien ensemble puisque le système final est totalement fonctionnel et répond au cahier des charges. On arrive à envoyer des données depuis le M5Stack Core2 qui viennent des capteurs du système NMEA 2000. Les données sont bien reçues par le PC Fixe (administrateur) et enregistrer dans la base de données.

CONCLUSION

1. Synthèse et optimisations possibles

Le projet dans son ensemble s'est bien passé, mis à part le premier module Lorawan qui ne fonctionnait pas, tout le reste c'est passé comme prévu.

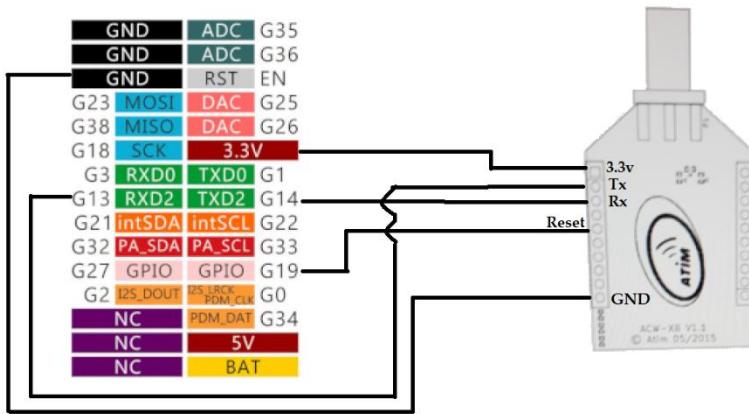
Le projet dans son ensemble est fonctionnel et répond à toutes les exigences du cahier des charges.

- ❖ Amélioration 1 : Au lieu d'envoyer le CRC complet, exemple : 101010, on peut comme pour le flag faire une transformation en décimale pour réduire le nombre d'octets envoyés dans la trame Lorawan. Ce type d'amélioration permettra notamment de diminuer le taux de bits perdus entre l'envoi et la réception des données.
- ❖ Amélioration 2 : modifier l'IHM afin de déduire quels capteurs sont connectés ou non et donc permettre d'informé l'utilisateur si il manque des capteurs
- ❖ Amélioration 3 : ce servir du Course Over Ground et du Speed Over Ground afin de remplacé le cap ou la vitesse si jamais l'on arrive pas a les récupérés pour une certaine raison.

ANNEXES

1. Schéma

1.1. M5Stack Core2 + Carte ATIM LoRaWan + Xbee Waveshare



2. Bibliographie

[1] M5STACK. « Bibliographie ». [En ligne]. Adresse URL :

<https://m5stack.com/> (page consultée en mai 2022)

[2] PlatformIO Community. « Bibliographie ». [En ligne]. Adresse URL :

<https://community.platformio.org/> (page consultée en mars 2022)

[3] Grepper. « Bibliographie ». [En ligne]. Adresse URL :

<https://www.codegrepper.com/search.php> (page consultée en mars 2022)

3. Licences

- Paho-mqtt : License EPL
- Librairie NMEA2000 : License MIT
- M5Stack Core 2 : License MIT
- Plateformio : Apache V2
- Toutes les autres librairies : MIT