École Nationale Supérieure de Techniques Avancées - Campus Brest

Rapport de stage

du 5 mai 2025 au 5 août 2025

Fabrication d'un robot sub-surface

Nom : Khaled CHATAH

Encadrant académique : Luc JAULIN
Tuteur de stage : Luc JAULIN
Laboratoire d'accueil : Lab-STICC

Rapport présenté dans le cadre d'un stage réalisé à ENSTA - Campus Brest

Année universitaire 2024–2025

Table des matières

Table des figures

Résumé

Ce rapport présente la conception et la fabrication d'un robot sub-surface tubulaire, capable d'évoluer à la fois sous l'eau (jusqu'à une profondeur maximale de 10 mètres) et à la surface. Le système dispose de deux modes de fonctionnement : un mode autonome, et un mode télécommandé, utilisant soit le Wi-Fi à la surface, soit des signaux acoustiques en immersion.

L'objectif principal est de permettre l'exécution de missions sous-marines variées, en contrôlant manuellement le robot ou en le laissant naviguer de manière autonome. Le robot est également conçu pour maintenir une profondeur stable durant la navigation, et pour capturer des photographies sous l'eau.

Ce projet a été réalisé au sein du laboratoire Lab-STICC dans le cadre d'un stage à ENSTA Campus Brest.

Abstract

This report presents the design and development of a tubular-shaped sub-surface robot capable of navigating both underwater (up to a depth of 10 meters) and on the surface. The system features two operating modes : an autonomous mode, and a remote-controlled mode using either Wi-Fi at the surface or acoustic signals underwater.

The main objective is to carry out various underwater missions, either by manually controlling the robot or letting it navigate autonomously. The robot is also designed to maintain a stable depth during navigation and to capture underwater photographs.

This project was conducted at the Lab-STICC laboratory as part of an internship at ENSTA Campus Brest.

Mots-clés / Keywords : Robot sub-surface, I^2C, Arduino, Raspberry Pi, DTMF, IMU, simulation, contrôle autonome, robotique

Chapitre 1

Modélisation et Simulation

Dans ce chapitre, nous présentons les différentes simulations effectuées dans le cadre du développement du robot sub-surface. Cela inclut le modèle d'état et la modélisation physique

1.1 Modélisation du système

Le robot sub-surface développé présente une géométrie et un comportement hydrodynamique qui s'apparentent à ceux d'un avion en vol. En effet, le robot est équipé d'ailes fixes, inclinées selon un angle de 5° dans le sens horaire. Contrairement à un avion classique, ces ailes ne sont pas orientables. Cette configuration particulière implique qu'un mouvement vers l'avant produit non seulement une poussée horizontale, mais aussi une composante verticale orientée vers le bas. Cette force contribue activement à la plongée du robot sans nécessiter de système de contrôle actif de profondeur au niveau des ailes.

Ce principe de conception permet une simplification mécanique, tout en assurant un comportement stable en immersion lors des phases de déplacement horizontal. Par ailleurs, pour la simulation, nous supposons que le poids et la poussée d'Archimède se compensent presque entièrement (flottabilité quasi neutre) ; le résultant statique vertical — éventuellement dirigé vers le haut — est donc négligeable et n'est pas pris en compte.

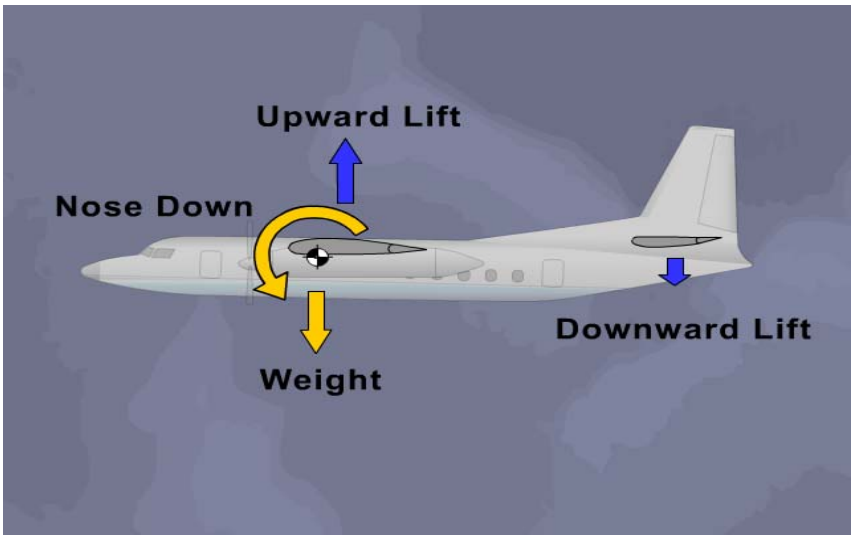

FIGURE 1.1 – Analogies aérodynamiques entre le robot sub-surface et un avion

1.2 Modèle d'état

Dans le cadre de la simulation du comportement dynamique du robot sub-surface, nous adoptons une modélisation sous forme d'équations d'état. Le système est représenté par un vecteur d'état $X \in \mathbb{R}^6$, défini comme suit :

$$X = \begin{pmatrix} x \\ y \\ z \\ v \\ \theta \\ \omega \end{pmatrix}$$

où :

— x, y, z : coordonnées locales du robot (en mètres)

— v : vitesse du robot dans le plan xOy (en m/s)

— θ : orientation du robot autour de son axe local z (en radians)

— $\omega = \dot{\theta}$: vitesse angulaire (en rad/s)

L'évolution temporelle du système est décrite par une fonction f, telle que :

$$\frac{dX}{dt} = f(X, u_1, u_2)$$

où u_1 et u_2 sont les vitesses de rotation des propulseurs gauche et droit respectivement (en rad/s).

La dynamique du robot est alors modélisée par le vecteur suivant :

$$f(X, u_1, u_2) = \begin{pmatrix} v\cos(\theta) \\ v\sin(\theta) \\ -K_{\text{th}} \cdot K \cdot (u_1 + u_2) \\ \dfrac{K_{\text{th}}(u_1 + u_2) - K_{\text{drag}} \cdot v^2}{M_r} \\ \omega \\ \dfrac{K_{\text{th}} \cdot r \cdot (u_1 - u_2)}{I_r} \end{pmatrix}$$

avec les paramètres suivants :

— K_{th} : coefficient de poussée ($\text{N} \cdot \text{s/rad}$)

— K_{drag} : coefficient de traînée ($\text{N} \cdot \text{s}^2/\text{m}^2$)

— K : constante arbitraire de simulation (sans unité)

— M_r : masse du robot (kg)

— r : distance entre les propulseurs et le centre de gravité (m)

— I_r : moment d'inertie du robot autour de son axe z ($\text{kg} \cdot \text{m}^2$)

1.3 Simulation

Dans cette section, nous présentons deux approches complémentaires de simulation mises en œuvre pour analyser et tester le comportement du robot sub-surface.

1.3.1 Simulation dans VS Code

La première simulation est effectuée dans un environnement Python sous VS Code. Elle permet d'appliquer numériquement le modèle d'état défini dans la section précédente afin de simuler le comportement du robot en réponse à des consignes u_1 et u_2 constantes ou variables.

Cette simulation inclut :

— L'intégration temporelle des équations différentielles avec Euler.

— L'observation de l'évolution des états $x, y, z, v, \theta, \omega$ dans le temps.

— La possibilité de visualiser des trajectoires 2D dans le plan xOy pour différentes valeurs d'entrée.

Ce type de simulation est essentiel pour valider le bon comportement du modèle avant de passer à une implémentation matérielle réelle.

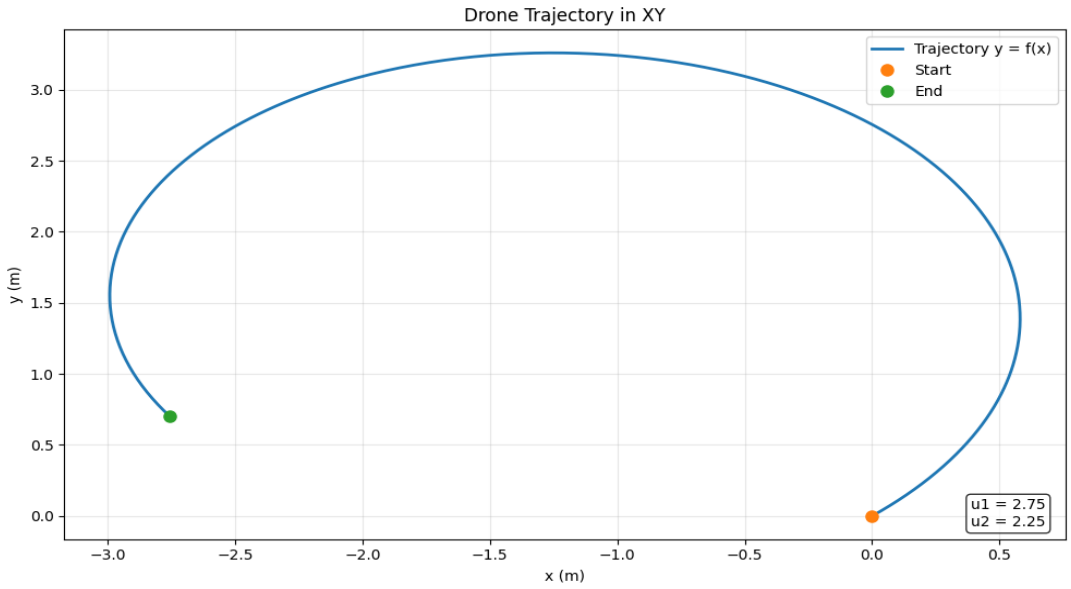

FIGURE 1.2 – Simulation d'une trajectoire circulaire dans le plan XY.

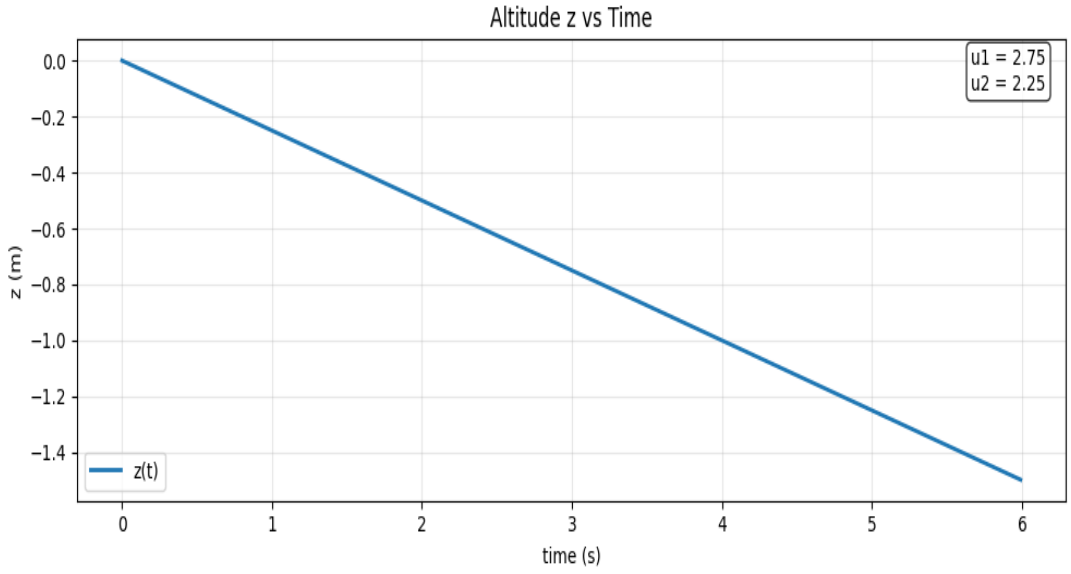

FIGURE 1.3 – Évolution de la profondeur Z pour la trajectoire circulaire.

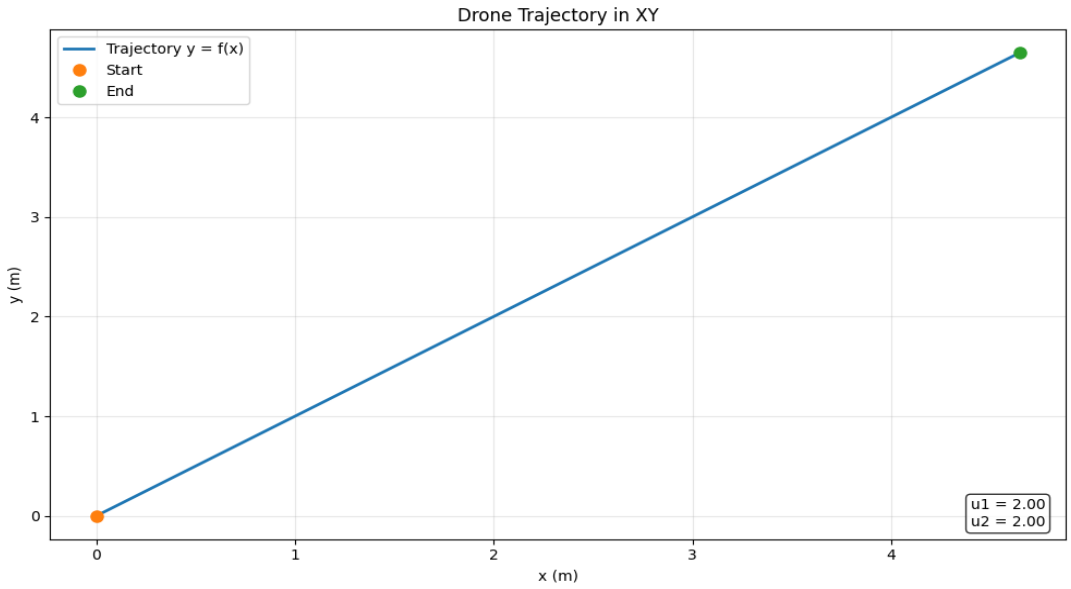

FIGURE 1.4 – Simulation d'une trajectoire linéaire dans le plan XY.

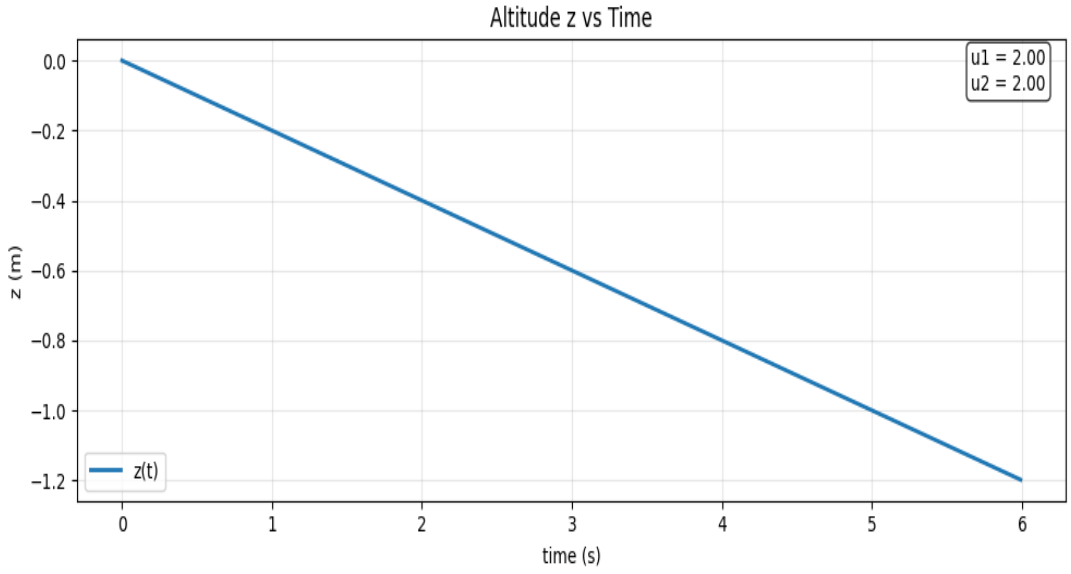

FIGURE 1.5 – Évolution de la profondeur Z pour la trajectoire linéaire.

1.3.2 Simulation dans CoppeliaSim

La deuxième simulation est réalisée dans l'environnement 3D **CoppeliaSim**, qui permet de simuler le comportement physique réel du robot dans un monde virtuel.

L'intérêt principal de cette simulation est de :

— Approcher la dynamique réelle du robot en tenant compte des interactions physiques (masse, volume, flottabilité, frottement).

— Tester différents designs mécaniques avant prototypage.

— Évaluer l'effet de paramètres tels que la poussée, la traînée ou la répartition des masses.

Cette étape permet ainsi de choisir plus judicieusement les valeurs physiques à appliquer lors de la fabrication (masse, inertie, dimensions...), afin que le prototype réel soit cohérent avec les hypothèses du modèle simulé.

Présentation de CoppeliaSim **CoppeliaSim** (anciennement appelé V-REP) est un logiciel de simulation 3D de robots, permettant de modéliser, animer et analyser le comportement de systèmes robotiques dans un environnement virtuel. Il permet d'intégrer à la fois la cinématique, la dynamique, les capteurs, les moteurs et les interactions physiques avec l'environnement. Son moteur de physique intégré permet de tester des comportements proches de la réalité.

Simulation d'un robot sub-surface cylindrique Dans le cadre de cette simulation, l'objectif était de représenter un robot sub-surface de forme cylindrique, soumis à des forces physiques proches de celles rencontrées dans le milieu réel.

Les actions menées sont les suivantes :

— Création d'un objet flottant approximativement **poussant** (quasi-neutre en flottabilité), de forme cylindrique.

— Implémentation des forces principales : **poussée d'Archimède** (buoyancy), **poids** et **force de traînée** (drag).

— Ajustement des paramètres physiques (masse, longueur, volume) pour obtenir un équilibre réaliste entre flottabilité et gravité.

Ce modèle permet de tester l'effet des forces de propulsion futures, tout en validant les hypothèses prises dans la phase de modélisation.

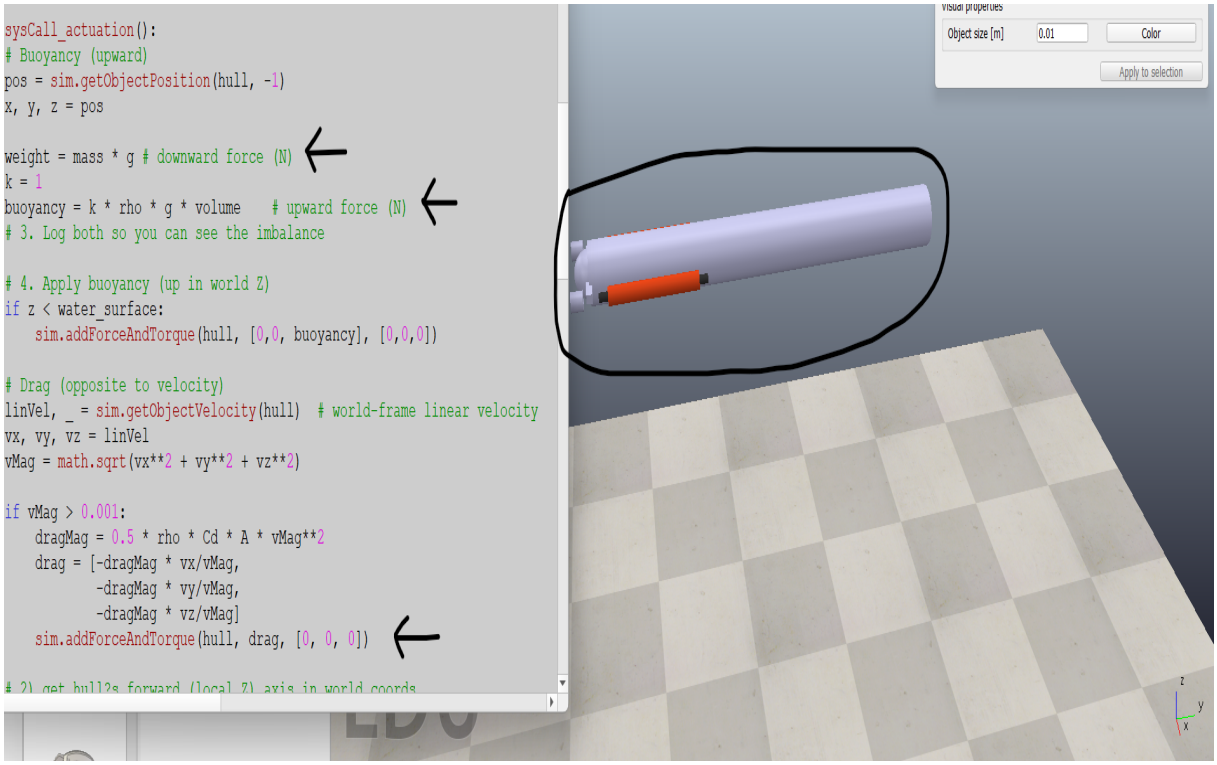

FIGURE 1.6 – Script d'implémentation des forces de flottabilité et de traînée dans CoppeliaSim.

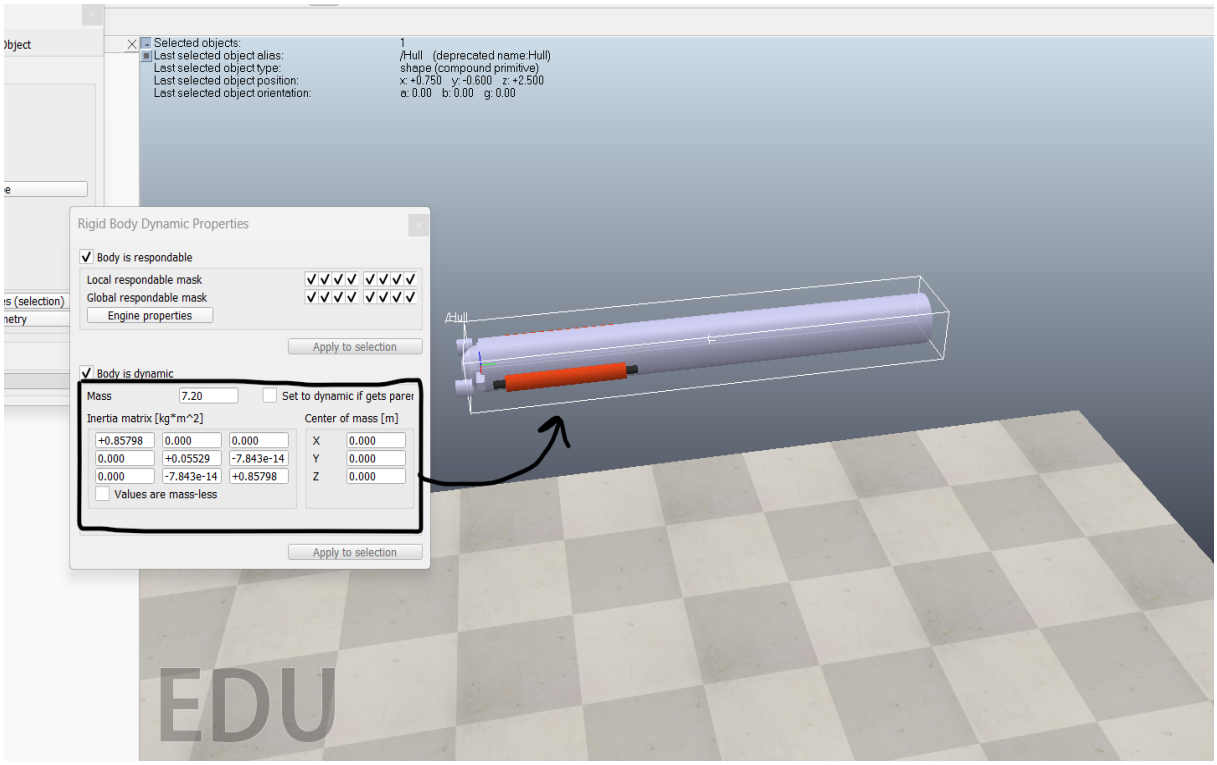

FIGURE 1.7 – Vue dynamique du robot dans l'environnement de simulation CoppeliaSim.

Chapitre 2

Prototypage

Ce chapitre décrit les différentes étapes du prototypage du robot sub-surface, en distinguant les aspects mécaniques, électroniques (hardware) et logiciels (software). Chaque sous-système a été développé indépendamment, tout en gardant à l'esprit leur intégration finale au sein du robot.

2.1 Réalisation mécanique

La partie mécanique inclut la structure externe du robot, les ailes, les systèmes d'étanchéité, les propulseurs et les supports internes pour les composants électroniques. Des matériaux résistants à la pression et à la corrosion ont été sélectionnés pour garantir le bon fonctionnement sous l'eau jusqu'à 10 mètres de profondeur.

2.1.1 Outils utilisés

Deux outils principaux ont été utilisés pour la conception et la fabrication mécanique des pièces du robot :

Autodesk Inventor Professional

Autodesk Inventor Professional est un logiciel de CAO (conception assistée par ordinateur) permettant la modélisation 3D paramétrique de pièces et d'assemblages mécaniques. Il est particulièrement adapté à la conception de systèmes mécaniques complexes, avec des fonctions avancées d'esquisse, de contraintes et de simulation.

Ultimaker Cura 5.0

Ultimaker Cura 5.0 est un logiciel open-source de découpage (slicer) développé pour préparer les fichiers d'impression 3D. Il permet de convertir les modèles 3D (fichiers `.stl`)

en instructions G-code compréhensibles par les imprimantes 3D. Il permet également de configurer les paramètres d'impression comme la température, la vitesse, la densité de remplissage, etc.

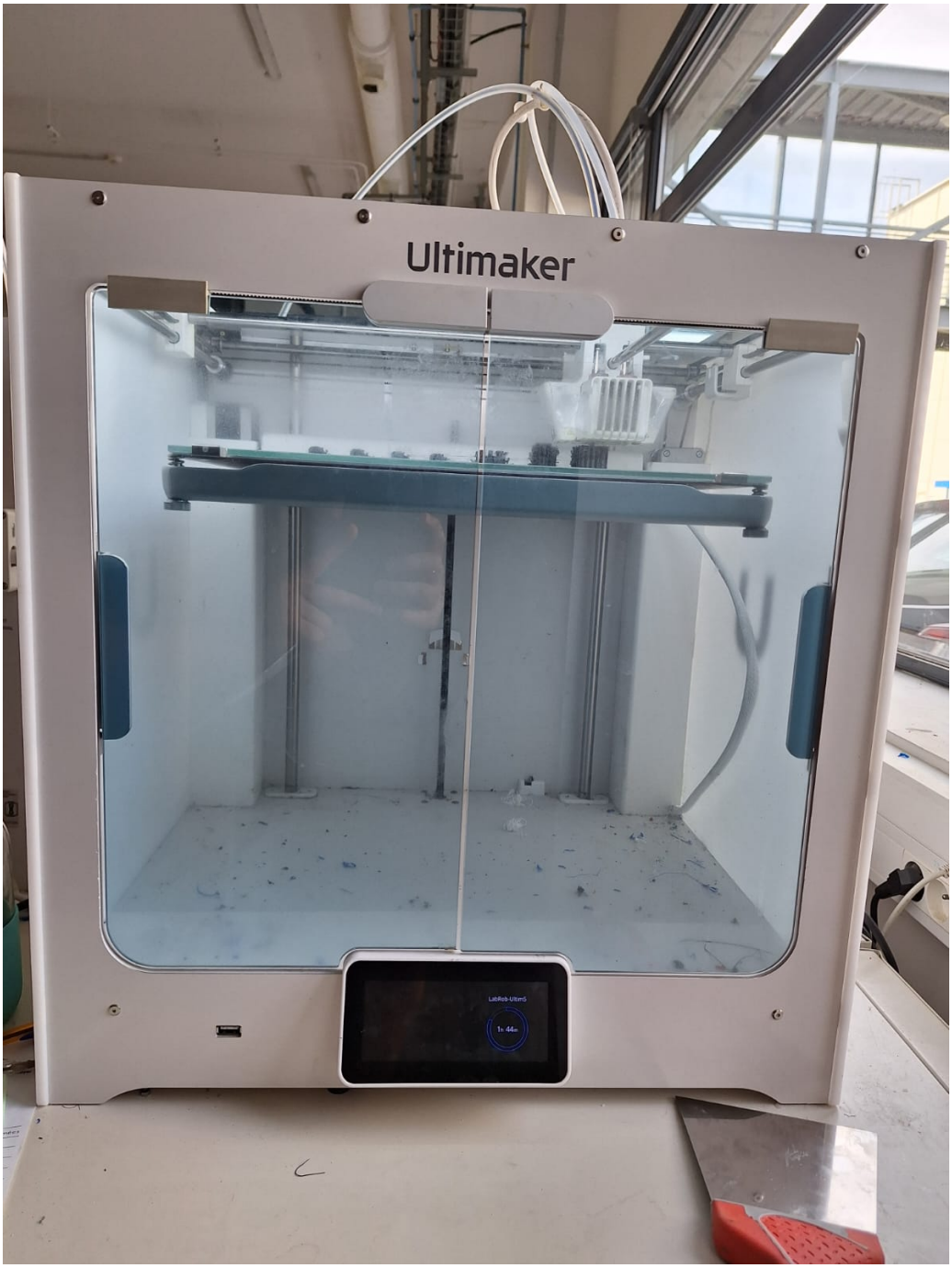

FIGURE 2.1 – Préparation d'une impression sur Ultimaker Cura 5.0.

2.1.2 Processus de fabrication

Le processus général suivi pour chaque pièce est le suivant :

1. Conception du modèle 3D dans Autodesk Inventor Pro
2. Exportation au format `.stl`
3. Importation dans Ultimaker Cura 5.0 pour la découpe (slicing)
4. Envoi du G-code à l'imprimante pour la fabrication

2.1.3 Pièces imprimées en 3D

Les pièces suivantes ont été modélisées puis imprimées :

1. Ailes inclinées (Inclined Wings) Les ailes sont inclinées selon un angle fixe afin de générer une poussée verticale vers le bas (lift inversé) lorsque le robot avance.

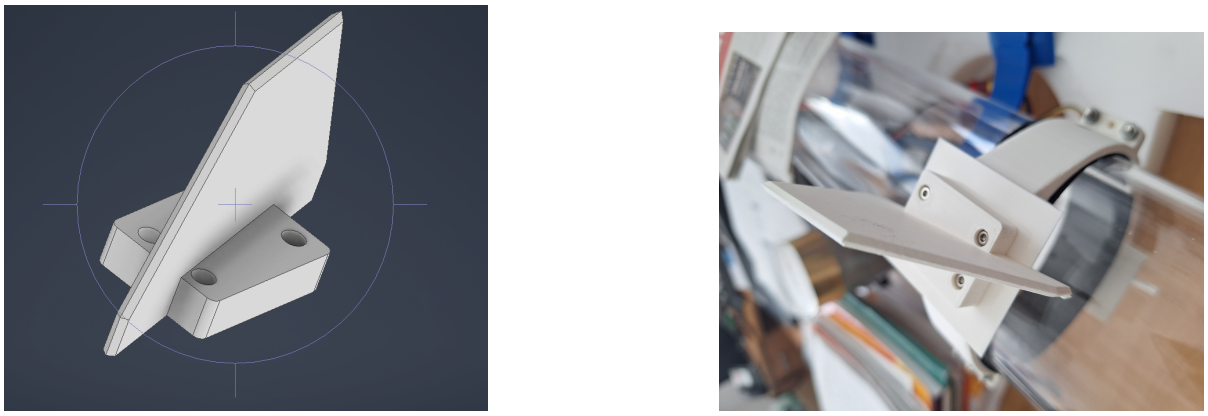

FIGURE 2.2 – Ailes inclinées — modélisation (gauche) et pièce réelle (droite).

2. Colliers de fixation (Clamps) Les colliers s'attachent autour du tube principal pour permettre la fixation de modules sur le corps du robot.

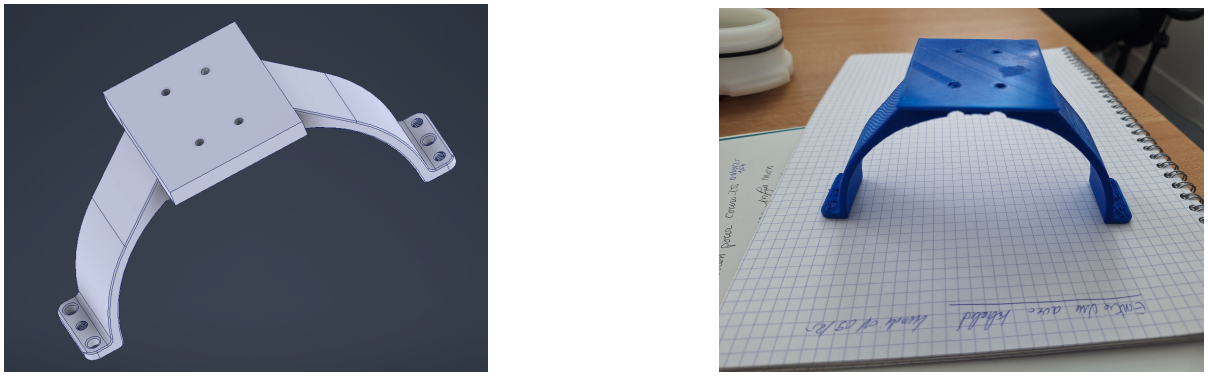

FIGURE 2.3 – Colliers de fixation — modélisation (gauche) et pièce réelle (droite).

3. Disque interne (Inside Disk) Le disque supporte la plaque en PVC qui porte les composants électroniques à l'intérieur du tube.

FIGURE 2.4 – Disque interne — modélisation (gauche) et pièce réelle (droite).

4. Bouchons d'extrémité (End Caps) Les bouchons d'extrémité, combinés à des joints toriques (O-rings), assurent l'étanchéité de la structure.

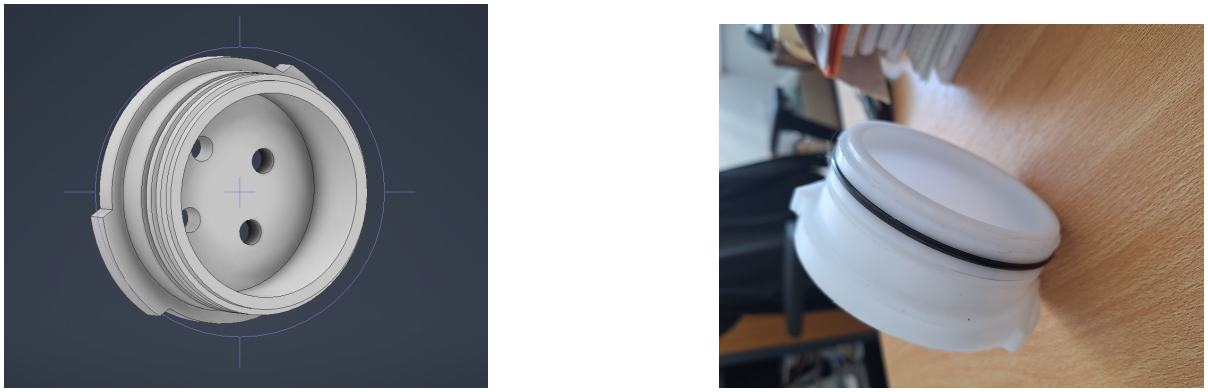

FIGURE 2.5 – Bouchons d'extrémité — modélisation (gauche) et pièce réelle (droite).

2.1.4 Composants non imprimés

Certains composants ont été achetés ou récupérés et intégrés sans impression :

- **Tube en acrylique** : corps principal du robot.
- **Plaque en PVC** : support interne pour l'électronique.
- **Visserie** : utilisée pour fixer les bouchons et faire passer les câbles (thrusters, baromètre...).

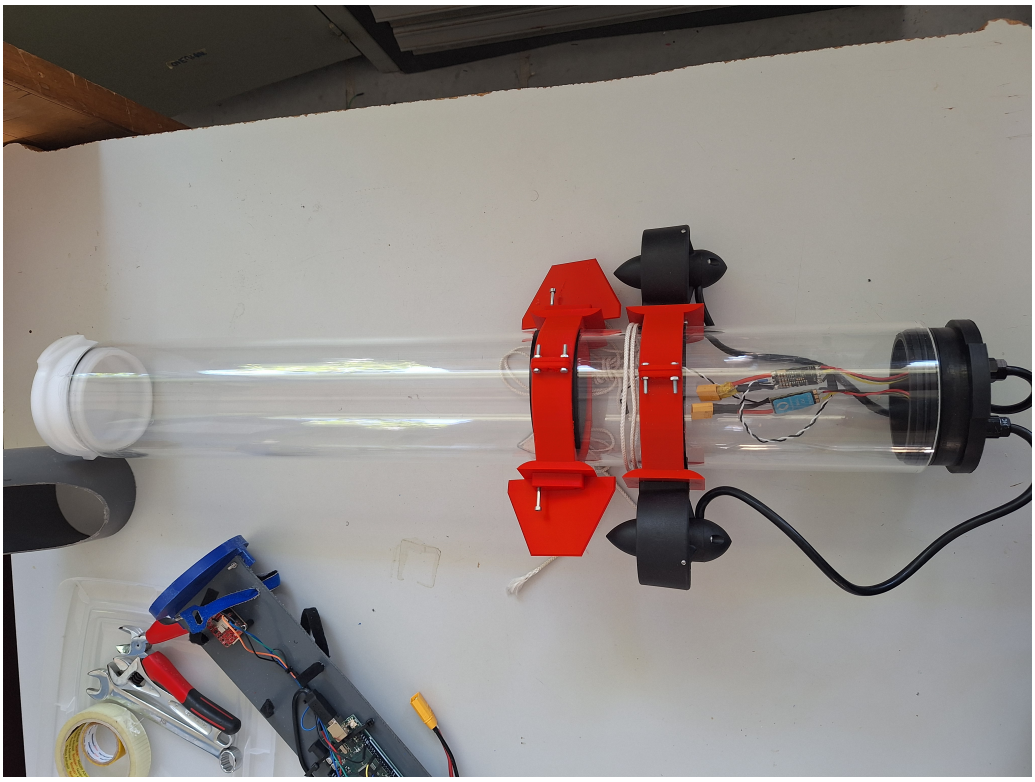

FIGURE 2.6 – Vue générale de l'assemblage mécanique du robot.

2.2 Hardware

L'architecture matérielle du robot sub-surface repose sur une sélection de composants électroniques embarqués, de systèmes de propulsion et d'éléments d'alimentation soigneusement choisis pour répondre aux contraintes environnementales et fonctionnelles du milieu aquatique.

2.2.1 Composants électroniques

Le robot intègre plusieurs dispositifs électroniques essentiels pour la collecte de données, le contrôle, et la communication :

Unités de traitement

- **Raspberry Pi 4 B** : utilisé comme unité centrale pour la gestion logicielle haut niveau, les communications Wi-Fi et le traitement de données (caméra, navigation).
- **Arduino Mega 2560** : utilisé pour le contrôle bas niveau en temps réel, notamment le pilotage des moteurs et la lecture des capteurs rapides.

Capteurs

- **OpenLog Artemis (IMU)** : fournit les données d'orientation (accéléromètre, gyroscope, magnétomètre).
- **Capteur barométrique** (Blue Robotics) : utilisé pour estimer la profondeur.
- **Capteur piézoélectrique** : détection des signaux acoustiques sous-marins.
- **Caméra OV5647** : capture d'images sous-marines en temps réel, reliée au Raspberry Pi.

2.2.2 Propulsion

- **2 Thrusters T60** : permettent le déplacement du robot sous l'eau.
- **2 ESC** : contrôleurs de vitesse électroniques, connectés à l'Arduino pour le pilotage des thrusters.

2.2.3 Alimentation

- **Batterie LiPo 3S 8000mAh** : source principale d'énergie pour l'ensemble du système.
- **TracoPower (6–18V → 5V, 6A)** : convertisseur de tension utilisé pour alimenter les composants numériques sensibles (Raspberry Pi, capteurs).
- **Power Distribution Board** : Optionnel pour la distribution des courrants vers les propulseurs et le hardware.

2.2.4 Composants divers

En complément des modules principaux, plusieurs composants auxiliaires ont été intégrés au système :

- **Carte DTMF MT8870** : utilisée pour décoder des commandes envoyées sous forme de signaux audio (Dual Tone Multi Frequency).
- **Convertisseur de niveau logique bidirectionnel 5V - 3.3V** : permet la communication entre les circuits 5V (Arduino) et 3.3V (Raspberry Pi, capteurs sensibles).
- **Résistances de tirage de 4.7 kΩ** : placées sur les lignes SDA et SCL pour assurer la fiabilité de la communication I^2C.
- **Circuit amplificateur custom pour le capteur piézoélectrique** : utilisé pour amplifier les signaux faibles (figure 3.1)

2.2.5 Schéma matériel (Hardware Diagram)

Le schéma fonctionnel du système embarqué est présenté à la Figure 2.7. Il illustre l'architecture générale des composants électroniques ainsi que les relations de communication entre eux.

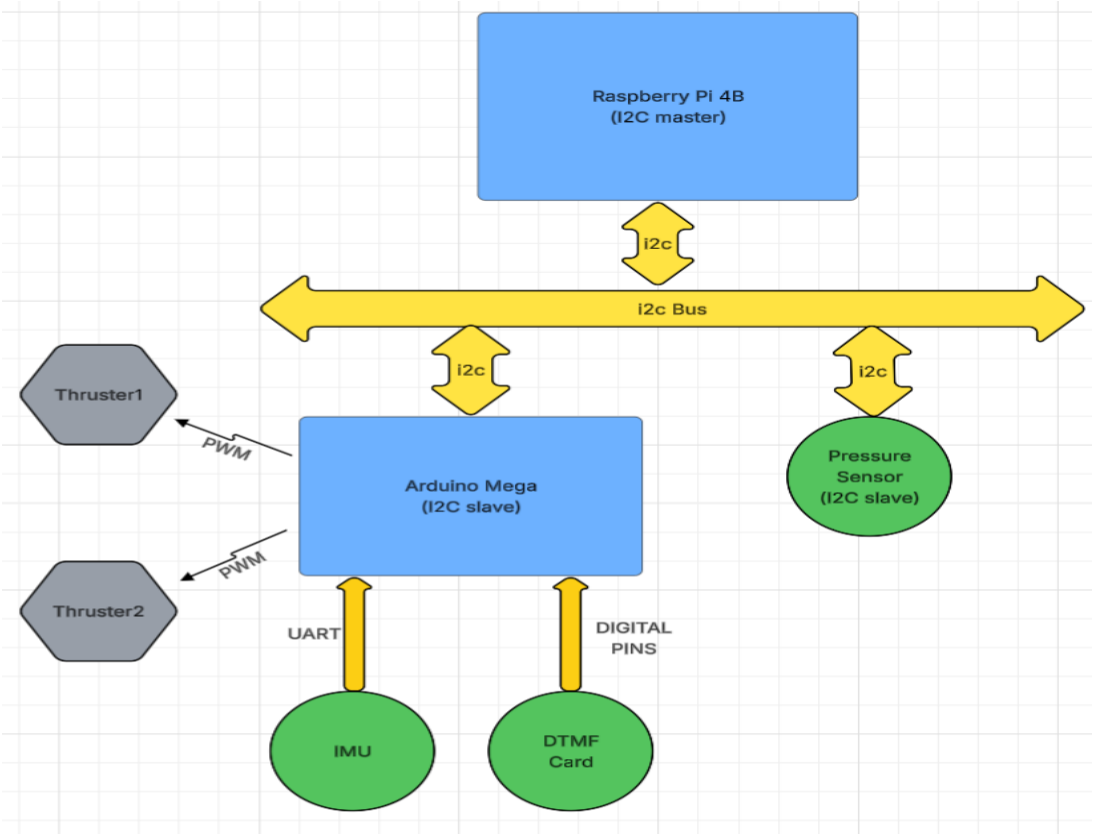

FIGURE 2.7 – Architecture matérielle simplifiée du robot sub-surface.

Rôle des composants dans l'architecture

Raspberry Pi 4 B

Le **Raspberry Pi** est l'unité principale du système. Il joue le rôle de **cerveau du robot**, hébergeant l'ensemble des algorithmes de navigation, d'asservissement et de télécommande. C'est également le **maître du bus I²C** : il attribue une adresse à chaque périphérique esclave connecté et peut envoyer ou lire des données ciblées selon cette adresse.

Arduino Mega 2560

L'**Arduino Mega** agit en tant qu'**esclave I²C**. Il a pour rôle de :

— Transmettre en continu les données de l'IMU (OpenLog Artemis) au Raspberry Pi.

— Lire les consignes de vitesse reçues du Raspberry Pi et les transformer en signaux PWM pour piloter les deux thrusters.
— Lire les données du module DTMF afin de déterminer si le mode télécommande est activé.

Capteur barométrique

Le capteur de pression (baromètre Blue Robotics) répond aux requêtes du Raspberry Pi via le bus I^2C. Ses données seront utilisées dans les fonctions futures d'asservissement en profondeur ou d'autres fonctionnalités définies par l'utilisateur.

Caméra

Bien qu'elle ne soit pas représentée sur ce schéma, une caméra OV5647 est également intégrée au système. Elle est connectée directement au Raspberry Pi via l'interface CSI.

Communication I^2C

L'ensemble des échanges entre les composants principaux s'effectue via le bus I^2C, avec le Raspberry Pi comme maître et les autres éléments comme esclaves. Le protocole de communication, incluant le format des trames et la logique de synchronisation, sera détaillé dans une section ultérieure.

Ce schéma a servi de référence tout au long de l'intégration physique du système, garantissant la cohérence entre les modules et le respect des contraintes de tension, de communication et de routage des signaux.

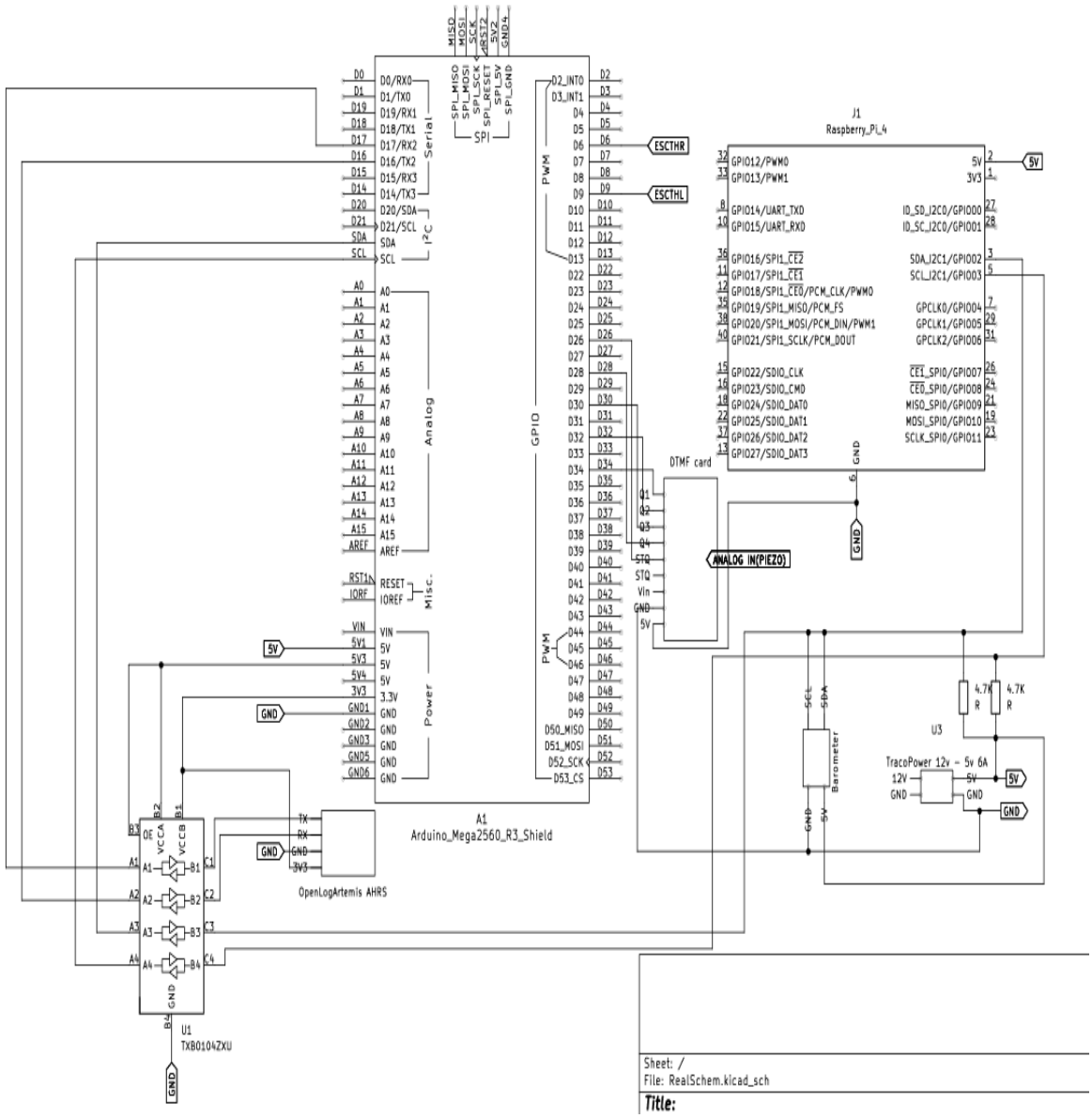

FIGURE 2.8 – Schéma complet de l'architecture électronique réelle du robot.

2.3 Software embarqué

Le logiciel du robot a été conçu pour gérer les deux modes de fonctionnement : autonome et télécommandé. Il inclut la gestion des capteurs, le contrôle des moteurs, la navigation, la communication ainsi que les algorithmes d'évitement d'obstacles et de régulation de profondeur.

2.3.1 Configuration initiale du système embarqué

Avant de pouvoir intégrer les différents modules dans le code principal, certaines étapes de configuration doivent être réalisées. C'est notamment le cas pour le module **OpenLog Artemis (OLA)**.

Chargement d'un firmware personnalisé pour l'OpenLog Artemis

Le module OpenLog Artemis n'est pas utilisé avec le firmware fourni par le fabricant (le firmware OLA par défaut), car ce dernier ne répond pas aux besoins du projet. Un **firmware personnalisé** est utilisé à la place.

Ce firmware permet :

1. D'**activer l'interface UART** du microcontrôleur embarqué, afin que les données IMU soient transmises en série.
2. De **modifier la structure des données** envoyées, afin qu'elles soient compatibles avec la bibliothèque Arduino `RazorIMU_9DOF.h`.

Les modifications suivantes doivent être effectuées dans le fichier `UserSetup.h` avant la compilation du firmware :

— Décommenter les lignes suivantes :

```
//#define HW__VERSION_CODE 16832 // SparkFun "OpenLog Artemis" version "DEV-1683
//Uart SerialLog(1, 13, 12);
//#define LOG_PORT SerialLog
```

— Commenter la ligne suivante :

```
#define LOG_PORT Serial
```

Ces modifications permettent d'activer l'interface UART matérielle du module et de rediriger les données vers la bonne sortie série.
Une fois ces lignes modifiées, le code peut être téléversé via l'IDE Arduino. Le module OLA sera alors prêt à communiquer avec l'Arduino Mega via UART, dans un format compatible avec la bibliothèque logicielle utilisée.

Configuration du Raspberry Pi 4

Pour permettre au Raspberry Pi 4 de communiquer avec les autres composants (notamment l'Arduino Mega et les capteurs), il est nécessaire d'activer et de configurer correctement l'interface I^2C.

Étapes de configuration :

1. Installer le système d'exploitation Raspberry Pi OS à l'aide de l'outil **Raspberry Pi Imager**, disponible sur Windows et Linux.
2. Activer l'interface I^2C via la commande :

   ```
   sudo raspi-config
   ```

   Dans le menu qui s'ouvre, se rendre dans `Interfaces` puis activer `I2C`.
3. Vérifier que le bus I^2C numéro 1 est activé au démarrage en ajoutant les deux lignes suivantes dans le fichier `/boot/firmware/config.txt` :

   ```
   dtparam=i2c1=on
   dtparam=i2c_arm=on
   ```

4. Redémarrer le Raspberry Pi, puis tester la détection des périphériques connectés sur le bus I^2C avec la commande :

   ```
   sudo i2cdetect -y 1
   ```

   Cette commande affiche une matrice avec les adresses détectées, confirmant la communication avec les périphériques I^2C connectés.

Organisation du programme embarqué sur l'Arduino

Le microcontrôleur Arduino joue le rôle d'esclave sur le bus I^2C. Il est responsable de :

— Lire les données de l'IMU et les transmettre sur demande.

— Recevoir et exécuter les consignes de vitesse envoyées par le Raspberry Pi.

— Piloter les moteurs via PWM.

Le fonctionnement général du programme est illustré dans les deux diagrammes suivants.

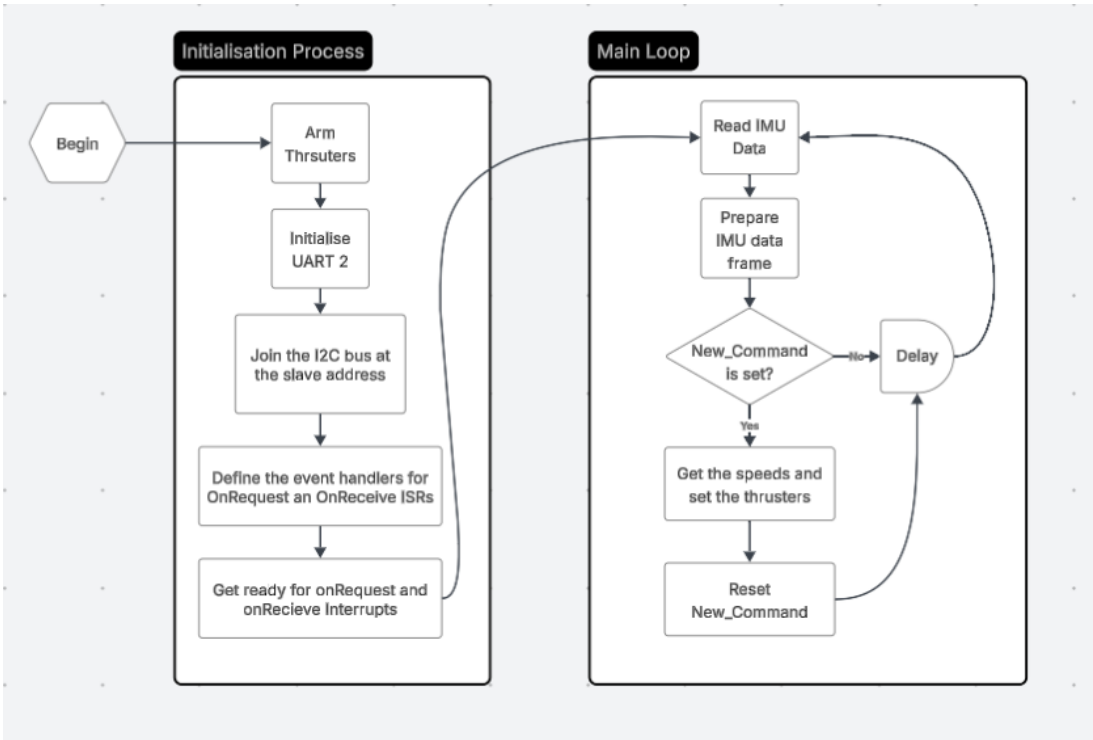

FIGURE 2.9 – Structure du programme principal exécuté sur l'Arduino

Le programme commence par :

— L'initialisation des propulseurs

— L'ouverture de la communication UART2 avec l'OpenLog Artemis

— L'enregistrement de l'Arduino sur le bus I^2C avec une adresse esclave

— La définition des fonctions d'interruptions pour la réception (‘onReceive‘) et les requêtes (‘onRequest‘)

Ensuite, la boucle principale effectue :

— La lecture des données IMU

— La mise en forme de la trame à envoyer

— Le traitement des consignes si une nouvelle commande a été reçue (vitesse)

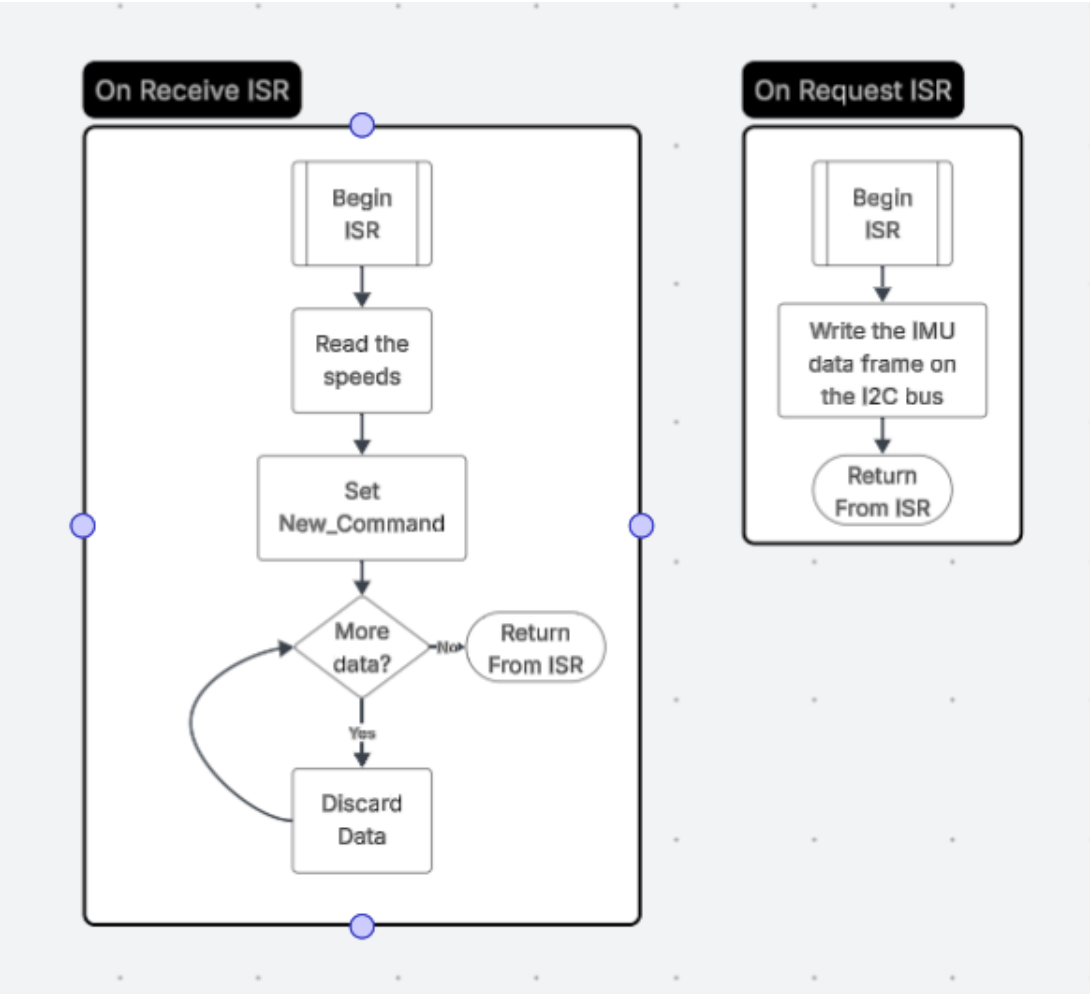

FIGURE 2.10 – Gestion des interruptions I^2C sur l'Arduino

Le traitement des interruptions est géré de manière non-bloquante via les routines suivantes :

- **onReceive** : déclenchée lors d'un envoi du maître (le Raspberry Pi). Elle lit les consignes de vitesse et les stocke.
- **onRequest** : déclenchée lors d'une lecture demandée par le maître. Elle écrit la trame IMU actuelle sur le bus I^2C.

Le système gère également les cas où plusieurs octets seraient envoyés (flush), et garantit la mise à jour des moteurs uniquement si une commande valide a été reçue.

Remarques sur les composants non implémentés

À cause de contraintes de temps liées à la durée du stage (3 mois), certains modules prévus initialement n'ont pas pu être intégrés dans la version actuelle du logiciel embarqué.

Plus précisément :

- Le **capteur barométrique** (pour la mesure de profondeur)
- Le **module DTMF** (pour le contrôle télécommandé par signal audio)

Cependant, l'**architecture matérielle** a été conçue de manière à permettre leur ajout sans modification majeure. Les connexions physiques sont déjà prévues sur les bus I^2C ou les entrées analogiques correspondantes.

Du point de vue logiciel, le fonctionnement actuel du code repose sur une fonction `readIMU()` dédiée à la récupération des données de l'IMU. Dans une version future complète, cette fonction pourra être remplacée par une fonction plus générale du type :

```
readAllSensors()
```

Celle-ci regrouperait la lecture :

- des données de l'IMU,
- des valeurs de pression issues du baromètre,
- et des commandes DTMF.

Les données seraient ensuite envoyées au Raspberry Pi selon le même protocole que celui déjà en place, garantissant ainsi une évolution fluide du système.

Organisation du programme C++ sur le Raspberry Pi

Le diagramme ci-dessous présente l'organisation fonctionnelle du code embarqué en C++ exécuté sur le Raspberry Pi. Il est structuré en trois parties : initialisation du système, boucle principale de traitement et communication, et gestion d'arrêt sécurisé via le signal SIGINT.

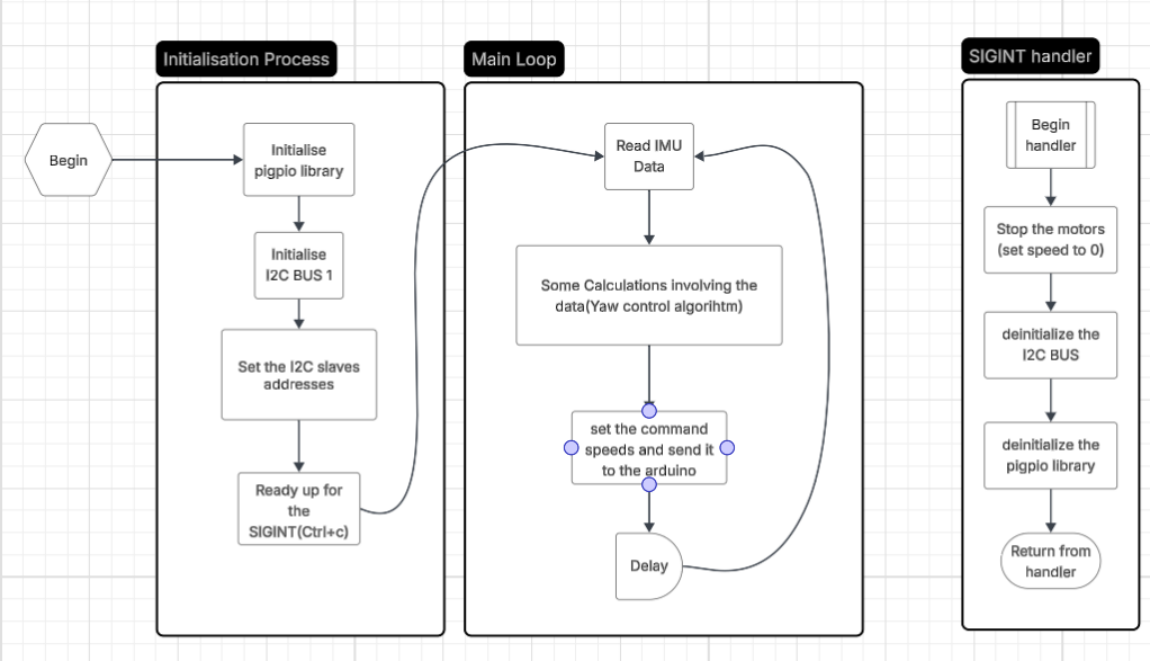

FIGURE 2.11 – Structure logique du programme C++ exécuté sur le Raspberry Pi

Initialisation du système :

- Initialisation de la bibliothèque `pigpio`, utilisée pour la gestion des GPIO et du bus I^2C.
- Activation du bus I^2C numéro 1.
- Définition des adresses des périphériques I^2C (comme l'Arduino).
- Mise en place d'un gestionnaire du signal SIGINT pour gérer proprement l'arrêt du programme.

Boucle principale :

- Lecture des données IMU en provenance de l'Arduino.
- Traitement logiciel (ex. : algorithme de contrôle du cap).
- Envoi des consignes de vitesse à l'Arduino.
- Pause avant la prochaine itération.

SIGINT Handler (interruption via Ctrl+C) :

- Mise à zéro des moteurs (consigne nulle).
- Libération du bus I^2C et arrêt de la bibliothèque `pigpio`.
- Sortie propre du programme.

Chapitre 3

Tests réalisés et validation

Ce chapitre présente les différents tests effectués pendant la période de stage. Chaque composant critique du système a été validé individuellement ou en condition de laboratoire. L'objectif était de s'assurer de la fiabilité des sous-systèmes avant l'intégration complète.

3.1 Tests de l'IMU

Un code dédié a été écrit sur l'Arduino Mega pour vérifier la réception correcte des données provenant de l'OpenLog Artemis via UART. Les résultats ont montré que l'IMU fonctionne correctement, avec une vitesse de lecture très élevée, ce qui permet un *temps de scrutation* très faible lors de l'acquisition des données. Cela est particulièrement favorable pour l'exécution simultanée d'autres tâches comme l'asservissement moteur ou la lecture d'autres capteurs.

3.2 Tests des propulseurs (T60)

Deux programmes de test ont été réalisés pour évaluer le contrôle des propulseurs via signal PWM :

- Le premier permettait d'**armer les ESCs** et de maintenir une vitesse constante.
- Le second envoyait des **vitesses variables** pour observer le comportement en cas de changements rapides de commande (amplitude et direction).

Il a été observé que le signal PWM neutre est situé autour de **1500 µs**, bien que cette valeur puisse varier selon la performance de l'Arduino. Pour une tension d'entrée de 12.5 V :

- Entre 1450 et 1550 µs (soit ±50 µs autour du neutre), les moteurs restent **à l'arrêt** (saturation à 0 rpm).

— À partir de ±480 µs autour du neutre, les moteurs atteignent leur **vitesse maximale** (saturation à pleine puissance).

Les fonctions `readIMUdata()` et `speedSet()` ont ensuite été intégrées dans le programme principal de l'Arduino.

3.3 Tests du module DTMF

Deux tests ont été réalisés pour valider l'utilisation du module DTMF (MT8870) :

1. Connexion directe entre le module DTMF (entrée audio depuis un PC) et les entrées numériques de l'Arduino. Les signaux DTMF ont été parfaitement décodés.
2. Test via transducteur piézoélectrique : le signal provenant du piézo était trop faible pour être détecté. Pour pallier cela, un **circuit amplificateur et filtre personnalisé** a été conçu et implémenté.

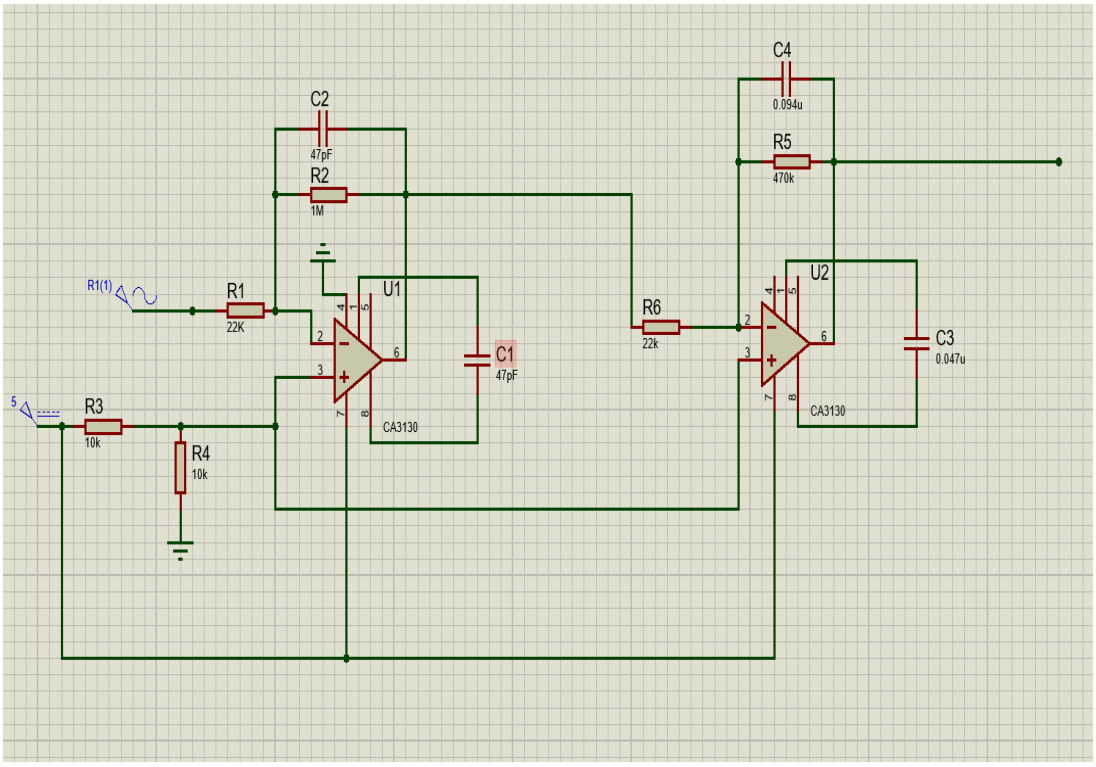

FIGURE 3.1 – Circuit amplificateur pour piézo

Ce circuit a ensuite été testé dans un récipient rempli d'eau avec le piézo immergé. Un émetteur audio (enceinte vibrante) placé à proximité du récipient a permis de transmettre les signaux DTMF dans l'eau. Ces signaux ont été reçus et décodés avec une excellente **qualité de signal**, démontrant une **très bonne SNR (Signal-to-Noise Ratio)**.

3.4 Test d'étanchéité

Une vérification simple de l'étanchéité a été réalisée : le tube contenant les éléments internes a été fermé avec les bouchons (end caps) et immergé à une profondeur de **4 mètres** pendant **1 heure**. Aucun signe d'infiltration d'eau n'a été détecté après immersion, validant le bon fonctionnement du système d'étanchéité (O-Rings + bouchons).

Conclusion

Ce stage de trois mois m'a permis de participer activement à la conception et au développement d'un robot sub-surface, depuis la modélisation jusqu'à la réalisation partielle du prototype. Ce projet m'a offert l'opportunité de travailler sur une large variété de compétences, notamment en simulation dynamique, électronique embarquée, modélisation 3D, communication I^2C, et prototypage mécanique.

J'ai pu approfondir mes connaissances en programmation bas-niveau sur Arduino et Raspberry Pi, en gestion de bus I^2C, en intégration matérielle et en conception de circuits analogiques (filtrage, amplification). Ce stage m'a également permis de mieux comprendre les contraintes du prototypage réel, l'importance des tests progressifs, et les défis de l'intégration multi-domaines (mécanique, électronique, logiciel).

Le travail réalisé constitue une base solide pour la suite du développement du robot. L'intégration future des capteurs restants (baromètre, caméra, DTMF), la mise en place d'une boucle d'asservissement en profondeur sont autant de pistes concrètes à explorer. Des essais en conditions réelles (en eau libre) permettront de finaliser la validation du prototype.

Bibliographie

[1] GitHub Repository, *OpenLog Artemis Custom Firmware, Example codes for Master and Slave*, Available at : `https://github.com/KHALEDCHATAH/Stage2A`

[2] Documentation, *Arduino Reference*, Available at : `https://www.arduino.cc/reference`

[3] Documentation, *pigpio C interface*, Available at : `https://abyz.me.uk/rpi/pigpio/`

[4] Documentation, *Raspberry Pi 4*, Available at : `https://www.raspberrypi.com/documentation/`