Rapport de Projet de Fin d'Études

Exploitation de la redondance d'actionnement pour piloter la réactivité des robots sous-marins

Août 2026

AGUIRRE Thomas
FISE 2026
Robotique Autonome

Encadrants :
Lionel LAPIERRE
Gabriel BETTON
Titouan BELIER

Rapport non confidentiel

Table des matières

Mots-clés

Robotique sous-marine, Commande 6 degrés de liberté, Quaternions, Architecture logicielle, Système embarqué

Keywords

Underwater robotics, Six-degree-of-freedom control, Quaternions, Software architecture, Embedded system

Résumé

Ce projet porte sur le développement du robot CUBE, un robot sous-marin de forme cubique composé de huit propulseurs positionnés aux sommets de la structure. La configuration de ces moteurs est issue de travaux de recherche antérieurs et permet une manœuvrabilité optimale selon les six degrés de liberté ainsi qu'un gain énergétique important. Le robot peut donc translater et effectuer des rotations selon chacun de ses axes. Le développement du CUBE se répartit donc en plusieurs phases. La première est la finalisation de la conception du robot comprenant la création et l'ajout de nouvelles pièces. Ces dernières ont pour but de faciliter le montage et le démontage du robot et donc son utilisation. Suite à cela a lieu la phase de création d'une simulation. Cette dernière se veut réaliste et représente le robot dans un bassin respectant les dimensions de celui présent sur le campus brestois de l'ENSTA. Le modèle 3D issu de la première phase de conception est donc utilisé et on y intègre une dynamique ainsi que des frottements fluides. À partir de ce jumeau numérique, une loi de commande à 6 degrés de liberté basée quaternions, et utilisant le suractionnement du robot, est créée. Sur le CUBE physique, on développe alors une architecture logicielle sous ROS 2 prête à accueillir le contrôle créé à l'aide de la simulation. Après l'usinage et l'impression des pièces, le robot est monté puis des tests en bassin sont réalisés. On obtient alors des résultats qui se sont avérés concluants et fidèles à ceux de la simulation.

Abstract

This project focuses on the development of the robot CUBE, a cube-shaped underwater robot composed of eight thrusters positioned at the vertices of the structure. The configuration of these thrusters is based on previous research and allows optimal maneuverability across the six degrees of freedom, as well as significant energy savings. The robot can therefore move in translation and rotate around each of its axes. The development of the CUBE is divided into several phases, beginning with the finalization of the robot's design, including the creation and addition of new parts. These parts are intended to facilitate the robot's assembly and disassembly, and thus its use. Following this, a realistic simulation is created, depicting the robot in a pool that matches the dimensions of the one located on the ENSTA Brest campus. The 3D model resulting from the first design phase is used, and dynamics as well as fluid friction are integrated into it. Based on this digital twin, a 6-degree-of-freedom control system based on quaternions—and utilizing the robot's overactuation—is created. On the physical CUBE, a ROS 2-based software architecture is then developed, ready to integrate the control system created through simulation. After the parts are machined and 3D-printed, the robot is assembled, and tests are conducted in the pool. The results are conclusive and consistent with those of the simulation.

1 Introduction

La croissance récente du milieu de la robotique a permis de faciliter l'exploration de milieux difficiles d'accès pour l'homme. Des grottes, cavités ou autres milieux toxiques pour nous autres humains sont désormais accessibles par l'intermédiaire de systèmes autonomes. C'est une des raisons pour lesquelles le robot Spot de Boston Dynamics a été développé. Il facilite alors l'inspection de sites nucléaires comme la zone d'exclusion de Tchernobyl en Ukraine [1]. L'exemple dont nous traiterons est l'exploration karstique sous-marine, un environnement difficile d'accès et où les déplacements sont réduits et compliqués. En effet, dans un tel contexte, nous avons donc besoin d'un robot dont la manœuvrabilité est optimale et qui est capable d'évoluer avec précision et stabilité à faible vitesse [2]. Ce cahier des charges est donc à appliquer dans un milieu sous-marin aux nombreuses contraintes de courants et dont les missions sont à réaliser en autonomie complète du fait des communications sous-marines limitées avec l'opérateur [3].

Ainsi, c'est avec cette problématique d'exploration sous-marine complexe que le robot CUBE a été développé au LIRMM de l'Université de Montpellier et au Lab-STICC de l'ENSTA, campus de Brest [4]. La collaboration entre ces deux laboratoires a permis d'imaginer un robot sous-marin de forme cubique composé de 8 propulseurs disposés sur les sommets de la structure. Les travaux de recherche de Huu Tho Dang réalisés dans le contexte de sa thèse, dont nous développerons le contenu par la suite, ont permis d'établir une configuration optimale pour les différents moteurs du robot [4]. Ces travaux permettent d'obtenir un robot capable de générer des efforts selon les 6 degrés de liberté avec une excellente manœuvrabilité. Ce dispositif permet donc de se déplacer facilement lors d'explorations karstiques où les changements de direction sont multiples et fréquents.

Au début des travaux, nous disposons donc d'une structure physique déjà existante avec les 8 moteurs installés dans la position prévue par la configuration optimale. En plus de cela, nous disposons d'un banc d'essai avec 8 moteurs alignés et un ensemble électronique qui nous permettra de faire de premiers essais et de développer notre architecture logicielle ROS 2.

C'est donc à partir de cette configuration que nous continuerons le développement du Cube. Lors de ce projet, notre mission principale est donc de mettre en service le robot avec un contrôle robuste aux différents changements en exploitant les quaternions et non les angles d'Euler. En effet, nous verrons par la suite que l'utilisation des quaternions permet de s'affranchir de nombreux problèmes de singularité que nous pouvons avoir avec les angles d'Euler. Dans ce rapport, nous verrons donc comment développer une chaîne de commande à 6 degrés de liberté robuste, basée quaternions, permettant au robot sous-marin CUBE de naviguer facilement en milieu karstique.

Pour répondre à cette problématique, notre travail sera composé de plusieurs phases qui pourront se faire en parallèle. Une première phase de finalisation de la conception du robot sera nécessaire avec la modélisation en CAO de nouvelles pièces, puis leur usinage

ou leur impression 3D avec enfin leur intégration sur le robot. Cette étape permet de compléter un montage qui, pour le moment, ne permet pas une utilisation idéale. Nous essaierons donc de faciliter le montage et le démontage du robot tout en intégrant, si nécessaire, de nouveaux capteurs. Une grande partie de ce travail sera également consacrée à l'intégration de toute l'électronique à l'intérieur du cylindre du robot, ce travail n'ayant pas été réalisé auparavant. Une seconde phase de ce projet sera consacrée à la mise en œuvre d'une simulation avec un modèle 3D fidèle placé dans les mêmes conditions que celles que nous rencontrerons en essais par la suite. Il faudra donc intégrer la dynamique du robot, ainsi que la flottabilité et les frottements fluides. Cette simulation nous permettra de développer nos algorithmes de contrôle qui pourront donc être testés immédiatement. Nous y intégrerons une architecture sous ROS 2 qui servira de référence et nous sera utile par la suite. En effet, cette dernière est intéressante dans la phase suivante du projet avec l'intégration de l'architecture logicielle, toujours sous ROS 2, sur l'ordinateur embarqué du robot. Cette étape est importante et permettra de structurer proprement les différents modules des composants, facilitant la communication entre composants et leur utilisation. Enfin, après le montage final du robot, la dernière phase sera consacrée au débogage en conditions réelles avec des essais en bassin.

L'organisation de ce travail nécessite une planification du projet à long terme. Un diagramme de Gantt prévisionnel a donc été construit dans les premiers jours de travail afin de mieux visualiser le travail attendu. Ce diagramme est consultable en annexe (fig. 32).

2 Construction du robot CUBE

Dans cette première partie, nous nous intéresserons précisément au robot CUBE physique. Nous commencerons donc par présenter ce qui nous a été donné au début du projet, que ce soit la structure déjà existante ou l'architecture électronique. Nous présenterons les différents éléments du robot et comment il a été conçu avant de poursuivre avec une présentation rapide des principaux éléments embarqués. Enfin, nous évoquerons les évolutions apportées lors de ces mois de travail, avec les nouvelles pièces mais aussi les améliorations et ajouts de composants à l'électronique.

2.1 Présentation du CUBE

2.1.1 Structure physique

Comme nous avons pu l'évoquer dans l'introduction, le robot est cubique et dispose de huit moteurs positionnés sur les sommets du cube. Il s'agit donc d'un robot sur-actionné qui se veut holonome grâce à sa configuration bien choisie. Dans cette sous-partie, nous nous intéresserons à la structure physique existante du robot qui nous a été donnée comme telle au début du projet.

Ainsi, comme nous pouvons le voir ci-dessous (fig. 1), on constate que le robot est composé de douze barres de profilés (point 1) sur lesquelles sont vissées des plaques de renfort triangulaires (2) qui maintiennent les barres entre elles. Le tout permet donc d'obtenir une structure cubique solide. Sur les coins, les plaques sont parfois remplacées par des pièces pyramidales triangulaires noires issues d'impressions 3D (3) sur lesquelles sont vissés et fixés les moteurs (4). Sur la figure 1, on peut également constater la présence d'échosondeurs (5). Sur la face de devant depuis la prise de vue de la photo, on remarque la présence de pièces en POM-C usinées (6) pour leur maintien. Cette même pièce est stabilisée par quatre tiges filetées (7) qui sont elles-mêmes vissées à des pièces en angle (8), fixées sur les barres de profilés, que l'on peut observer sur les quatre coins de la face. Cette structure est, pour le moment, répétée sur trois des six faces du cube.

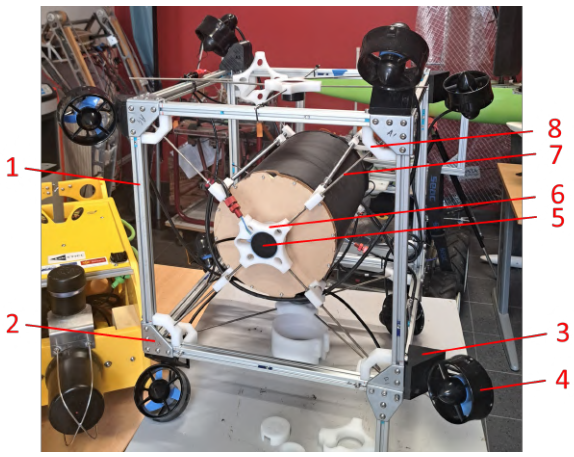


FIGURE 1 – Structure existante du robot CUBE

Continuons dans la présentation de la structure physique en nous intéressant au maintien du cylindre au centre du robot (fig. 2). Précisons d'abord qu'il s'agit d'un *Watertight Enclosure Tube* en aluminium de diamètre interne 200 mm et de longueur 300 mm. Sur chaque extrémité se trouve un *O-ring Flange* de diamètre interne 200 mm, pièce sur laquelle on positionne les joints et visse la tape. Enfin, la tape arrière est une *End Cap - Aluminum – 5 x M10 Holes* et la tape avant est une *End Cap - Acrylic - Blank - 75 mm*. Vous pouvez trouver les spécificités et illustrations de ces pièces sur le site de BlueRobotics ou en annexe 34.

Pour le maintien, on constate donc la présence de deux pièces en POM-C usinées, une première (1) liée à la tige de maintien (2) de l'échosondeur, pièce dans laquelle une plus grosse tige M10 filetée (3) servant au maintien du cylindre est vissée. Sur cette même tige, on ajoute donc la seconde pièce en POM-C (4) qui vient se plaquer contre le tube et réalise un épaulement. L'avantage de cette pièce est qu'elle peut tourner pour libérer le cylindre et faciliter le démontage. En reproduisant ce processus à quatre reprises sur cette face et sur la face opposée (prise de vue fig. 1), on obtient un maintien stable du tube qui peut donc être relativement facilement démonté. Le démontage est facilité par la possibilité d'écarter les tiges de maintien (3), laissant passer le cylindre sur le côté.

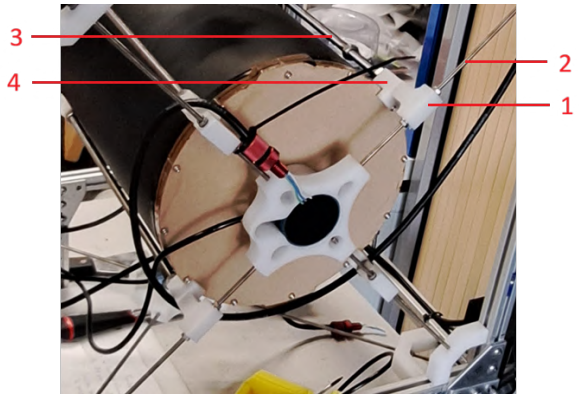


FIGURE 2 – Structure existante du robot CUBE

On notera néanmoins quelques défauts à cette structure. D'abord, l'ensemble des échosondeurs n'a pas été fixé puisque seuls ceux présents sur les faces avant et arrière (toujours depuis la prise de la fig. 1) sont présents ainsi que celui sur la face inférieure. Il faudra donc rajouter les autres échosondeurs et, nous le verrons par la suite, cela nécessitera la conception de nouvelles pièces. Un autre problème important pour la suite du projet est l'absence complète de structure interne au cylindre. En effet, aucune pièce n'a été prévue pour le maintien de l'architecture électronique à l'intérieur du robot. Une grande phase de conception sera donc nécessaire. Enfin, dernier problème, la tape arrière ne présente pas suffisamment de trous pour l'ensemble des câbles des différents éléments embarqués à l'extérieur et devant communiquer avec l'intérieur.

2.1.2 Configuration des moteurs

Dans la sous-partie précédente, nous avons très rapidement évoqué la disposition des moteurs sur la structure physique du robot. En effet, ces propulseurs sont fixés à des pyramides noires obtenues par impression 3D, mais celles-ci sont toutes différentes. Elles ont des ouvertures bien spécifiques et les hélices y sont positionnées avec des angles uniques à chacune. Cette disposition permet donc d'obtenir des moteurs avec des positions et des orientations bien précises.

Néanmoins, les modèles 3D des pyramides ainsi que la manière dont ont été fixés les propulseurs ne nous ont pas été donnés directement. Dans cette sous-partie, nous présenterons très rapidement les travaux de Huu Tho Dang et sa thèse sur « *Underwater robots for karst and marine exploration : A study of redundant AUVs* » [4], vers laquelle je vous renvoie pour tous les détails, et dans laquelle il se penche sur l'optimisation de la configuration des robots sous-marins sur-actionnés. Son mémoire est un pilier de ce projet et nous a permis d'éviter une longue phase de positionnement, sans doute empirique, à laquelle nous aurions pu faire face.

Ainsi, dans le chapitre 5 sur l'étude des configurations statiques, Huu Tho Dang traite du cas du robot CUBE et cherche à établir celle des huit propulseurs permettant de contrôler pleinement les six degrés de liberté. Son travail se fonde donc sur des positions déjà connues (5.1.2 de la thèse), les moteurs étant positionnés sur les sommets de la structure cubique, comme nous l'avons précisé précédemment. Il s'agit donc désormais de déterminer l'orientation de chacun des huit actionneurs et des vecteurs de poussée. Ces orientations doivent permettre d'obtenir les meilleures performances en termes de manœuvrabilité, d'efficacité énergétique, de réactivité et, bien sûr, être capables de générer des forces et couples selon chacune des directions. À partir de ces critères et d'un travail d'optimisation qui en découle (je vous invite une nouvelle fois à regarder l'ensemble de son travail), Huu Tho Dang obtient une matrice de configuration *A2* et donc une configuration *C2* pour les huit moteurs visibles ci-dessous (fig. 3).

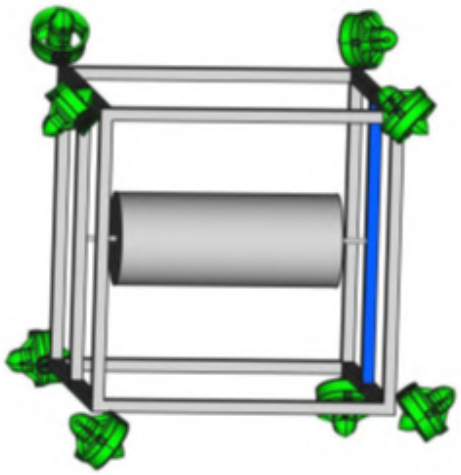


FIGURE 3 – Configuration *C2* obtenue à partir de la thèse de Huu Tho Dang (fig. 5.6)

On obtient ainsi une configuration bien précise optimisant les différents indices que nous avons cités plus haut. La configuration *C2* permet bien de générer des forces et couples selon les six degrés du robot rendant sa manoeuvrabilité omnidirectionnelle. La figure 3 montre bien que les fixations mécaniques des moteurs sur la structure ne sont pas définies. L'ajout des pyramides sur notre prototype permet donc d'assurer ces fixations tout en respectant la configuration *C2*.

2.1.3 Architecture électronique

Comme indiqué dans l'introduction, le robot nous a été donné en plusieurs parties. D'abord, la structure cubique que nous venons de traiter avec les huit propulseurs installés ainsi que le tube BlueRobotics vide. Ensuite, nous avons l'électronique installée sur un banc d'essai en bois (fig. 4) avec huit autres propulseurs que nous pourrons utiliser pour construire notre architecture logicielle et nos tests.

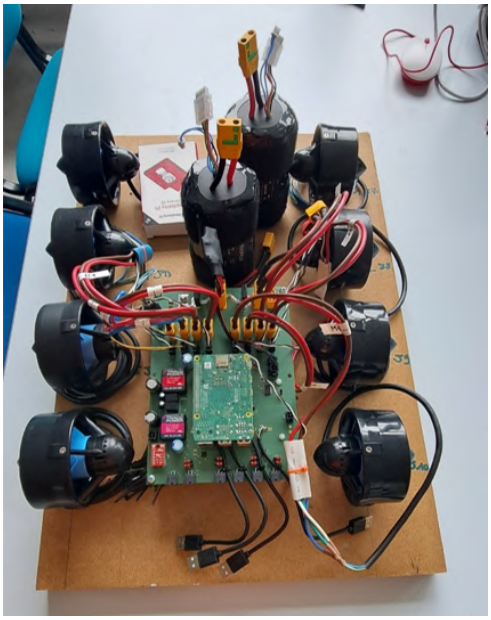


FIGURE 4 – Banc d'essai du robot comprenant l'électronique, les batteries et huit moteurs

Il est alors intéressant de lister les différents éléments de l'électronique présente sur ce banc d'essai puisqu'il s'agira de la même que celle présente dans le tube lors du montage final du robot par la suite. On a donc :

— 8 propulseurs T200 Thrusters de BlueRobotics. Il s'agit de moteurs bidirectionnels brushless accompagnés d'ESC [11]
— 1 carte de contrôle personnalisée (custom PCB) faite spécifiquement pour le robot CUBE et qui aura le rôle d'interface centrale entre les différents composants en permettant la gestion d'énergie et le port d'ordinateurs embarqués.
— Une Raspberry Pi 4 pour le calcul embarqué et l'exécution des différents programmes.

— 1 capteur BME280 permettant de mesurer la pression atmosphérique, la température et l'humidité. Installé sur la plaque PCB, et donc présent à l'intérieur du tube, on peut supposer que son rôle est simplement de pouvoir évaluer la température à l'intérieur du robot et d'éviter les surchauffes en mission [12]

— 1 centrale inertielle ICM-20948 pour estimer l'état du robot avec un accéléromètre, un gyroscope et un magnétomètre intégrés [10]

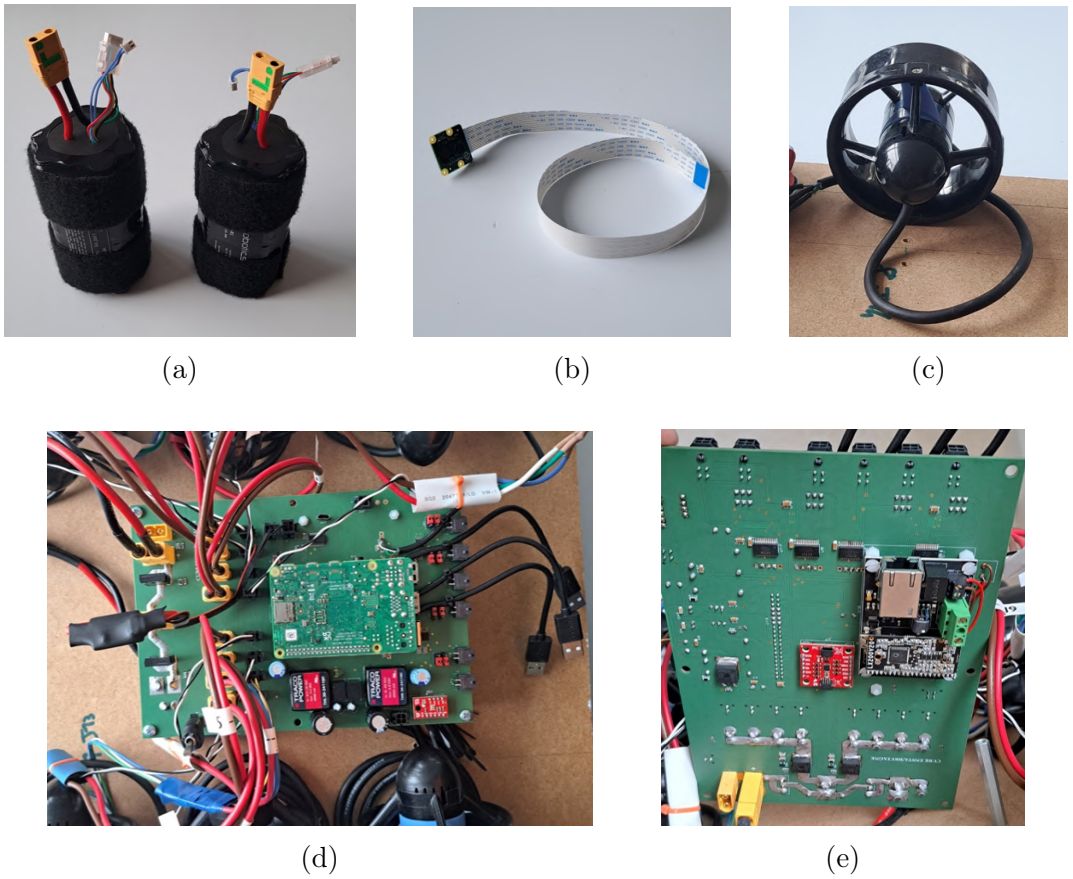
— 1 carte d'interface Ethernet BR-FATHOM-X-R1 de BlueRobotics permettant d'établir une liaison réseau Ethernet longue distance et haut débit entre le ROV et l'ordinateur de l'utilisateur [13].

— 1 microcontrôleur ATmega328P permettant de communiquer en I2C et d'assurer le lien entre la Raspberry Pi 4 et les huit propulseurs [14]

— 1 Raspberry Pi Camera V2 pouvant être facilement utilisée avec les Raspberry Pi et filmer en 1080p/30 fps.

— 2 batteries LI-4S-18AH-R1, lithium, 4S, 18 Ah de capacité

— 6 Ping Sonars de BlueRobotics, non intégrés à l'architecture électronique mais certains présents sur la structure (fig. 2)



(a) (b) (c)

(d) (e)

FIGURE 5 – Principaux composants de l'architecture matérielle du robot : (a) Battery-LI-4S-18AH-R1 - (b) Raspberry Pi Camera V2 - (c) T200 Thrusters BlueRobotics - (d) Carte PCB (grande plaque verte), au milieu la Raspberry Pi 4, en bas à droite en rouge le BME280 - (e) En rouge l'ICM-20948 et en noir le BR-FATHOM-X-R1 de BlueRobotics

Enfin, pour terminer avec cette partie, vous trouverez ci-dessous un schéma simplifié représentant l'architecture électronique globale et les communications entre les différents composants (fig. 6). En annexe, vous trouverez le schéma électronique complet du robot (réalisé par le centre de ressources du laboratoire).

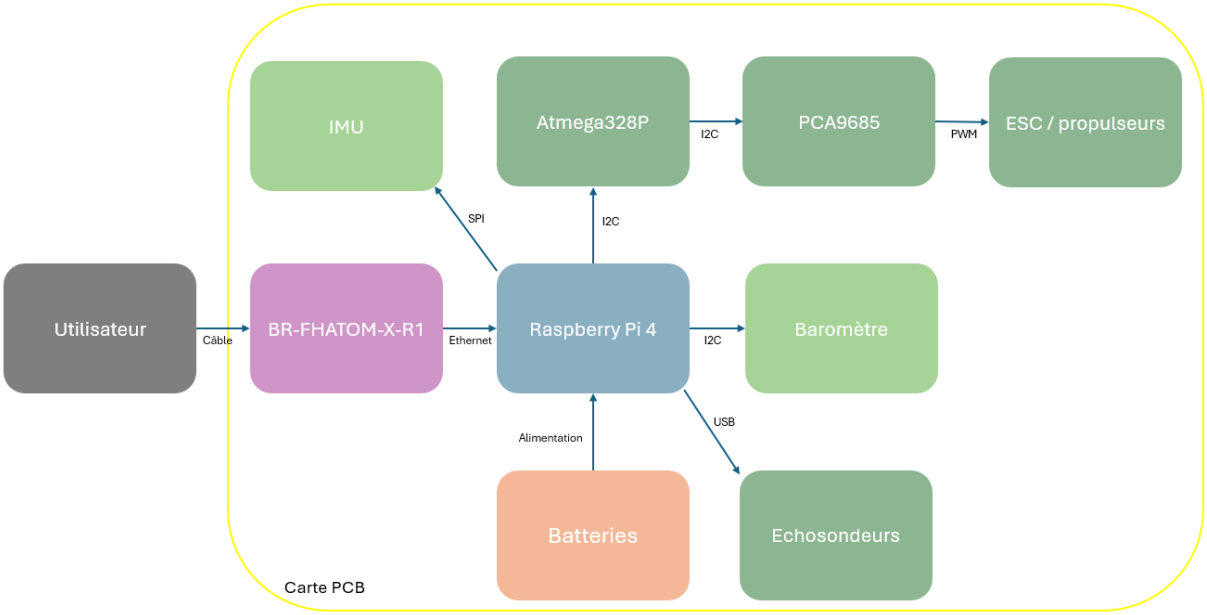


FIGURE 6 – Schéma simplifié de l'électronique du robot

Ce schéma montre évidemment le rôle essentiel et centralisé de la Raspberry Pi 4 qui permet de faire le lien entre l'ensemble des composants. On constate également l'importance de la carte PCB qui héberge physiquement les différents éléments et facilite la communication. Avec cette carte et ses étages d'alimentation comme support, les batteries alimentent donc la Raspberry Pi 4 qui, supervisée par l'utilisateur par une liaison Ethernet du BR-FATHOM-X-R1, communique en I2C, USB ou SPI, avec les capteurs. Enfin, à l'aide d'une communication I2C vers l'ATmega328P, les commandes sont envoyées au PCA9685 puis aux ESC.

Ainsi, c'est à partir de ce schéma simplifié que nous pourrons construire nos communications par la suite. Nous développerons donc toute l'architecture logicielle à partir de cette référence, mais cela fera l'objet d'une partie ultérieure du rapport.

2.2 Évolutions apportées au robot

Comme nous avons pu le dire dans la section 2.1.1, la structure physique initiale du robot CUBE était incomplète. L'intérieur du cylindre était vide et l'extérieur ne permettait pas un montage et un démontage faciles. Ainsi, nous commencerons par une modification de l'extérieur du robot avec l'ajout de nouvelles pièces. Par extérieur, on entend donc l'extérieur du cylindre et ce qui concerne la structure cubique. Il nous faudra donc également développer l'ensemble de la structure interne afin de maintenir l'électronique tout en permettant aux câbles de se diriger vers la tape arrière et de communiquer avec

l'extérieur. Cette sous-partie se consacrera d'abord à la conception des nouvelles pièces puis à l'assemblage du robot à partir des nouveautés développées.

2.2.1 Conception mécanique de nouvelles pièces

Pour ce travail de conception, on utilise *Autodesk Inventor Professional 2027*. On notera tout de même que la CAO initiale qui nous a été donnée contenait uniquement des barres profilées ainsi que 2 équerres par coin pour le maintien de la structure. Un travail que nous ferons portera donc sur l'assemblage global du robot et nous en verrons l'évolution ici. On commencera donc par la modification de la tape du cylindre, puis nous verrons l'ajout de coins aux faces de la structure cubique, d'un anneau central pour le démontage facile ainsi que la création de nouvelles pièces pour le maintien du cylindre. Enfin, nous terminerons avec le développement de multiples pièces pour la structure interne du robot.

Modification de la tape du cylindre Une des premières problématiques dont nous allons traiter est l'ajout de trous à la tape arrière du cylindre. En effet, le capuchon dont nous disposons est, comme nous avons pu le dire, le *End Cap - Aluminum - 5xM10 Hole - 75 mm* (Annexe 34). Il dispose donc uniquement de 5 trous M10, ce qui n'est évidemment pas suffisant. Commençons par lister l'ensemble des trous dont nous avons besoin ainsi que leur fonction. Nous en avons donc :

— 8 pour les moteurs Blue Robotics
— 6 pour les échosondeurs
— 1 pour le switch on/off
— 1 pour l'ombilical
— 1 pour le bouchon de mise sous vide
— 1 pour le capteur de pression
— 3 supplémentaires pour l'ajout ultérieur de composants

On obtient donc un total de vingt et un trous pour uniquement cinq déjà présents, soit seize trous à rajouter. On obtient donc le résultat ci-dessous (fig. 7).

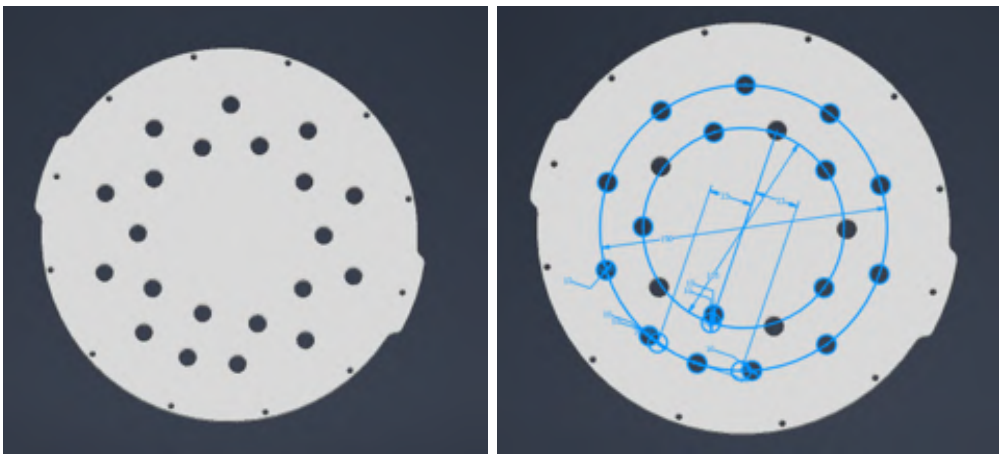


FIGURE 7 – Tape arrière modifiée, les nouveaux trous en bleu, les anciens non colorés

Cette nouvelle disposition est donc composée de quatre motifs circulaires à cinq trous espacés régulièrement le long des cercles. Ce motif est particulièrement visible sur le cercle intérieur avec la disposition de cinq nouveaux trous bleus se positionnant entre les originaux. L'ajout du vingt et unième trou et d'un petit décalage des trous environnants (partie basse de l'image de droite, fig. 7) permet de créer de l'espace suffisant pour permettre un montage et un démontage faciles des bouchons et pénétrateurs à l'aide du Bulkhead (fig. 8).

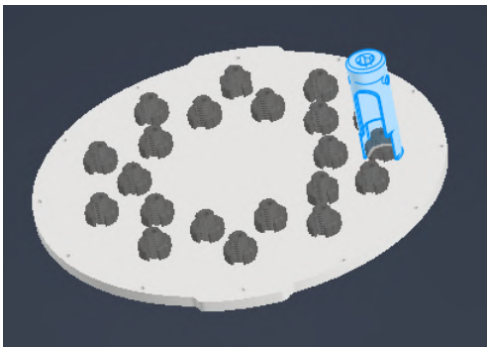


FIGURE 8 – Modélisation de l'utilisation du Bulkhead pour le montage et démontage des pénétrateurs

Création de coins supplémentaires Comme nous avons pu le voir dans la partie 2.1.1, et particulièrement avec la figure 1, le cube dispose, sur certaines faces, de pièces blanches en POM-C qui font les angles et maintiennent les tiges filetées. Pour rappel, ces pièces-là, que nous appellerons "pièces angulaires", permettent de positionner les tiges qui maintiennent les échosondeurs. L'idée dans cette partie est de reproduire ce processus sur l'ensemble des faces du cube. La seule contrainte que nous avons est la suivante : dans un coin, nous devons avoir trois pièces angulaires de taille différente afin de pouvoir visser chacune des pièces sans interférer avec le montage des autres. La conception est très simple et on obtient, dans chaque coin, le résultat suivant (fig. 9) :

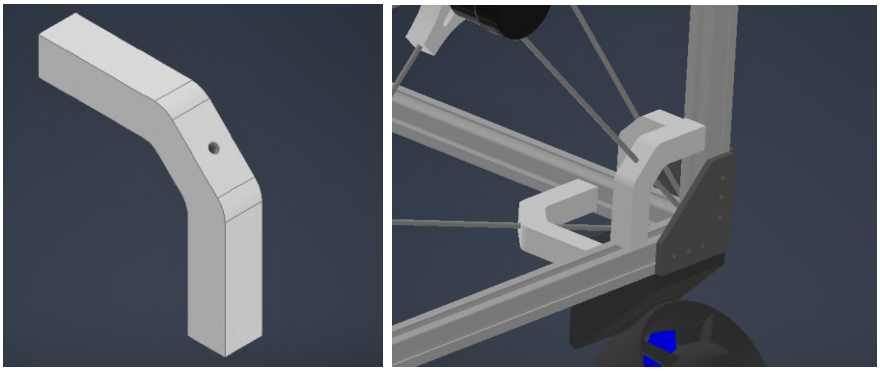


FIGURE 9 – Modélisation des pièces angulaires sur chaque coin du cube

Nouveau maintien du tube La structure de maintien du cube déjà existante de la partie 2.1.1 (visible sur les fig. 1 et 2) est fonctionnelle mais, comme nous l'avons dit à

plusieurs reprises, ne permet pas un démontage facile du robot. En effet, pour ouvrir le cylindre et retirer l'électronique, nous devons le sortir de son emplacement par le côté pour l'ouvrir car le dispositif ne permet pas de le faire autrement (fig. 20). L'idée est donc d'avoir un anneau qui laissera passer la tape avant, et cela tout en maintenant la structure. Se trouvant sur le même plan que la face du cube, elle doit également pouvoir se fixer à la structure tout en portant l'échosondeur. Après de nombreuses versions de la pièce, nous en sommes arrivés à la pièce finale suivante :

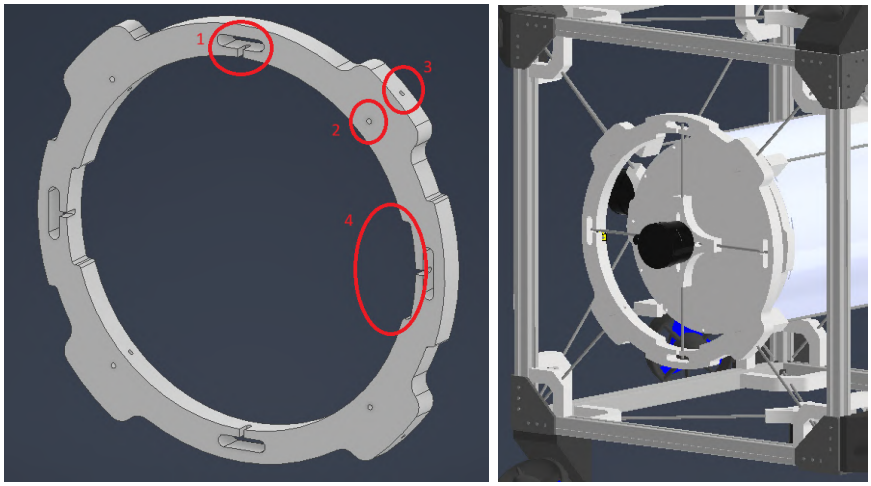


FIGURE 10 – Modélisation du nouvel anneau central de maintien du cube, à gauche la pièce isolée, à droite la pièce dans le montage

La version finale de l'anneau permet de répondre à plusieurs points du cahier des charges que l'on s'était fixés. D'abord, le point numéro 1 de la fig. 10 permet de venir y installer les tiges qui servent au maintien de l'échosondeur. La présence de lumières permet de sortir facilement ces barres et donc de démonter l'échosondeur de manière efficace. Un système vis-écrou est alors utilisé pour figer le tout sur les quatre tiges présentes. Ensuite, le point numéro 2 correspond à quatre trous, idéalement taraudés, qui permettent d'accueillir les longues tiges de maintien du cylindre (point 3 de la figure 2). Puis, il y a le point 3, il s'agit également de trous qui viendront accueillir les tiges fixées aux pièces angulaires. Celles-ci permettent donc de maintenir l'anneau en place et de le garder fixe par rapport à la structure cubique. Enfin, il y a le point 4, des petits creux qui permettent de faire passer les « oreilles » de la tape, que ce soit la tape avant ou la tape arrière selon la configuration.

Ainsi, pour réaliser le démontage, et inversement le montage, on commence par desserrer les systèmes vis-écrou des quatre tiges de l'échosondeur (point 1) puis on retire son ensemble. Ensuite, on retire la tape du cylindre en la faisant passer au travers de l'anneau et on accède à l'électronique. Ainsi, l'anneau reste toujours en position et n'a pas besoin d'être démonté.

D'autres petites pièces ont également été créées pour faciliter le maintien du cylindre et l'extraction de la tape (fig. 11). En répétant ce motif d'épaulement quatre fois, comme nous pouvions déjà l'observer sur la figure 10, on obtient, de ce côté du cylindre, un

maintien idéal et un système de démontage facile. Une nouvelle fois, le maintien de ces pièces rotatives se fera par un système vis-écrou.

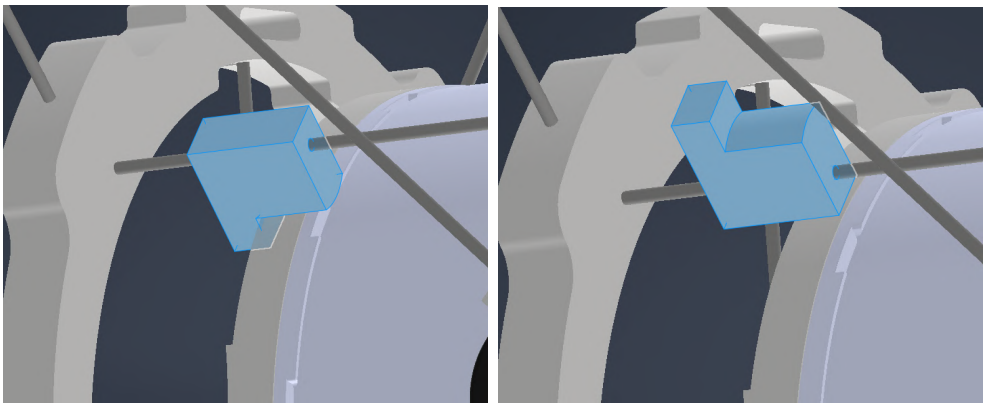


FIGURE 11 – Pièces rotatives de maintien du cylindre, à gauche en position de maintien, à droite en position libre pour laisser sortir la tape

Cependant, il faut également réaliser cet épaulement de l'autre côté du cylindre. Nous en sommes arrivés à la conception suivante :

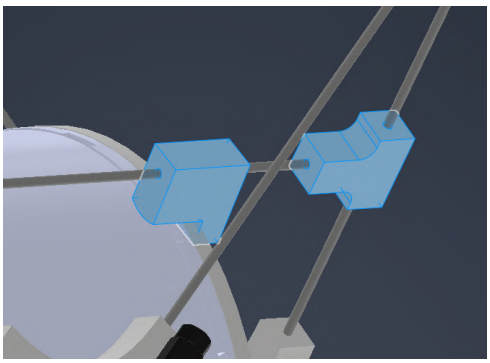


FIGURE 12 – Pièces de maintien du cylindre à l'arrière du cylindre

Ce concept s'inspire fortement de celui déjà existant observable dans la figure 2. La pièce de gauche est identique à celle que nous avions dans la figure 11 et permet de réaliser un épaulement du cylindre. La pièce de droite permet de maintenir la tige de maintien du cylindre en place (point 3, fig. 2) et peut se glisser le long de la barre qui la traverse. Ce glissement permet notamment d'obtenir une fixation solide du cylindre en rapprochant au plus près les pièces de la tape. Une fois de plus, on répète ce processus à quatre reprises et de manière identique au côté de la tape opposée.

Ainsi, le tout permet d'obtenir un maintien idéal du cylindre, les petites pièces que nous venons de voir permettent de maintenir le cylindre en place tandis que l'anneau central permet de sortir la tape sans devoir démonter la structure entourant le tube. Avec ce nouveau concept pour la structure externe, beaucoup de pièces n'ont pas à bouger. En effet, seules les pièces de la figure 11 doivent tourner pour libérer le passage, puis la tape est extraite, les autres pièces restent fixes. On notera néanmoins que l'anneau central, et

la structure l'entourant, peut être plutôt placé du côté de la tape arrière afin de faciliter l'extraction de l'électronique à l'intérieur.

Pour conclure, voici donc le concept final que nous avons obtenu pour la structure externe du robot CUBE :

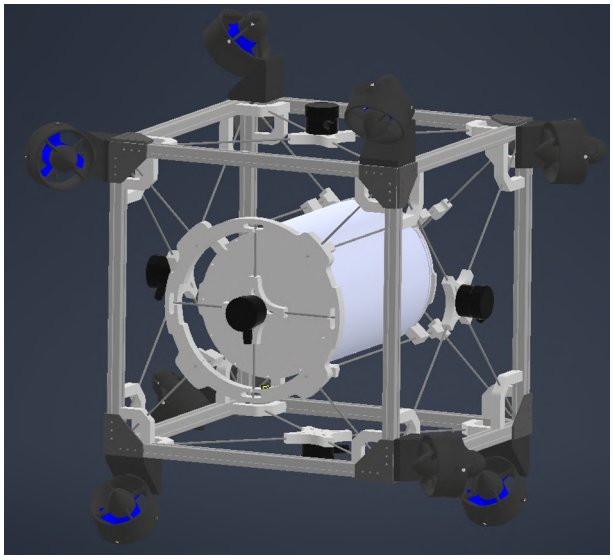


FIGURE 13 – Concept final pour la structure externe du robot CUBE

Structure interne du cylindre Désormais, on s'intéresse à la partie interne du cylindre. Nous devons donc placer l'électronique de la partie 2.1.3 dans un cylindre de 200 mm de diamètre (diamètre interne du tube BlueRobotics) et de 240 mm de long. Le tube étant bien de 300 mm de long, mais la présence des deux *O-ring Flanges* réduit la longueur exploitable à l'intérieur du tube. L'idée que nous avons développée est de construire un cylindre qui pourra se glisser à l'intérieur du tube BlueRobotics. On construit alors ce tube en créant d'abord deux anneaux sur lesquels nous visserons deux plaques pour les maintenir entre eux. Nous en sommes arrivés aux concepts suivants (fig. 14) :

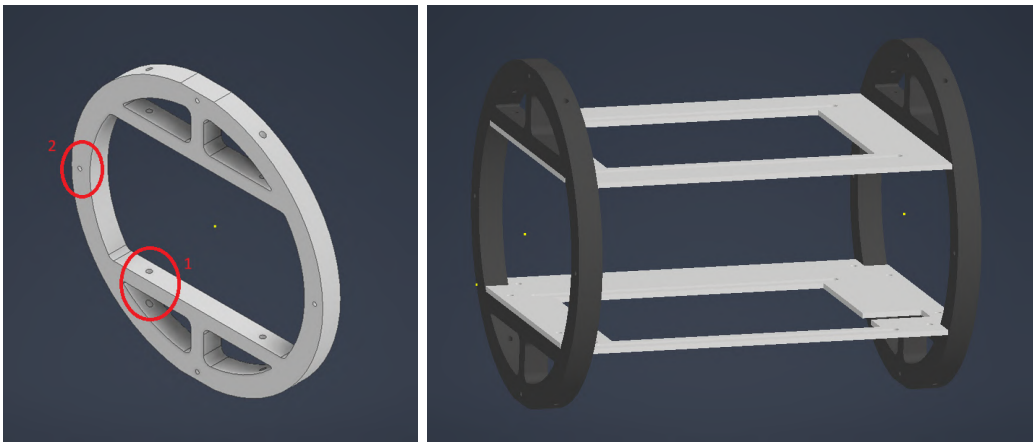


FIGURE 14 – Conceptions des anneaux internes et premier exemple de montage

Sur la gauche, on peut donc voir la pièce finale que nous avons obtenue. Elle permet de répondre à plusieurs choses. D'abord, le trou du point 1, présent quatre fois sur la pièce, permet de visser la plaque comme on peut le voir sur le montage de la figure 14. L'espace en dessous nous permet de passer un écrou et de réaliser un système vis-écrou une nouvelle fois. En réalisant cela avec les deux plaques, on obtient un montage relativement solide qui vient se glisser parfaitement dans le cylindre.

Le point 2 permet quant à lui de fixer la rotation à l'intérieur du cylindre. La présence de trous aux mêmes endroits sur les *O-ring Flanges* permet d'insérer de longues tiges traversant également notre anneau afin de l'empêcher de tourner à l'intérieur du cylindre. On obtient donc un montage solide qui ne bouge pas à l'intérieur du cylindre.

Intéressons-nous désormais à la conception des plaques, présentes dans la figure 14, dont voici le modèle :

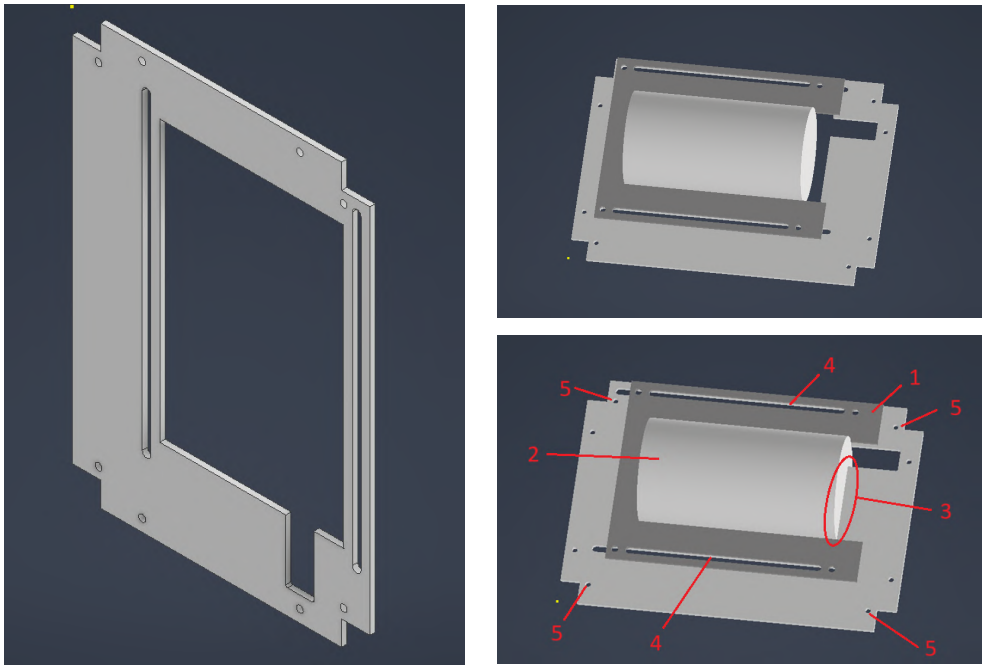


FIGURE 15 – Plaques du montage à l'intérieur du cylindre du robot CUBE ainsi que le système de fixation de la batterie

Sur la gauche, on peut donc voir l'une des deux plaques du montage interne et, sur la droite, on peut observer le système qui permet de serrer les batteries. Le concept est relativement simple : à l'aide d'une autre plaque (point 1), on insère la batterie (2) comme dans l'image du haut, puis on fait coulisser cette même plaque jusqu'à ce que la batterie bute contre les bords (3). À cela, on glisse dans les lumières (4) des bandes auto-agrippantes qui permettent de compléter la fixation de la batterie en l'entourant. Sur cette première plaque, on dispose également de quatre trous (5) qui permettent de visser les entretoises, servant de supports de l'électronique. On remarquera néanmoins que la batterie n'est pas centrée sur cette plaque alors que c'est le cas sur l'autre. Cela s'explique simplement par le peu d'espace présent dans le cylindre et la nécessité de laisser de la place à certains composants.

Enfin, en ajoutant donc l'électronique, l'autre plaque ainsi que les batteries, on obtient le montage interne suivant (fig. 16) :

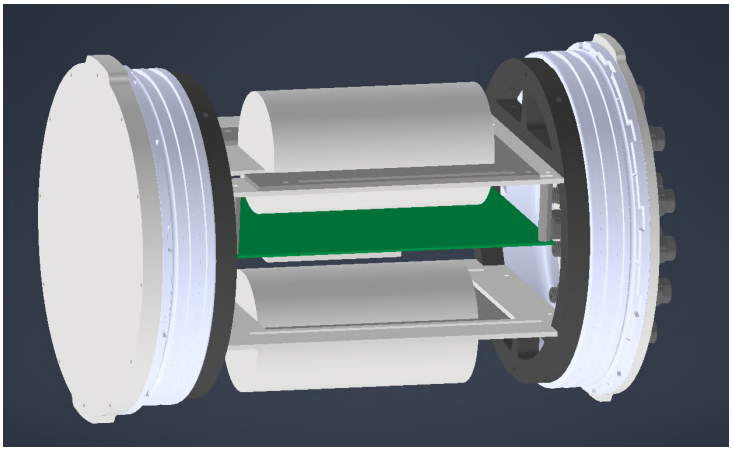


FIGURE 16 – Montage final à l'intérieur du tube du robot CUBE

En vert, l'électronique de la figure 4, accrochée à la plaque supérieure par l'intermédiaire d'entretoises. Les trous dans les anneaux noirs permettent de laisser passer les câbles pour atteindre la tape arrière tandis que le positionnement des batteries permet de laisser suffisamment de place aux différents composants électroniques. Le tout se glisse et se fige parfaitement dans le tube BlueRobotics.

2.2.2 Assemblage du robot

Après cette longue phase de conception de pièces et du montage du robot CUBE, intéressons-nous désormais à l'assemblage physique de ce que nous avons vu. Néanmoins, après une longue attente et de nombreux retards pour la conception des pièces, qui devaient être réalisées par une personne interne au laboratoire, uniquement une pièce a été usinée. En effet, nous disposons d'une version de l'anneau central en POM-C mais ne comprenant pas l'ensemble des trous nécessaires à l'installation des barres (Annexe 35). Ainsi, nous avons pris la décision tardive de ne pas mettre en place la nouvelle structure externe que nous avions conçue et de conserver celle existante, bien que moins pratique. Cependant, il est nécessaire de rajouter des trous à la tape ainsi que de réaliser les pièces internes au cylindre, et c'est donc ce que nous verrons dans cette partie en plus de l'intégration électronique.

Modification de la tape du cylindre Pour les mêmes raisons que précédemment, nous avons dû trouver une solution alternative afin d'ajouter les trous sur la tape du cylindre. L'utilisation d'une découpeuse au jet d'eau était privilégiée, mais nous avons finalement dû opter pour une perceuse à colonne, jugée moins précise pour le positionnement. Les trous ont donc été placés manuellement, mais le résultat reste très convaincant (fig. 17). On notera néanmoins la présence de vingt trous, contre vingt et un initialement, et donc la suppression d'un trou dit « supplémentaire » de la partie 2.2.1, et ce pour faciliter le perçage manuel.

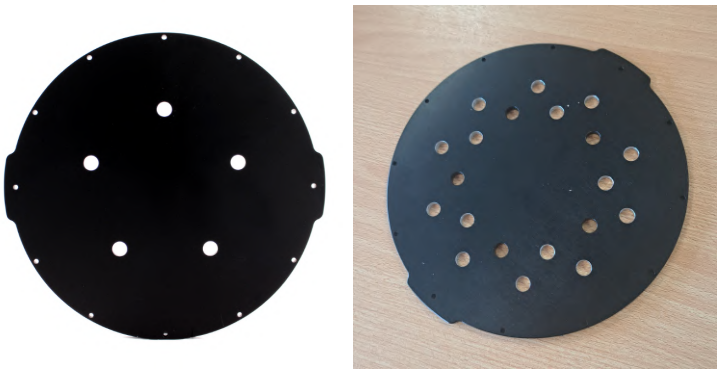


FIGURE 17 – Ajout de 15 trous sur la tape arrière du robot CUBE, à gauche le modèle de départ et à droite après modification

Structure interne du cylindre Intéressons-nous désormais aux pièces de la structure interne du cylindre. A défaut de pouvoir usiner les pièces en POM-C, nous avons donc fait le choix de l'impression 3D. Nous utilisons alors une imprimante *Bambu Lab A1 0.4 nozzle* et du *PETG HF*. Les principaux paramètres d'impression sont les suivants : Hauteur de couche de 0.16 mm, 6 parois, motif de remplissage en quadrillage. Ces paramètres nous permettent d'obtenir des pièces relativement solides, prêtes à supporter le poids de toute l'électronique. On obtient alors l'ensemble des pièces de la partie *Structure interne du cylindre* de la partie 2.2.1 ainsi que le montage à vide :

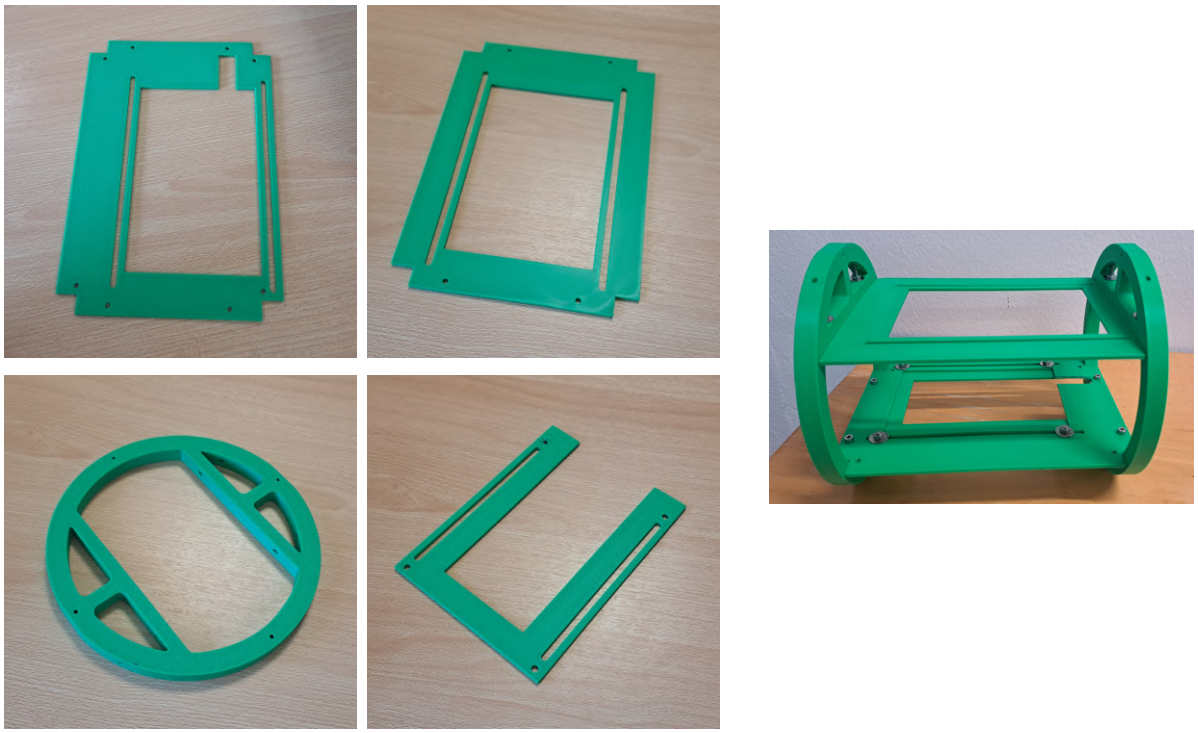


FIGURE 18 – Impression 3D des différentes pièces de la structure interne du robot CUBE et montage de la structure

Intégration électronique Ainsi, on réalise le montage de la structure interne du cylindre avec l'électronique et on obtient le premier résultat suivant :

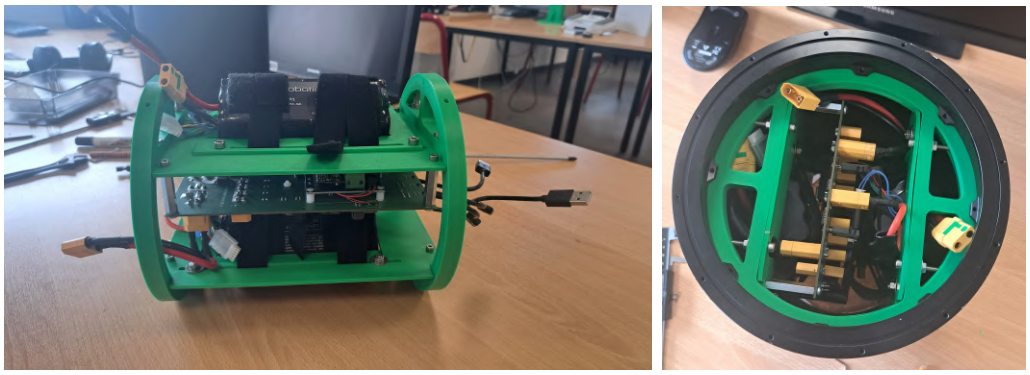


FIGURE 19 – Montage de la structure interne, hors et dans le cylindre (ESC absents pour l'instant)

On constate donc que, assez logiquement, notre structure s'insère parfaitement dans le cylindre et sa longueur permet d'être serrée entre les deux *O-ring flanges* de chaque côté du tube.

Après cela, on soude deux embouts mâles de câbles Ethernet à un câble Fathom ROV Tether de 25 mètres. Cela nous permettra d'obtenir un long câble Ethernet pour nous connecter au robot lorsqu'il fonctionnera sous l'eau, et ce sans perdre la connexion. On fait passer ce câble au travers de la tape arrière, tout comme les câbles des huit propulseurs ou du capteur de pression, en s'assurant de l'étanchéité et en y ajoutant également des bouchons (fig. 21).

On ajoute ensuite les câbles des moteurs avec des dominos électriques intégrés afin de rendre les liaisons électriques démontables. En effet, cela a un avantage majeur lors de la sortie de l'électronique du tube : pour démonter, on peut retirer la tape arrière (fig. 20) puis débrancher chacun des dominos, le câble ethernet et les câbles du switch, pour pouvoir accéder sans souci à la structure interne. Celle-ci est alors séparée complètement de la tape arrière. On peut alors la retirer et l'utiliser loin du robot CUBE, tandis que les propulseurs et leurs câbles restent donc sur la structure cubique :

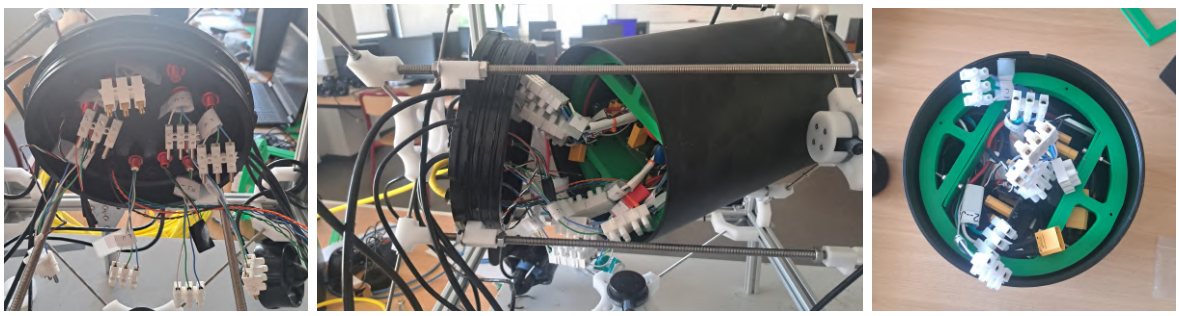


FIGURE 20 – Au milieu le montage avec branchement des dominos électriques, à gauche la partie tape du montage, à droite la partie tube du montage

Une fois le montage précédent réalisé et positionné sur le Cube, on obtient notre montage final prêt à être mis à l'eau :

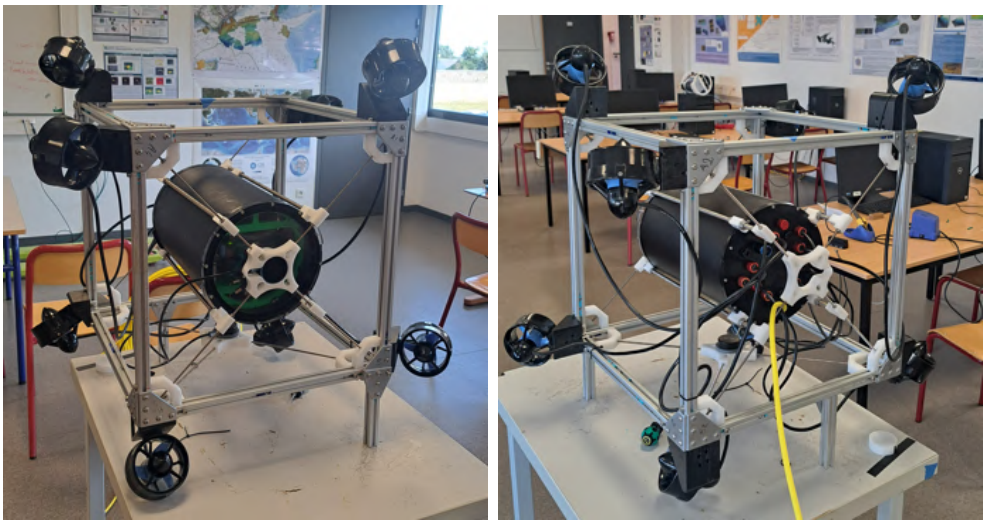


FIGURE 21 – Montage final du robot CUBE

3 Architecture logicielle

Maintenant que nous avons identifié l'architecture matérielle du robot, on peut se pencher sur l'environnement logiciel afin de communiquer avec les différents composants et de créer des connexions entre eux. Dans cette partie, nous nous pencherons d'abord rapidement sur l'installation du système d'exploitation avant d'évoquer la communication avec les composants et le développement de l'architecture logicielle.

3.1 Installation OS et ROS

Afin d'arriver à la mise en place de l'architecture logicielle sur le robot CUBE, nous devons donc d'abord installer un système d'exploitation et ROS 2. Ainsi, sur la carte SD de la Raspberry Pi 4, on installe d'abord Ubuntu Server 22.04.5 (64 bits). Avec cela, on peut déjà se connecter en SSH à la Raspberry PI. Puis, on réalise l'installation de ROS 2 Humble sur le robot [8]. À partir de cela, nous allons donc pouvoir essayer de communiquer avec les différents composants et construire notre architecture ROS 2.

3.2 Communication avec les composants

3.2.1 IMU

Mise en place du node et du driver Comme nous avons pu le dire précédemment, la centrale inertielle ICM-20948 communique avec la Raspberry Pi 4 en SPI [10]. Après l'activation de l'interface SPI sur la Rasp, la communication avec les composants SPI, et donc l'IMU, est possible et permet de vérifier son bon fonctionnement.

On crée donc le node et le driver IMU, ceux-ci nous permettent pour le moment de visualiser les valeurs du gyroscope et de l'accéléromètre selon les axes x, y et z. La connexion à l'IMU permet de constater néanmoins que les premières valeurs obtenues sont absurdes et qu'une calibration est nécessaire. Pour ce faire, on utilise une méthode que l'on place dans notre driver. La calibration est faite de manière statique sur plusieurs milliers de valeurs (au choix) afin de compenser le biais conséquent présent au départ sur les différents axes. Pour le gyroscope, on suit les étapes suivantes :

— Démarrage de l'IMU : premières valeurs ignorées
— Acquisition de n valeurs avec IMU fixe
— Calcul de la moyenne du gyroscope pour le biais pur et calcul de offset_{axe}
— Calibration gyroscope (gyro_scale sensibilité constructeur du capteur) :

$$g_{axe\_correction} = \frac{g_{axe\_mesure} - \text{offset}_{axe}}{\text{gyro\_scale}} \tag{1}$$

— Sauvegarde des valeurs de calibration

De son côté, l'accéléromètre bénéficie d'une mise à l'échelle sans suppression de biais. On vérifie uniquement son bon fonctionnement en s'assurant d'une norme totale proche de *1 g*.

On fixe néanmoins des conventions bien précises que nous utiliserons pour le contrôle également :

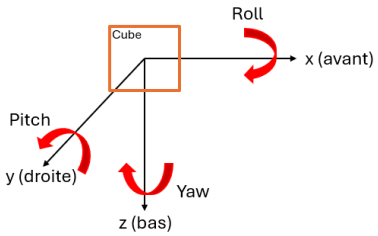


FIGURE 22 – Conventions fixées pour les axes du robot CUBE

Ces conventions sont donc évidemment les mêmes pour le gyroscope et l'accéléromètre. Néanmoins, plusieurs signes ont dû être inversés et des axes interchangés pour les respecter, les axes de l'ICM-20948 n'étant pas comme ceux désirés. En effet, on constate que, dans sa configuration de départ, x est bien vers l'avant mais l'axe z est vers le haut et y vers la gauche.

Pendant un temps, nous avons également rencontré un défaut matériel sur la centrale inertielle initialement installée sur le robot. En effet, l'axe z de l'accéléromètre restait bloqué sur 32767, valeur semblant correspondre à la saturation, alors que les autres axes ne présentaient pas ce problème. Après de nombreuses tentatives de résolution, un changement d'IMU a eu lieu pour régler le problème, avec succès.

Dans ce processus, le node de l'IMU lit et publie donc les valeurs (corrigées) renvoyées par le driver, à savoir les vitesses angulaires ainsi que les accélérations linéaires selon chacun des axes. Néanmoins, il publie également un quaternion *(w, x, y, z)* issu de la fusion par le filtre de Madgwick que nous allons expliquer dès maintenant.

Filtre de Madgwick Intéressons-nous rapidement au filtre de Madgwick, algorithme essentiel de notre contrôle. En effet, il s'agit d'une fusion de capteurs, en l'occurrence du gyroscope et de l'accéléromètre, qui nous permet d'obtenir une estimation de l'état du robot sous forme de quaternion. Pour construire notre filtre, on se base sur le document original de S. O. H. Madgwick sur son filtre [5].

Pour réaliser ce filtre, on commence par une simple normalisation des accélérations selon chaque axe pour nous focaliser uniquement sur la direction du vecteur :

$$a_x = \frac{a_x}{\|\mathbf{a}\|}, \quad a_y = \frac{a_y}{\|\mathbf{a}\|}, \quad a_z = \frac{a_z}{\|\mathbf{a}\|} \tag{2}$$

Puis, on calcule $f(q, d, s)$ l'erreur entre la gravité attendue selon le quaternion actuel

et la gravité mesurée. Avec q le quaternion, d ici la direction de la gravité dans le repère terrestre, et s la mesure normalisée dans le repère du capteur, on a :

$$f(q, d, s) = q^* \otimes d \otimes q - s \tag{3}$$

Sans magnétomètre et avec l'accéléromètre uniquement pour cette étape, $d = (0, 0, 0, 1)$ et $s = (0, a_x, a_y, a_z)$, cette expression devient :

$$f(q, \hat{a}) = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} 2\left(q_2 q_4 - q_1 q_3\right) - a_x \\ 2\left(q_1 q_2 + q_3 q_4\right) - a_y \\ 1 - 2\left(q_2^2 + q_3^2\right) - a_z \end{bmatrix} \tag{4}$$

Puis, on procède à une descente de gradient qui permet de corriger le quaternion pour réduire l'erreur. On calcule le gradient de la fonction objectif ∇f par rapport aux composantes du quaternion, avec J le Jacobien de f. Dans notre cas, on obtient alors :

$$\nabla f = J^T(q) f(q, \hat{a}) = \begin{bmatrix} -2q_3 f_1 + 2q_2 f_2 \\ 2q_4 f_1 + 2q_1 f_2 - 4q_2 f_3 \\ -2q_1 f_1 + 2q_4 f_2 - 4q_3 f_3 \\ 2q_2 f_1 + 2q_3 f_2 \end{bmatrix} \tag{5}$$

Avant la fusion finale, on calcule la dérivée de l'orientation :

$$\dot{q}_\omega = \frac{1}{2} q \otimes (0, \omega) = \frac{1}{2} \begin{bmatrix} -q_2 g_x - q_3 g_y - q_4 g_z \\ q_1 g_x + q_3 g_z - q_4 g_y \\ q_1 g_y - q_2 g_z + q_4 g_x \\ q_1 g_z + q_2 g_y - q_3 g_x \end{bmatrix} \tag{6}$$

Enfin, place à la fusion et à l'intégration. On soustrait une fraction du gradient à la dérivée tout juste calculée, puis on intègre simplement sur le temps. D'abord, la fusion :

$$\dot{q} = \dot{q}_\omega - \beta \frac{\nabla f}{\|\nabla f\|} \tag{7}$$

Puis l'intégration et la dernière normalisation :

$$q_{k+1} = q_k + \dot{q}\, dt, q_{k+1} \leftarrow \frac{q_{k+1}}{\|q_{k+1}\|} \tag{8}$$

On obtient alors notre nouveau quaternion issu du filtre de Madgwick, qui sera publié par le node de l'IMU.

On notera tout de même que β a un rôle bien particulier dans ce filtre. Plus il est élevé,

plus la correction provenant de l'accéléromètre prend d'importance ; plus il est faible, plus l'intégration provenant du gyroscope en prend. C'est donc un juste milieu à trouver entre convergence rapide mais davantage sensible aux perturbations (β grand) et correction plus lente mais lissée et pouvant dériver (β faible). Avec plus de temps, une étude de l'influence de β sur le comportement de notre robot aurait pu être faite.

3.2.2 BME280

Le capteur BME280, pour la mesure de la pression, de la température et de l'humidité, communique via le bus I2C [12]. Un scan rapide du bus I2C permet de trouver l'adresse du capteur qui se trouve, dans notre cas, en 0x77. Ainsi, il est très facile de communiquer avec lui et de récupérer les trois informations importantes. On crée donc d'abord un driver puis un node pour publier ces données. Ce composant est sans doute le plus simple à implémenter et repose sur l'utilisation d'une bibliothèque `adafruit_bme280`.

Attention, pour la suite, le BME280 n'interviendra pas dans le contrôle, le capteur étant placé à l'intérieur du cylindre étanche. Un capteur *Bar High-Resolution Depth/Pressure Sensor* de chez BlueRobotics a été placé physiquement sur la tape arrière mais n'a pas encore été intégré électroniquement [16]. Toute référence ultérieure à un capteur de pression fera donc allusion à ce dernier, bien que non installé dans l'électronique.

3.2.3 Manette

Pour la manette, l'intégration dans l'architecture logicielle est particulière. En effet, elle nécessite un script Python sur le PC de l'utilisateur ainsi qu'une manette *Logitech F310 gamepad*. Ce code réalise la conversion très simple entre les boutons et les joysticks vers des consignes de mouvement sur notre robot 6 axes :

— Joystick gauche - vertical : translation avant-arrière Fx
— Joystick gauche - horizontal : translation latérale Fy
— Joystick droit - vertical : Pitch
— Joystick droit - horizontal : Yaw
— LB/RB : Roll
— LT/RT : Montée et descente Fz
— Bouton A : mode manuel

On obtient alors un vecteur de commande $[Fx, Fy, Fz, Roll, Pitch, Yaw, manual]$ qui est envoyé en UDP vers la Raspberry Pi 4 à environ 20 Hz sur le port 5005.

Sur le robot, on a donc un nœud conçu spécifiquement pour la manette qui lit, à la même fréquence, les données reçues sur le port 5005. Certaines sécurités, permettant d'éviter les accumulations inutiles, fluidifient ainsi le code et les valeurs reçues servent donc de consigne de contrôle. On obtient donc finalement 8 commandes moteurs en sortie. Nous verrons en détail le traitement des consignes de contrôle par la suite. Le bouton A, quant

à lui, renvoie 0 ou 1 et permet de changer en mode manuel comme nous allons le voir ci-dessous.

3.2.4 Moteurs

Sur la Raspberry Afin de contrôler les moteurs, on utilise une carte Arduino avec laquelle on communique en I2C. On crée donc un driver pour les moteurs capable, entre autres, d'envoyer une liste de 8 PWM à l'adresse 0x08, celle du microcontrôleur ATmega328P [14]. La méthode envoie donc un unique paquet I2C pour les huit moteurs et non un paquet par moteur comme nous avons pu le faire au début du projet, mais cela était source d'erreurs. Les valeurs envoyées sont avant cela bornées entre -100 et 100 puis encodées sur un octet entre 0 et 255. Les valeurs entre 0 et 100 sont donc les commandes positives, l'encodage conserve la valeur, tandis que celles entre 255 et 156 sont les commandes négatives : 246 correspond à une commande de -10 :

$$u_{\text{encod}} = \begin{cases} u, & \text{si } u \geq 0, \\ 256 + u, & \text{si } u < 0, \end{cases} \qquad u \in [-100, 100] \tag{9}$$

Quant au nœud, c'est très simple, il reçoit les commandes du contrôle autonome ou celles de la manette et envoie les commandes, à l'aide du driver, aux huit propulseurs. Pour choisir entre l'envoi des commandes autonomes ou de la manette, le nœud regarde le booléen du bouton A dans le callback de la manette : s'il vaut True, on passe en mode manette sans retour possible et on écoute les consignes de contrôle de la manette.

Certaines sécurités sont par ailleurs prévues comme l'arrêt complet des moteurs en l'absence de nouvelle commande après un certain temps ou en cas d'arrêt (Ctrl+C) par l'utilisateur.

Réception sur le microcontrôleur Évoquons très brièvement le microcontrôleur vers lequel les commandes sont donc envoyées et dont le code est accessible et modifiable. Sur ce code, on y reçoit constamment des trames I2C de 9 octets comprenant un en-tête puis les 8 valeurs pour la poussée des moteurs. On décode alors pour obtenir, cette fois-ci, des commandes entre -100 et 100. Une fonction permet alors de convertir ces valeurs en des impulsions de largeurs comprises entre 1100 μs et 1900 μs pour correspondre au protocole PWM classique des ESC. Le PCA9685 l'appliquera alors pour envoyer les signaux PWM de poussée aux ESC.

3.3 Communication entre les nœuds

Nous venons donc de voir ce que chacun des composants faisait individuellement. Désormais, intéressons-nous aux nœuds ROS 2 et spécifiquement aux publications et abonnements. On a donc :

— `imu_node` : publie sur `/imu`, message ROS `sensor_msgs/Imu`
— `pressure_node` : publie sur `/pressure_data`, message ROS `std_msgs/Float32MultiArray`
— `joystick_receiver_node` : publie sur `/cmd_joystick`, message ROS `std_msgs/Float32MultiArray`
— `control_node` : s'abonne à `/imu` et `/pressure_data`, calcule et publie 8 consignes sur `cmd_motor`
— `motor_node` : s'abonne à `/cmd_motor` et `/cmd_joystick`, puis pilote les moteurs

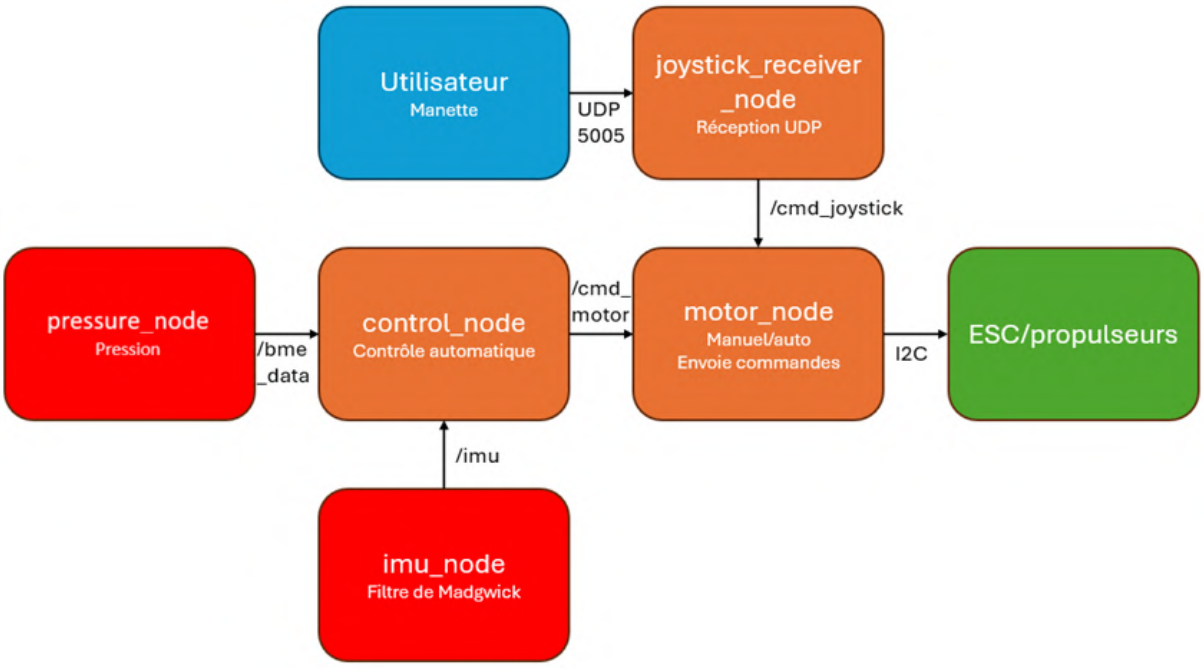


FIGURE 23 – Graphe de l'architecture logicielle ROS 2 du robot CUBE

On obtient le graphe final représentant l'architecture logicielle. En rouge les nœuds qui ne font que publier, correspondant donc aux 2 capteurs que nous utilisons (les sonars n'ayant pas été intégrés à temps). En orange, les nœuds impliqués directement dans l'envoi de commande. D'abord `control_node` qui réalise le contrôle automatique en calculant donc les forces et couples, traduits en commandes pour les 8 moteurs, à envoyer au robot pour réaliser la mission définie. Puis, `joystick_receiver` envoie également des commandes obtenues à partir des forces et couples commandés depuis les touches de la manette. Enfin `motor_node` traite ces données, en mode manuel ou non, et les envoie aux ESC.

4 Navigation du CUBE

4.1 Création de la simulation Unity

Désormais, nous allons nous tourner vers la navigation de notre robot dans un environnement sous-marin et rentrer dans le cœur du `control_node` de la figure 23. Pour cela, nous allons commencer par créer une simulation Unity qui servira de support pour le développement de notre contrôle. Celui-ci sera par ailleurs présenté dans un second temps avec un point théorique suivi de résultats dans la simulation. La création de cet environnement présente de nombreux avantages : elle facilite la création des méthodes de contrôle et d'un grand nombre de trajectoires tout en nous permettant de prendre certains risques et libertés que nous ne pourrions pas prendre en conditions réelles. L'intérêt réside également dans un réalisme fidèle permettant la mise en place de premiers réglages de coefficients et gains des algorithmes de contrôle. L'intégration de l'écosystème ROS 2 en simulation [9] a également montré son intérêt en facilitant le développement de l'architecture logicielle du robot de la partie 3, toutes deux étant donc similaires. Cette similitude est la raison pour laquelle nous ne présenterons pas l'architecture ROS 2 de la simulation dans ce rapport.

4.1.1 Physique

Commençons donc le développement de notre simulation par la création d'une physique. On commence donc par l'ajout d'un *GameObject* en forme de cube, auquel on ajoute la modélisation finale du robot obtenue dans la partie 2 (fig. 13) et qui servira pour le visuel 3D. Ce modèle permettra également de faciliter l'ajout des multiples capteurs et propulseurs par la suite. Le *GameObject* permet donc l'ajout d'un *RigidBody* et d'un *BoxCollider*. Le premier permet de mettre en place les propriétés physiques comme la masse ou l'inertie, tandis que le deuxième est uniquement utile pour la détection de contacts. C'est ainsi que nous allons développer la physique du robot, dotée de 3 composantes majeures, dont nous allons voir les détails.

Poussée d'Archimède La première composante de la physique du robot que nous avons dû ajouter est donc la poussée d'Archimède. La première étape est de diviser notre robot en plusieurs *BoxCollider* (fig. 24).

Ainsi, on calcule la poussée d'Archimède en prenant en compte le taux d'immersion du robot :

$$F_A = \rho \; V \; g \; \alpha(d) \tag{10}$$

Avec :

— F_A la force issue de la poussée d'Archimède
— ρ la masse volumique de l'eau

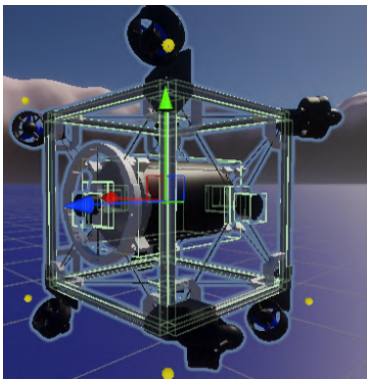


FIGURE 24 – Visualisation des différents *BoxCollider* pour le calcul de la poussée d'Archimède

— g la pesanteur
— V le volume total du robot
— $\alpha(d)$ la fraction du volume immergé

Le taux d'immersion $\alpha(d)$ est compris entre 0 et 1 et est calculé à partir de l'immersion du robot. Dans nos simulations et en pratique, le robot sera normalement amené à être complètement immergé et on aura donc constamment $\alpha(d) = 1$

À cela, on va ajouter un simple terme d'amortissement vertical F_B pour éviter les oscillations irréalistes : si le robot monte, F_B est dirigé vers le bas, s'il descend, F_B est dirigé vers le haut. La masse est prise de manière à obtenir une flottabilité neutre :

$$F_B = -\lambda_b \; m \; \dot{y} \tag{11}$$

Avec :

— F_B la force de l'amortissement vertical
— λ_b le coefficient d'amortissement vertical
— m la masse
— $\dot{y}$ la vitesse sur l'axe vertical

La résultante globale $F = F_A + F_B$ nous donne donc le terme qui définira le modèle de flottabilité utilisé dans notre simulation.

Frottements En plus de la poussée d'Archimède, il nous faut également ajouter des amortissements au robot. Ces derniers agissent donc comme frottements hydrodynamiques et correspondent à la deuxième composante à intégrer. Nous avons donc deux amortissements. D'abord, l'amortissement linéaire :

$$F_t = -\lambda_t.m.v \tag{12}$$

Avec :

— λ_t le coefficient d'amortissement en translation (à ajuster pour rendre le modèle plus fidèle au vrai robot)
— v la vitesse linéaire du robot

Puis l'amortissement angulaire :

$$\tau_a = -\lambda_r.m.\omega \tag{13}$$

Avec :

— λ_r le coefficient d'amortissement en rotation (à ajuster pour rendre le modèle plus fidèle au vrai robot)
— ω la vitesse angulaire du robot

Ces deux amortissements permettent donc de ralentir le robot dans l'eau et de représenter les différents frottements de manière simplifiée. Une modélisation dynamique complète du robot, telle que celle de Fossen, fait intervenir de nombreux termes plus complexes et nécessite une étude plus poussée [15]. Dans notre cas, les deux coefficients sont ajustables afin de reproduire au mieux le comportement du robot dans l'eau. Une nouvelle fois, il s'agit d'approximations importantes qui ne remplaceront pas une étude hydrodynamique complète du robot mais seront suffisantes pour le travail que nous avons à réaliser.

Actionneurs Enfin, il nous reste une dernière composante importante qui applique des forces à notre robot, il s'agit des propulseurs. En effet, chacun d'entre eux applique une force $F_i = T_i.d_i$ sur son sommet du cube en r_i, à partir d'une commande moteur T_i et d'une direction d_i.

L'utilisation de Unity comme environnement permet alors d'appliquer ces forces à leur position respective et d'obtenir les effets de translation et rotation générés par les moteurs. On a alors la force et le couple totaux des huit propulseurs :

$$\mathbf{F}_{\text{total}} = \sum_{i=1}^{8} T_i \mathbf{d}_i, \quad \boldsymbol{\tau}_{\text{total}} = \sum_{i=1}^{8} T_i (\mathbf{r}_i \times \mathbf{d}_i) \tag{14}$$

4.1.2 Capteurs

Présentons désormais rapidement les trois capteurs dont nous avons besoin dans le contrôle, à savoir l'IMU, les sonars et le baromètre. On notera néanmoins que la caméra a été implémentée dans la simulation mais n'est jamais utilisée concrètement, elle ne sera donc pas présentée ici.

Capteur inertiel Ainsi, le premier capteur que nous allons présenter est l'IMU basée sur le modèle ICM-20948 que nous utilisons sur le robot [10]. Celui-ci nous permet ici d'obtenir une mesure de l'orientation en quaternion $q = (w, x, y, z)$. En pratique, ce quaternion est

obtenu à partir du filtre de Madgwick [5]. Nous obtenons également la vitesse angulaire dans le repère local ainsi que la mesure de l'accélération linéaire. Tout cela est possible à partir du *rigidbody* de la simulation Unity.

Baromètre Pour le baromètre, second capteur de notre simulation, l'implémentation est très simple et nous donne la pression relative $P = \rho.g.d.$

Echosondeurs Dans Unity, pour les 6 échosondeurs au centre de chaque face, on utilise un *Raycast* qui nous retourne une valeur entre 0.2 et 20 m, -1 sinon. On obtient alors les valeurs des six échosondeurs en un seul message pour simplifier l'utilisation.

4.2 Contrôle

Intéressons-nous désormais au contrôle que nous appliquerons à notre robot Cube. L'idée que nous évoquons depuis le début est donc de pouvoir translater et tourner selon chacun des axes x,y et z. Comme nous avons pu le dire, la configuration que nous avons mise en place permet au robot d'être sur-actionné mais surtout holonome.

Pour mettre en place un contrôle sur ce robot, on se base sur les travaux de L.Lapierre, R.Zapata, P.Lepinay et B.Ropars dans *Karst Exploration : Unconstrained Attitude Dynamic Control for an AUV* [2] reposant sur une structure de contrôle NGC-A (*Navigation, Guidance, Control and Actuation*) comprenant l'estimation de l'état, la génération de la trajectoire, le calcul des efforts nécessaires et enfin l'allocation. Nous verrons que notre structure reprend ce format et permet de commander efficacement le robot. Nous ne parlerons bien évidemment pas de tout ce qui a pu être présenté dans le document, mais uniquement de ce qui nous a été utile.

4.2.1 Représentations et erreurs en quaternion

Le premier détail important est l'utilisation d'une représentation du robot en quaternions. En effet, la représentation classique que l'on fait des robots se base habituellement sur les angles d'Euler (ϕ, θ, ψ) pour le roulis, le tangage et le lacet (ou *roll*, *pitch*, *yaw*). Celle-ci présente de nombreux avantages comme sa facilité de compréhension et sa représentation directe du comportement du robot. Néanmoins, elle présente deux problèmes majeurs que nous avons pu rencontrer dans un premier temps en voulant mettre en place un contrôle similaire avec les angles d'Euler.

D'abord, le blocage de Cardan qui s'explique par une perte de degré de liberté dans certaines configurations. Pour expliquer brièvement cela, prenons la matrice de rotation complète issue de celle de chaque axe avec leur propre angle : de roulis $R_x(\phi)$, tangage $R_y(\theta)$ ou lacet $R_z(\psi)$. Cette matrice s'écrit alors :

$$R = R_z(\psi).R_y(\theta).R_x(\phi) \tag{15}$$

Le blocage de Cardan intervient alors lorsque, par exemple, $\theta = \pi/2$. La matrice R devient :

$$R = \begin{bmatrix} 0 & \sin(\phi - \psi) & \cos(\phi - \psi) \\ 0 & \cos(\phi - \psi) & -\sin(\phi - \psi) \\ -1 & 0 & 0 \end{bmatrix} \tag{16}$$

On remarque alors immédiatement que la matrice R dépend désormais de la différence entre le roulis et le lacet, et non plus de chacun d'entre eux séparément. Une augmentation de α degrés pour chacun des deux angles nous donne la même matrice de rotation R. C'est alors cette perte de degré de liberté dans la représentation, causant notamment plusieurs triplets d'angles à représenter la même orientation, qui caractérise le blocage de Cardan.

Dans le même style, une même rotation peut être obtenue par plusieurs triplets d'angles d'Euler, le suivi d'orientation est alors rendu difficile et c'est pour cela que nous privilégions les quaternions représentés de la manière suivante :

$$q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}, q = q_0 + q_1 i + q_2 j + q_3 k \tag{17}$$

Cette représentation nous débarrasse du blocage de Cardan que nous venons d'évoquer, et est idéale dans le cadre de notre robot sous-marin holonome, avec pour contrainte $||q|| = 1$. q et $-q$ représentant la même orientation, la contrainte de la partie scalaire positive, $q_0 \geq 0$, nous permet de lever l'ambiguïté. La représentation d'une rotation avec un angle β autour d'un axe unitaire $n = (n_x, n_y, n_z)$ est alors très simple et se fait de la manière suivante :

$$q = \begin{bmatrix} cos(\beta/2) \\ n_x.sin(\beta/2) \\ n_y.sin(\beta/2) \\ n_z.sin(\beta/2) \end{bmatrix} \tag{18}$$

Ainsi, à partir de cette représentation, on peut très facilement définir un quaternion d'erreur q_e entre l'actuel q et le désiré q_d. Ce quaternion est défini par la formule suivante où, q étant unitaire, q^{-1} est le conjugué [7] :

$$q_e = q^{-1} \otimes q_d \tag{19}$$

q_e représente donc la rotation relative à réaliser pour passer de q à q_d.

4.2.2 Loi de commande

Le contrôle dynamique que nous mettons en place repose sur une approche de Lyapunov-backstepping. On définit donc une loi de commande permettant au quaternion d'erreur q_e de converger vers le quaternion identité $[1, 0, 0, 0]^T$. La première structure obtenue est la suivante et provient de la proposition 1 du papier :

$$W_c = (q_e \otimes W_d \otimes q_e^*)_{1:3} + K_1 \tilde{q}_e \tag{20}$$

Avec :

— $W_d = 2.q_d^* \otimes \dot{q_d}$ le quaternion associé à la vitesse angulaire de la trajectoire référentielle

— $\tilde{q}_e$ la partie vectorielle de l'erreur et K_1 le gain

Ainsi, ce terme permet de faire converger progressivement q vers q_d en réduisant donc l'erreur. Le premier terme dit feedforward permet d'obtenir la vitesse angulaire désirée W_d dans le repère du robot à partir d'une rotation par q_e. Le second terme est le terme de correction où $\tilde{q}_e$ est la partie vectorielle de q_e. Plus l'erreur est grande, plus la correction l'est aussi, et inversement. Nous appliquerons cette loi par la suite.

4.2.3 Loi de commande dynamique

En considérant la dynamique réelle du robot, à savoir $\tau = J\dot{\omega} + f(\omega, v, \eta)$ avec J la matrice d'inertie, ω la vitesse angulaire et f comprenant plusieurs effets dynamiques. Dans notre cas, avec un modèle simplifié, on a $J = I_3$ et $f = 0$. On obtient la loi de commande suivante :

$$\Gamma_d = \dot{W}_c + K_2(W_c - \omega) + K_3 \tilde{q}_e \tag{21}$$

Le premier terme est donc un feedforward dynamique, anticipant les variations de la consigne de vitesse. Le second joue le rôle d'amortisseur sur l'erreur de vitesse angulaire. Enfin, le troisième terme introduit une correction sur l'erreur d'orientation.

4.2.4 Allocation de commandes

À partir de la loi de commande que nous venons de voir, nous obtenons donc un torseur de commande $[F_d, \Gamma_d]^T$. On construit ensuite une matrice d'allocation B en utilisant les positions r_i et directions d_i des propulseurs :

$$B = \begin{bmatrix} d_1 \ ... \ d_8 \\ r_1 \times d_1 \ ... \ r_8 \times d_8 \end{bmatrix} \tag{22}$$

Sur cette matrice, on obtient alors les directions de poussée des 8 propulseurs sur les 3 premières lignes puis leur contribution au moment sur les 3 dernières. Le lien entre le

torseur de commande et les poussées individuelles T à appliquer est donc obtenu avec la pseudo-inverse de Moore-Penrose [6] :

$$T = B^{+} \begin{bmatrix} F_d \\ \Gamma_d \end{bmatrix} \tag{23}$$

Avec, pour $rang(B) = 6$:

$$B^{+} = B^{T}(BB^{T})^{-1} \tag{24}$$

L'avantage du sur-actionnement intervient alors car la pseudo-inverse permet d'obtenir la solution (parmi une infinité) permettant de répartir justement les efforts sur les différents propulseurs.

Une dernière étape de l'allocation est la normalisation des commandes. En effet, on divise simplement les valeurs de T par le maximum de ses composantes, ce qui n'affecte donc pas les directions, lorsque ce maximum dépasse 1 :

$$T \leftarrow \frac{T}{max_i|T_i|} \tag{25}$$

4.2.5 Dead-reckoning

Une des étapes importantes de notre contrôle est le dead-reckoning. On utilise alors les mesures de l'accéléromètre et du baromètre afin de calculer la position et la vitesse du robot [17]. On soustrait d'abord la gravité g_{world}, exprimée dans le repère du robot, à l'accélération mesurée du robot en utilisant le quaternion q de l'état actuel du robot :

$$a_{body} = a_{IMU} - (q^{*} \otimes g_{world} \otimes q)_{1:3} \tag{26}$$

Puis, afin de pouvoir intégrer, on passe notre nouvelle accélération dans le repère monde :

$$a_{world} = (q \otimes [0, a_{body}^{T}] \otimes q^{*})_{1:3} \tag{27}$$

Enfin, on termine le dead-reckoning par l'intégration nous donnant la vitesse et la position estimées de notre robot :

$$\begin{aligned} v_{k+1} &= v_k + a_{world}.\Delta t \\ p_{k+1} &= p_k + v_k.\Delta t \end{aligned} \tag{28}$$

On notera néanmoins que la présence du baromètre sur le robot nous permet d'obtenir

des valeurs plus précises et robustes que celles du dead-reckoning pour la profondeur. Pour l'axe vertical, on traite la donnée différemment :

$$\begin{aligned} v_{ver} &= \frac{d_k - d_{k-1}}{\Delta t} \\ p_{ver} &= d_k \end{aligned} \tag{29}$$

4.2.6 Suivi de chemin

Enfin, nous arrivons dans la dernière partie de notre contrôle avec ce qui constitue l'élément principal de notre mission : le suivi de chemin. On construit notre chemin comme un ensemble de N points $X(s)$ d'abscisse curviligne s. Dans notre cas, on construit donc un chemin à la forme hélicoïdale, donc une forme de cercle dont la profondeur augmente avec l'avancement du chemin.

A partir de ce chemin tout juste créé, on calcule d'abord l'erreur de position dans le repère du chemin. Pour cela, on utilise un point de référence mobile sur le chemin, le lièvre, avec une position X_s et une orientation tangente au chemin Q_s. On obtient alors l'écart de position entre le robot (position p) et ce point dans le repère local du chemin :

$$\begin{bmatrix} 0 \\ x_1 \\ y_1 \\ z_1 \end{bmatrix} = Q_s^* \otimes \begin{bmatrix} 0 \\ p - X_s \end{bmatrix} \otimes Q_s \tag{30}$$

On utilise alors ces erreurs pour le calcul de la loi de guidage et de l'angle d'approche. On obtient deux angles de correction :

$$\delta_y = -arctan(K_y y_1),\ \delta_z = +arctan(K_z z_1) \tag{31}$$

Qu'on transforme en rotation quaternion autour d'un axe du chemin :

$$Q_{\delta_y} = \left[cos(\tfrac{\delta_y}{2})\ sin(\tfrac{\delta_y}{2}) n_y^T\right]^T,\ Q_{\delta_z} = \left[cos(\tfrac{\delta_z}{2})\ sin(\tfrac{\delta_z}{2}) n_z^T\right]^T \tag{32}$$

Puis, on obtient la représentation de l'écart d'attitude, le quaternion de correction, à ajouter à la tangente du chemin pour que le robot vise le chemin :

$$Q_\Delta = Q_{\delta_y} \otimes Q_{\delta_z} \tag{33}$$

On définit alors l'orientation de guidage Q_D, obtenue à partir de l'application de Q_Δ à l'orientation tangentielle du chemin Q_s.

$$Q_D = Q_\Delta \otimes Q_s \tag{34}$$

Enfin, on construit un vecteur vitesse désirée v_d du robot qui va nous permettre d'obtenir une accélération linéaire désirée et donc la force à appliquer. On obtient d'abord le quaternion V_d à partir d'une vitesse de référence v_{ref} définie selon l'axe longitudinal du robot :

$$V_d = Q_D \otimes \begin{bmatrix} 0 \\ v_{ref} \\ 0 \\ 0 \end{bmatrix} \otimes Q_D^* \tag{35}$$

Le vecteur vitesse désirée est donc la partie vectorielle de V_d. Il permet alors de converger vers le chemin ou, lorsque le robot est dessus, de suivre la tangente. On obtient désormais la commande de l'accélération linéaire v_c à partir de la vitesse du robot v et de v_d :

$$\dot{v_c} = \dot{v_d} + K_v(v_d - v) \tag{36}$$

Avec :

— v_d le vecteur vitesse désirée
— v le vecteur vitesse du robot
— K_v le gain de correction

La force désirée F_d qui servira de commande est alors simplement obtenue à partir de :

$$F_d = m\dot{v_c} \tag{37}$$

Enfin, il nous reste à trouver le couple désiré Γ_d. Dans l'exemple que nous utilisons, nous choisissons d'orienter le robot vers le centre de l'hélicoïde. Assez simplement, on définit donc le vecteur reliant la position du robot p à celle du centre de l'hélice p_h :

$$d_c = p_h - p,\ \hat{d_c} = \frac{d_c}{||d_c||} \tag{38}$$

Le quaternion d'erreur q_e est donc obtenu à partir du quaternion cible q_{cible} correspondant à la rotation permettant l'alignement de l'axe du robot avec $\hat{d_c}$, tout en maintenant le robot à l'horizontale :

$$q_e = q^{-1} \otimes q_{cible} \tag{39}$$

Dans un premier temps, la loi de commande (4.2.2) vue plus tôt nous permet d'obtenir W_c, puis on obtient Γ_d à partir de la loi de commande dynamique (4.2.3). On a alors le torseur $[F_d, \Gamma_d]^T$ et ce dernier est donc envoyé en tant que commande.

4.3 Résultats

On arrive alors à la dernière partie de ce rapport avec les résultats, que ce soit en simulation ou en bassin. Nous verrons donc d'abord les résultats sur Unity puis évoquerons ce que nous avons pu observer en bassin, avec le peu de temps que nous avions.

4.3.1 Simulation

Dans cette sous-partie, focalisons-nous d'abord sur les résultats en simulation. On passera outre le bon fonctionnement des différents capteurs qui ont été vérifiés individuellement et ont tous montré des résultats cohérents. Portons donc plutôt notre intérêt sur la mission que l'on s'était fixée, à savoir le suivi de chemin hélicoïdal avec orientation précise du robot, dans notre cas vers le centre de l'hélice. Toujours sur notre simulation Unity, on réalise donc des essais nous permettant d'obtenir des résultats convaincants à partir des méthodes de contrôle que nous venons d'évoquer. On observe alors le comportement du robot :

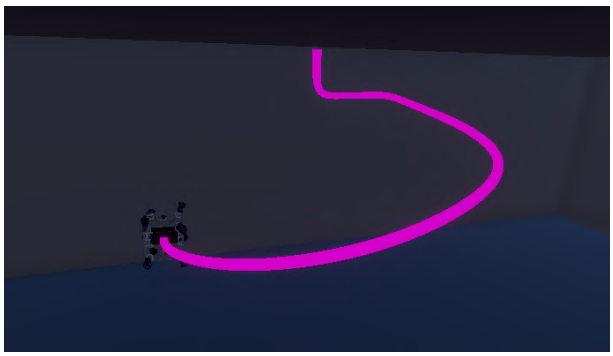


FIGURE 25 – Visualisation de la mission de suivi de chemin hélicoïdal sur la simulation Unity

Ce qu'on peut observer sur cette figure, c'est la présence de notre robot dans un bassin réalisant la mission fixée. Le trait rose représente le chemin parcouru par le robot depuis le début de son parcours. On constate donc une première forme hélicoïdale pertinente, mais il convient tout de même de comparer cette trajectoire au chemin cible. Pour cela, on compare la trajectoire du robot à la trajectoire désirée à partir de différents points de vue (fig. 26).

Au niveau du comportement du robot, on constate donc une première phase que l'on pourrait qualifier de transition entre sa position de départ et le suivi du chemin. Le robot

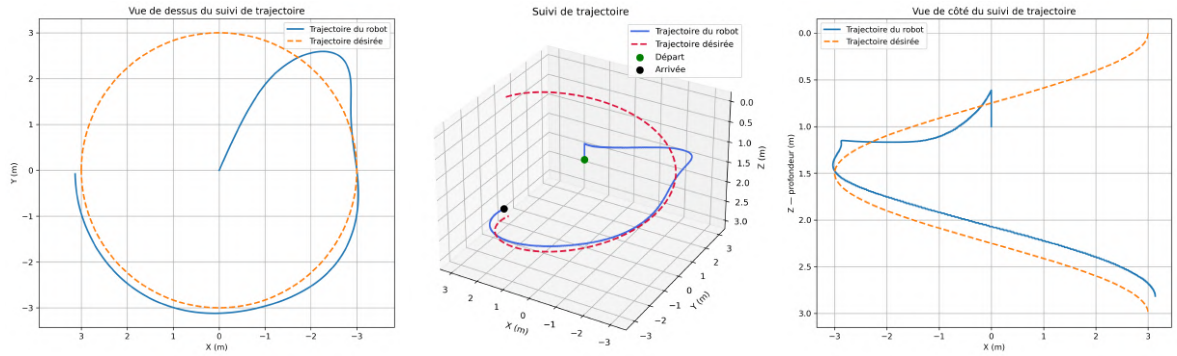


FIGURE 26 – Visualisation de la trajectoire du robot par rapport à la trajectoire désirée lors de la mission de suivi de chemin hélicoïdal

se déplace donc d'abord vers le chemin puis, dans un second temps, cherche à se stabiliser autour de celui-ci.

Ce que l'on constate alors immédiatement, c'est la difficulté du robot à suivre la trajectoire en profondeur. On remarque cependant que la trajectoire selon le plan horizontal est très bonne et l'erreur se réduit grandement avec le temps. Néanmoins, la présence d'un retard de suivi de la profondeur semble persister avec les multiples essais. L'hypothèse la plus probable pourrait être simplement la limitation des propulseurs du robot. Ceux-ci sont limités en puissance et doivent à la fois produire les mouvements de translation et de rotation. L'augmentation de profondeur étant le mouvement le plus difficile car opposé à la poussée d'Archimède, il se pourrait donc que le mouvement vertical soit trop faible par rapport à ce qui est nécessaire. Une augmentation manuelle du gain lié à ce mouvement cause d'ailleurs des dérives des autres mouvements. En effet, en augmentant le gain, la profondeur est mieux suivie jusqu'à un certain point où le suivi hélicoïdal se dégrade.

Afin de mieux visualiser cette erreur, intéressons-nous à l'erreur globale, comprenant ou non la profondeur, ainsi qu'aux erreurs selon chacun des axes :

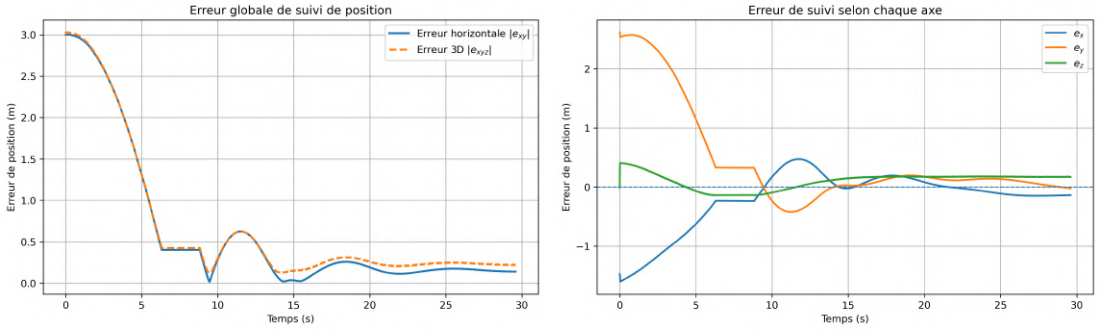


FIGURE 27 – Représentation de l'erreur globale ainsi que des erreurs particulières à chaque axe

Sur le graphe de l'erreur globale, on constate tout de même que l'erreur 3D pourrait avoir une tendance à se stabiliser, tout comme l'erreur sur le plan horizontal. Dans la même logique, les graphes des erreurs de position selon les axes expliquent les évolutions globales :

l'erreur verticale ne semble pas converger vers 0 mais plutôt se stabiliser autour d'une certaine valeur, laissant une erreur persistante, tandis que les erreurs du plan horizontal ont une tendance à osciller autour de 0. Sur un autre aspect, on constate toujours bien évidemment la phase de transition puis de stabilisation. Une meilleure gestion des gains pourrait évidemment nous donner de meilleurs résultats.

Un autre aspect que nous n'avons pas évoqué encore est l'orientation du robot vers sa cible. En effet, dans cette situation, nous avons choisi de cibler le centre de l'hélicoïde en gardant le robot à l'horizontale. Intéressons-nous donc à l'erreur d'orientation vers ce point cible. Celle-ci est calculée à partir du quaternion d'erreur et avec extraction de l'angle (eq. 18) :

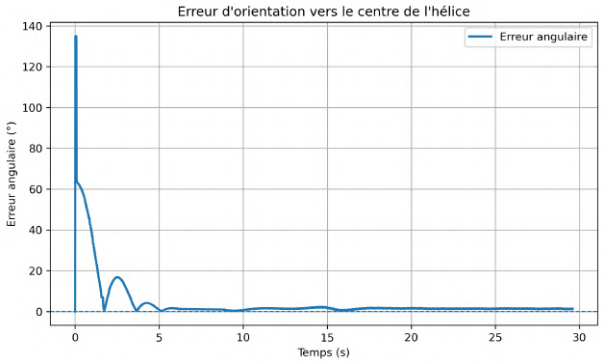


FIGURE 28 – Erreur d'orientation du robot dans le cadre du suivi de chemin hélicoïdal

On constate une nouvelle fois une première phase de transition où le robot semble se "mettre en position", puis il se stabilise et l'erreur reste stable. On constate donc que l'erreur d'orientation devient rapidement faible. En moins de 8 secondes, le robot a trouvé une orientation convenable. Une fois de plus, des gains différents pourraient changer les comportements, évitant les quelques oscillations ou accélérant la convergence. On note néanmoins la grande efficacité, et sans singularité, que semble nous proposer l'utilisation de quaternions pour le suivi d'orientation.

Enfin, nous terminons cette partie sur les résultats en simulation avec les commandes envoyées aux 8 propulseurs :

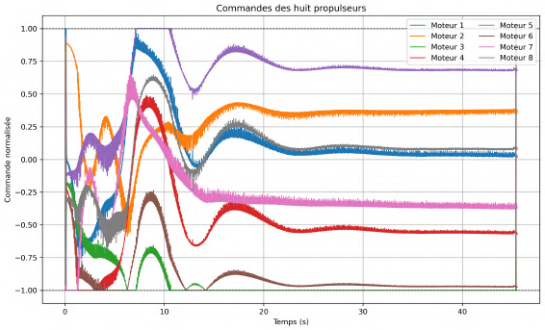


FIGURE 29 – Commandes envoyées aux 8 propulseurs dans le temps

On constate donc directement et assez logiquement que les commandes ne dépassent

pas 100%. Certains moteurs atteignent temporairement la valeur maximale sans que cela ne pose de problème dans le mouvement du robot. On remarque bien la répartition des commandes entre les différents moteurs. Le sur-actionnement du robot est donc bien exploité pour réaliser les mouvements de translation et de rotation en simultané.

On constate néanmoins de faibles oscillations, plus marquées sur certains moteurs que d'autres, présentes en continu autour d'une valeur moyenne. Cela pourrait être simplement dû aux nouveaux calculs des commandes à chaque itération qui cherchent à corriger ce qui pourrait être de très faibles oscillations du robot. L'utilisation de la pseudo-inverse de Moore-Penrose, dont nous avons parlé plus tôt, pourrait aussi en être à l'origine : de faibles mouvements du CUBE pourraient provoquer des changements de commande importants. Ces oscillations ne sont pas très problématiques à ce stade pour la simulation, mais le seront sans doute pour le robot lors de tests en bassin. En effet, l'électronique pourrait ne pas supporter une telle sollicitation et créer des bugs ou même des casses. Dans de futurs travaux, un filtrage pourrait être implémenté pour limiter ces variations.

4.3.2 Résultats expérimentaux

Après avoir réalisé les tests en simulation, nous avons donc fait face à une longue phase de conception mécanique et électronique du robot CUBE. Durant cette période, nous avons rencontré de nombreux problèmes ralentissant le projet. Après des problèmes de conception de pièces puis multiples casses sur l'électronique lors du montage final, nous sommes arrivés à la mise à l'eau du robot :

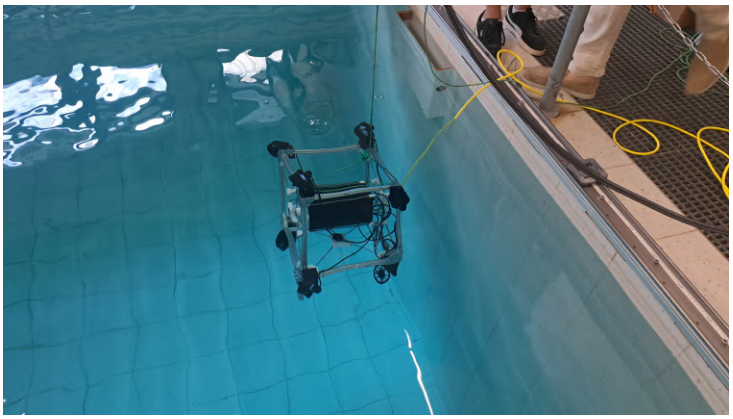


FIGURE 30 – Première mise à l'eau du robot dans le petit bassin du Lab-STICC

Nous avons alors pu vérifier l'étanchéité du robot ainsi que sa flottabilité. Le premier point fut un succès tandis que le deuxième montra une flottabilité négative alors que nous cherchions à avoir un robot neutre. Ce point n'a pas pu être amélioré à temps. L'équilibrage pourrait également être perfectionné au vu du poids du tube relativement important et positionné légèrement en avant. Cela ne semblait tout de même pas être un problème majeur de ce robot.

Après cela, nous sommes donc passés à la phase de transmission de consignes au robot de la même manière qu'expliquée dans la partie 3.2.3 sur la manette. On a donc des

consignes sur les 6 degrés de liberté converties en force et couple, enfin répartis entre les propulseurs comme vu précédemment. Le comportement observé lors de ces essais s'est révélé très satisfaisant. Les mouvements sont cohérents et correspondent exactement aux commandes envoyées depuis la manette. L'ensemble des mouvements a pu être testé ainsi que différentes combinaisons. Cela nous a permis de valider différents aspects de notre architecture, à commencer par la communication entre l'utilisateur et le robot, puis la bonne structure de la matrice d'allocation, du repère du robot, ainsi que les sens des moteurs. Au global, la structure de l'architecture ROS 2 semble bien fonctionner, bien que certains capteurs, inutiles pour le moment, aient été désactivés pour s'assurer du bon fonctionnement global des essais.

Durant ces essais, nous n'avons donc pas pu parvenir à un pilotage automatique, mais cette étape à la manette apparaît comme essentielle au vu des travaux de contrôle qu'il reste désormais à venir. En effet, les deux contrôles utilisent le même repère ainsi que la même matrice d'allocation. Comme vu dans la présentation de l'architecture logicielle, la structure en mode manuel comme automatique reste presque identique.

Les tests en bassin ont également pu mettre en lumière certains problèmes. Beaucoup d'entre eux ont été résolus, comme la gestion des crashs, initialement chaotiques et trop fréquents. D'autres demanderont une attention dans les travaux futurs comme la perte de communication parfois aléatoire mais dont la fréquence a également été grandement réduite. On notera comme problème à surveiller la puissance envoyée aux propulseurs à long terme, qui est à gérer afin d'éviter une surchauffe ou une explosion d'une piste de la carte PCB, comme cela a pu nous arriver sur la fin du projet.

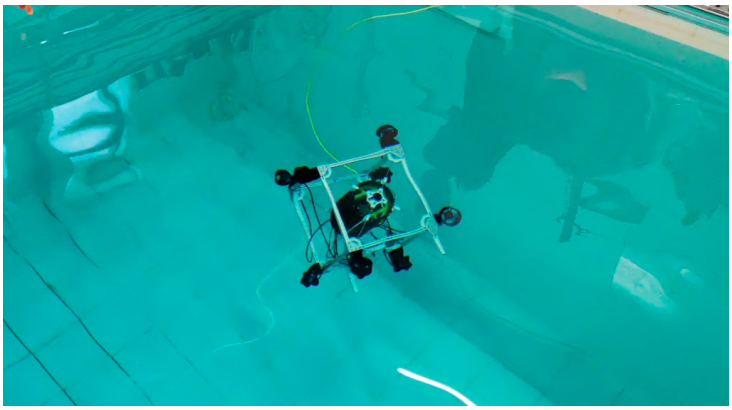


FIGURE 31 – Essais en bassin avec commande à la manette depuis le poste utilisateur

5 Conclusion

À travers ce rapport, nous avons présenté l'ensemble des travaux réalisés sur le robot CUBE durant ce projet de fin d'études. Son développement a donc commencé par une grande partie de conception mécanique, d'abord en CAO puis en impression 3D, qui a permis d'aboutir à un montage cohérent du robot. Des pièces pour l'extérieur du cylindre ont alors été conçues mais n'ont malheureusement pas pu être usinées. D'autres pour la structure interne du tube ont bien été imprimées, permettant le montage de l'électronique dans le robot. En parallèle, nous avons procédé à la construction d'une simulation sur UNITY. Nous avons alors intégré une dynamique au robot ainsi que différents frottements afin de reproduire au mieux le comportement réel du robot. En utilisant les travaux de recherche de Huu Tho Dang, nous pouvions également intégrer la configuration optimale *C2* des huit moteurs que l'on retrouve dans sa thèse. Cette simulation permet alors de tester le contrôle orienté quaternions. Cette loi de commande spécifique permet de s'affranchir, par exemple, du blocage de Cardan et donc d'obtenir une certaine robustesse. Après l'intégration de ROS 2 intégrant la simulation UNITY, le contrôle a alors été testé et nous a donné des résultats très encourageants, bien que présentant quelques petits défauts comme le suivi de profondeur. Enfin, une architecture logicielle ROS 2 plus complexe a été développée sur la Raspberry PI 4 du robot et nous a permis d'y ajouter la chaîne de commande à la manette et automatique. Seul le contrôle à la manette a été testé et nous a montré des résultats très encourageants pour quiconque prendra la relève sur ce projet.

Ainsi, à l'issue de ce projet, nous avons obtenu uniquement des résultats de contrôle du robot à la manette. Pour la suite, la principale perspective d'amélioration sera donc de continuer ce développement pour arriver à un contrôle automatique utilisant la base construite avec la simulation. Le code associé est déjà présent sur le robot mais n'a été testé que hors de l'eau. Il conviendrait donc de vérifier le bon fonctionnement de ce code en portant une attention particulière sur les changements de repère qui pourraient être la source de nombreux problèmes, comme ce fut le cas jusqu'à présent. L'intégration électronique dans le code des échosondeurs et du capteur de pression sera également un passage obligatoire. D'autres améliorations sont évidemment possibles avec l'usinage et l'intégration des pièces prévues pour l'extérieur du robot. Il faudrait également porter un regard attentif sur la gestion des crashs en réduisant leur fréquence et en minimisant les conséquences.

Sur un aspect plus personnel, ce projet m'aura permis de développer des compétences dans de multiples domaines. D'abord en conception mécanique, avec la création de multiples pièces suivie de leur fabrication et de leur intégration. Ensuite en électronique avec une grosse partie consacrée à la soudure de composants et à l'intégration à la carte PCB ainsi qu'à la réparation de différents éléments. Nombreuses sont également les compétences que j'ai pu enrichir en développement de simulation, bien que simple pour ce projet, et particulièrement aussi en commandes des systèmes avec l'intégration du contrôle basé

quaternions. Pour terminer, j'ai surtout pu me renforcer en architecture logicielle ROS 2 avec le développement, de zéro, d'une première architecture pour la simulation UNITY et d'une deuxième intégrée au robot.

Au niveau de l'organisation du travail, par rapport au diagramme de Gantt initialement imaginé, beaucoup de choses ont changé (diagramme de Gantt final, fig. 33). La partie modélisation et intégration des pièces a pris la majorité de mon temps avec aussi l'intégration électronique comprenant l'ajout de capteurs, les réparations et les différentes modifications ou améliorations. Inversement, les parties simulation, contrôle et architecture logicielle ont rapidement avancé et ont été terminées rapidement. Que ces parties aient avancé si vite et que le projet ait perdu autant de temps sur la construction du robot restera un regret de ces mois de stage.

Remerciements

Je conclurai ce rapport en remerciant d'abord Lionel Lapierre de m'avoir donné l'opportunité de travailler sur un tel projet, pour son encadrement et son soutien tout en me laissant la liberté de m'exprimer et de prendre des initiatives. Je tiens également à remercier Titouan Bélier et Gabriel Betton, tous deux encadrants du projet et dont la présence et les conseils ont été essentiels pour l'avancée du projet. Je remercie enfin Maël Godard et Antoine Morvan pour leur aide précieuse et leur disponibilité au quotidien.

6 Annexes

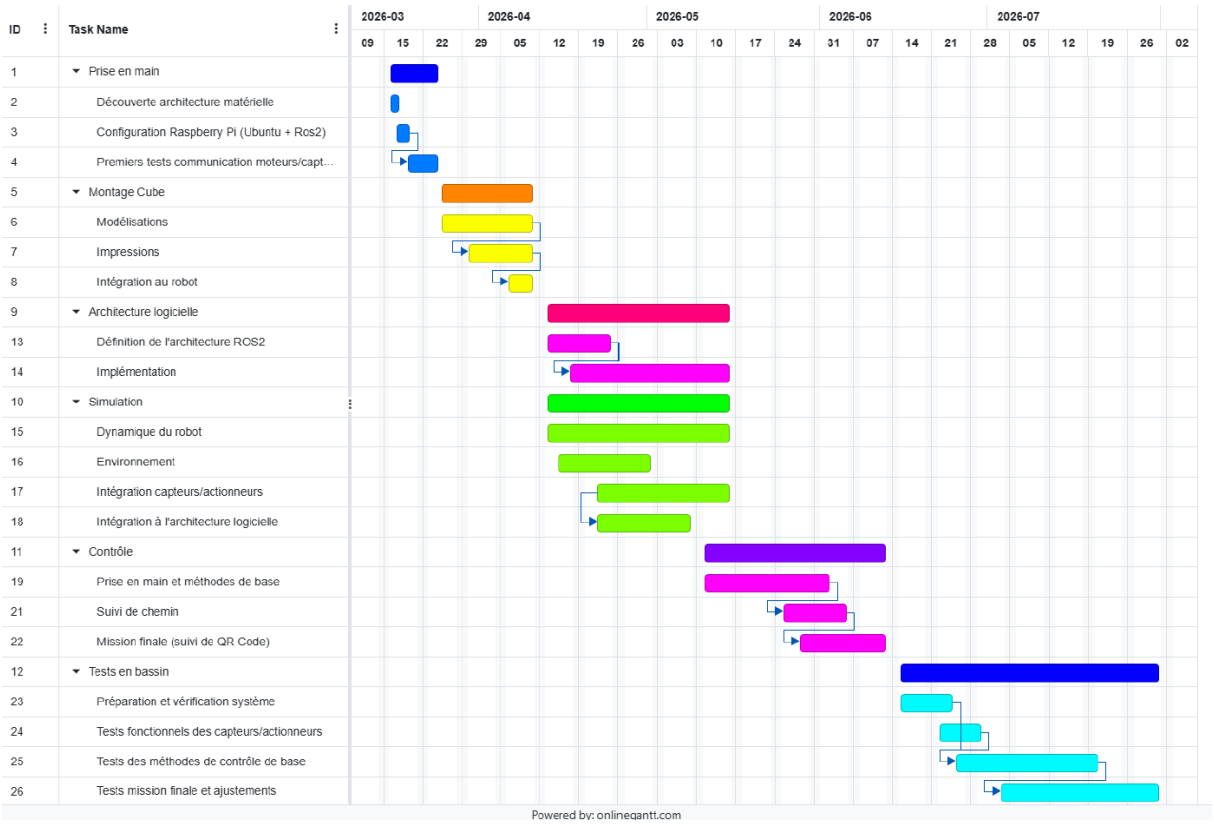


FIGURE 32 – Diagramme de Gantt prévisionnel du projet

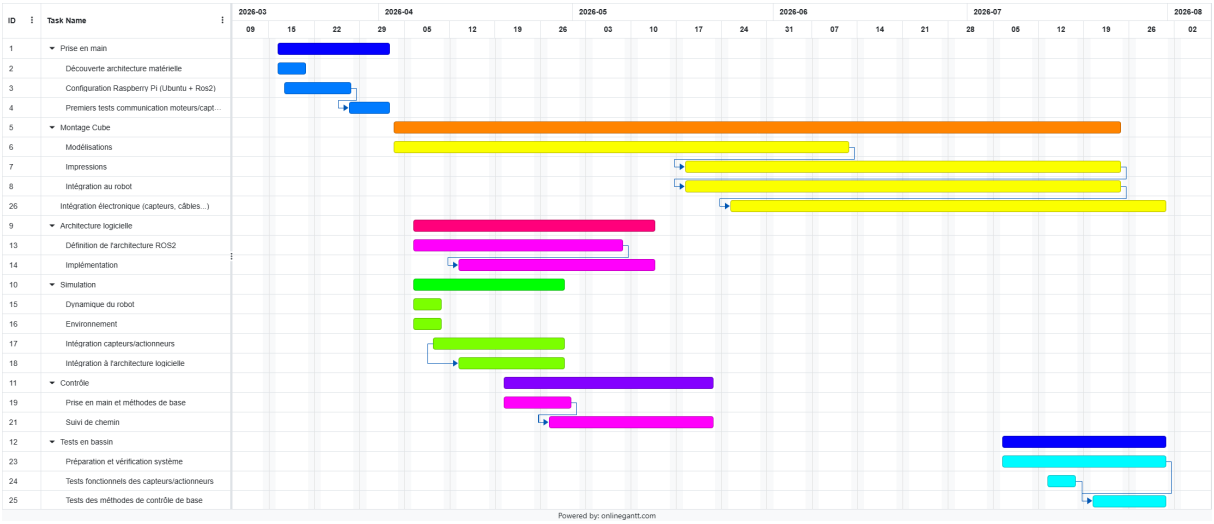


FIGURE 33 – Diagramme de Gantt final du projet

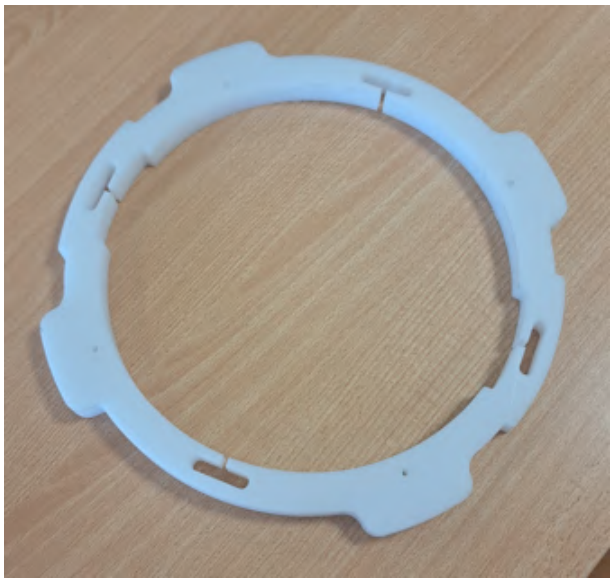


FIGURE 35 – Anneau partiellement usiné

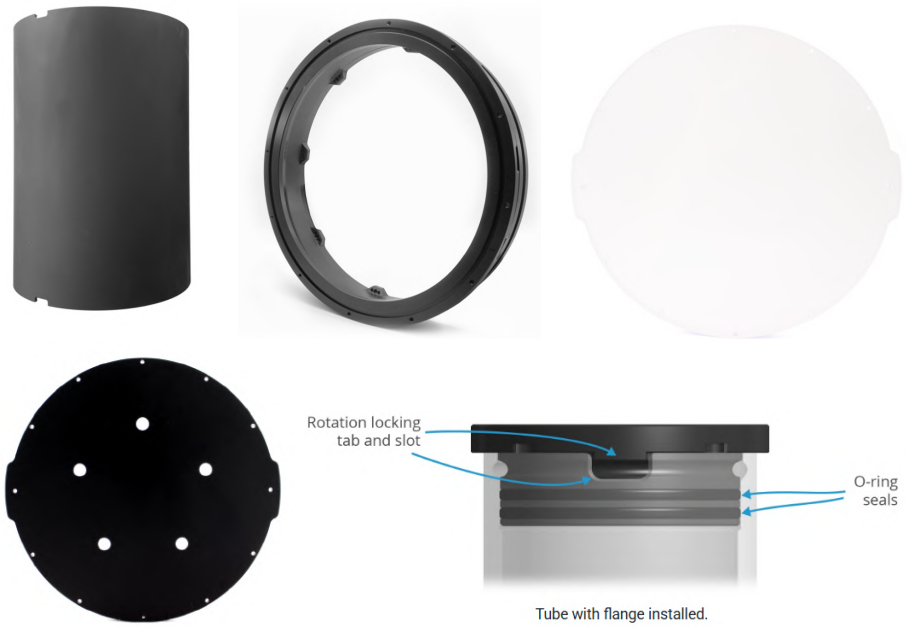
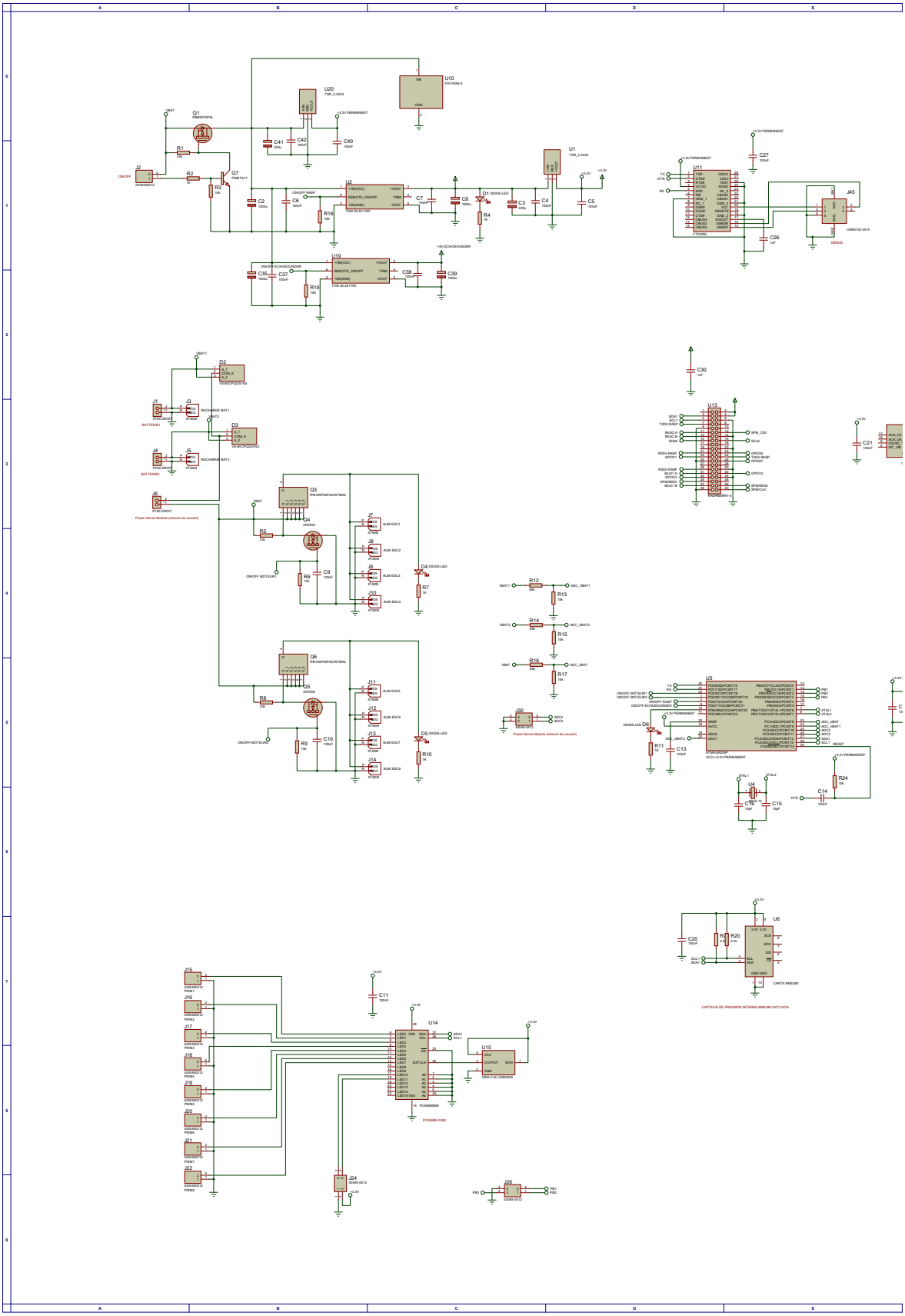


FIGURE 34 – Pièces BlueRobotics pour le robot CUBE (illustrations site BlueRobotics)

GIT

Il existe un GIT contenant la quasi-intégralité des documents du projet : l'ensemble des fichiers de l'architecture logicielle du robot, les documents de l'architecture électronique, les fichiers de la CAO, le manuel d'utilisation, quelques médias et les fichiers de la simulation Unity.

Pour accéder à ce git, merci de m'envoyer un mail à `thomas.aguirre@ensta.fr`.



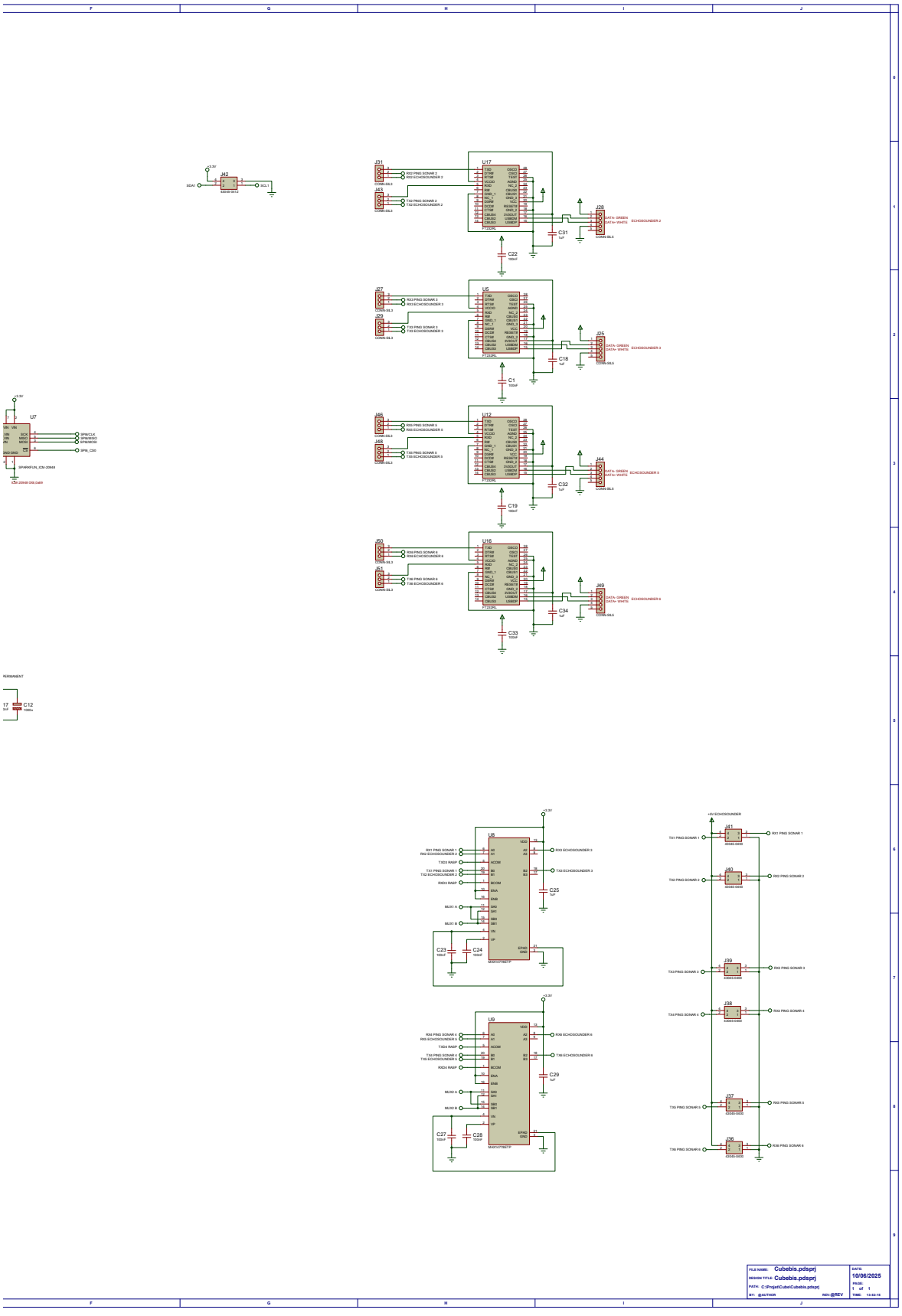


FIGURE 36 – Plan du robot CUBE

Références

[1] Boston Dynamics, *A Retrospective on Uses of Boston Dynamics' Spot Robot*, Boston Dynamics, disponible en ligne : https://bostondynamics.com/blog/retrospective-on-boston-dynamics-spot-robot-uses/, consulté le 14 août 2026.

[2] L. Lapierre, R. Zapata, P. Lepinay et B. Ropars, *Karst Exploration : Unconstrained Attitude Dynamic Control for an AUV*, Ocean Engineering, vol. 219, article 108321, 2021. DOI : https://doi.org/10.1016/j.oceaneng.2020.108321.

[3] N. Saeed, A. Celik, T. Y. Al-Naffouri and M.-S. Alouini, "Underwater Optical Wireless Communications, Networking, and Localization : A Survey," *Ad Hoc Networks*, vol. 94, p. 101935, 2019. doi : 10.1016/j.adhoc.2019.101935.

[4] H. T. Dang, *Underwater Robots for Karst and Marine Exploration : A Study of Redundant AUVs*, PhD thesis, Université de Montpellier, 2021. NNT : 2021MONTS038. HAL : https://theses.hal.science/tel-03417084.

[5] S. O. H. Madgwick, R. Vaidyanathan et A. J. L. Harrison, *An Efficient Orientation Filter for IMU and MARG Sensor Arrays*, Department of Mechanical Engineering, University of Bristol, April 2010.

[6] T. A. Johansen and T. I. Fossen, "Control Allocation—A Survey," *Automatica*, vol. 49, no. 5, pp. 1087–1103, 2013. DOI : https://doi.org/10.1016/j.automatica.2013.01.035.

[7] K. Shoemake, *Animating Rotation with Quaternion Curves*, Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques, 1985. Disponible en ligne : https://api.semanticscholar.org/CorpusID:11290566.

[8] Open Robotics, *ROS 2 Documentation : Humble Hawksbill*, disponible en ligne : https://docs.ros.org/en/humble/, consulté le 16 août 2026.

[9] Unity Technologies, *ROS-TCP-Connector*, Unity Robotics, disponible en ligne : https://github.com/Unity-Technologies/ROS-TCP-Connector, consulté le 16 août 2026.

[10] RobotShop, *Platine de Déploiement 9DOF IMU ICM-20948 (Qwiic)*, SparkFun Electronics, réf. SEN-15335. Disponible en ligne : https://eu.robotshop.com/fr/products/platine-deploiement-9dof-imu-icm-20948-qwiic, consulté le 16 août 2026.

[11] Blue Robotics, *T200 Thruster*, Blue Robotics. Disponible en ligne : https://bluerobotics.com/store/thrusters/t100-t200-thrusters/t200-thruster-r2-rp/, consulté le 16 août 2026.

[12] RobotShop, *Capteur atmosphérique SparkFun BME280 (Qwiic)*, SparkFun Electronics. Disponible en ligne : https://eu.robotshop.com/fr/products/capteur-atmospherique-sparkfun-bme280-qwiic, consulté le 16 août 2026.

[13] Blue Robotics, *Fathom-X Tether Interface Board*, Blue Robotics. Disponible en ligne : https://bluerobotics.com/store/comm-control-power/tether-interface/fathom-x-tether-interface-board-set-copy/, consulté le 16 août 2026.

[14] RS Components, *Microcontrôleur ATmega328P*, Microchip Technology. Disponible en ligne : https://fr.rs-online.com/web/p/microcontroleurs/1310276, consulté le 16 août 2026.

[15] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*, 2nd ed., Wiley, 2021. DOI : https://doi.org/10.1002/9781119575016.

[16] Blue Robotics, *Bar High-Resolution Depth/Pressure Sensors*, Blue Robotics. Disponible en ligne : https://bluerobotics.com/store/sensors-cameras/sensors/bar-depth-pressure-sensor/, consulté le 16 août 2026.

[17] D. H. Titterton and J. L. Weston, *Strapdown Inertial Navigation Technology*, 2nd ed., Institution of Engineering and Technology, 2004. DOI : https://doi.org/10.1049/PBRA017E.