Rapport de stage de PFE

Intégration d'échosondeurs dans l'architecture matérielle et logicielle du Robot CUBE

Hugo Gace - ROB23 - hugo.gace@ensta-bretagne.org
Maitre de stage: Lionel Lapierre

Résumé

L'exploration karstique constitue un enjeu sociétal majeur. La ressource en eau douce qu'héberge les karsts va devenir indispensable au fil des années afin d'alimenter les réserves d'eau potable. Néanmoins cet environnement étant difficile d'accès et dangereux pour les plongeurs spéléologues, le robot sous-marin pourrait remplacer ces plongeurs pour limiter les risques.
Le robot 'Cube' est un robot d'exploration karstique expérimental développé par le LIRMM de forme cubique compoprtant 8 moteurs, un par sommet. L'objectif de ce projet est dans un premier temps d'implémenter des capteurs extéroceptifs sur ce robot afin de lui permettre de percevoir un environnement restreint et fermé tel que le karst. Dans un second temps, l'objectif est d'associer les données des capteurs à l'algorithme de contrôle pour permettre au robot d'évoluer de manière autonome dans ce même environnement.

Abstract

Karstic exploration is a major societal issue. The freshwater ressources found in karsts will become essential over the years to provide drinking water reserves. However, this environment being difficult to access and dangerous for cave divers, the underwater robot could replace divers to limit the risks.
The 'Cube' robot is an experimental karstic exploration robot developed by the LIRMM. It is cubic-shaped and has 8 motors, one per vertex. The aim of this project is firstly to implement exteroceptive sensors on this robot to enable it to perceive a narrow and closed environment such as karst. Secondly, the aim is to associate the sensor data with the control algorithm to enable the robot to evolve autonomously in this same environment.

Remerciements

Premièrement, je tiens à remercier mon maitre de stage Lionel Lapierre pour m'avoir pris en stage, pour m'avoir suivi et aidé tout au long de ce stage, pour m'avoir fait découvrir le monde de la recherche mais aussi celui du travail de terrain et pour m'avoir fait découvrir la robotique karstique.
Je voudrais remercier les membres de Polytech Montpellier Eric Dubreuil, Michel Facerias et Olivier Moïse pour leur aide et leurs bonnes idées.
Merci aussi à Tho Dang, chercheur au LIRMM, pour son travail complet sur le Robot CUBE, sa gentillesse et les conversations intéressantes que nous avons eues autour d'un café.
Merci à toute l'équipe du NavScout, chercheurs comme plongeurs, pour leur accueil et leur bonne humeur lors des missions sur le terrain.

Partenaires

Table des matières

1 Introduction **6**
1.1 Contexte . . . 6
1.1.1 Contexte de la localisation sous-marine . . . 6
1.1.2 Contexte du PFE . . . 7
1.2 Objectifs du projet . . . 8

2 Implémentation logicielle des échosondeurs **9**
2.1 Première approche . . . 9
2.2 nouvelles approches . . . 10
2.2.1 USB/UART . . . 10
2.2.2 I2C/UART . . . 12
2.3 Utilisation de ROS 2 . . . 14
2.3.1 Publisher et launch . . . 14
2.3.2 Lancement du nœud ROS au boot . . . 15

3 Implémentation matérielle des échosondeurs **18**
3.1 Support pour échosondeurs . . . 18
3.1.1 Vérification de la position du sonar . . . 18
3.1.2 Vérification du maintien en position de l'anneau par serrage . . . 19
3.2 Conception du PCB . . . 23

4 Évitement d'obstacle **25**
4.1 Simulation . . . 25
4.2 Contrôle et évitement d'obstacle . . . 28

5 Conclusion **32**

Annexes **I**
Liste des Acronymes . . . I
Liste des Figures . . . II

A Déroulement du projet **III**

B codes **IV**
B.1 Publisher . . . IV
B.2 Launch file . . . IV

C Support **VI**
C.1 Couple serrage standard . . . VI

D PCB **VII**
D.1 Schéma électronique . . . VII
D.2 Conception du PCB . . . VIII

E Simulation **IX**
E.1 Fonction "TriangleRayIntersection" . . . IX
E.2 Fonction "SONAR_PROFILO" . . . XVI

Glossaire

ASCII étendu Ensemble de normes de codage ayant en commun le codage ASCII, développer pour palier au manque de caractères du codage ASCII américain. En France, le terme ASCII étendu fait souvent référence à la norme de codage ISO-8859-1 comprenant notamment les lettres accentuées. 13

kernel Noyau du système d'exploitation de Linux, permettant la gestion des ressources du système et la communication des différents matériels et logiciels. 9, 14

LIRMM Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, organisme d'accueil du stage, laboratoire rattaché à l'Université de Montpellier. 1, 2, 6, 7

PMMA Le poly méthacrylate de méthyl acrylique, aussi connu comme le plexiglass, est un thermoplastique transparent. 22, 23

POM-C Le polyoxyméthylène, ou polyacétal copolymère, est un thermoplastique usinable utilisé pour fabriquer des pièces à hautes propriétés mécaniques. 19, 22, 23

Udev Système de gestion du répertoire "/dev", utile notamment pour gérer et renommer les périphériques. 11, 12

vis CHC Vis à tête cylindrique hexagonale creuse. 19, 20

1 Introduction

1.1 Contexte

1.1.1 Contexte de la localisation sous-marine

Le karst est un ensemble de cavités sous-terraines et de surface créées par l'eau lors de son passage dans un environnement de roches solubles telles que le calcaire ou la dolomite. Cela crée un réseau de galeries partiellement inondées d'eau douce appelées aquifères [Bak03]. L'exploration et la cartographie de ces réseaux se fait par les spéléo-plongeurs, mais cela est très risqué du fait d'être dans un milieu fermé où il est impossible de rejoindre rapidement la surface en cas de problème. De plus les topographies réalisées comportent bien souvent des décalages, les spéléo-plongeurs ne transportant pas des appareils de mesure précis, ou ne sont pas complètes du fait qu'il y ait des passages trop étroits pour pouvoir être franchis par un humain. L'utilisation de robots sous-marins pour explorer et cartographier les karst est très pertinent, à la fois pour limiter les risques humains, mais aussi pour pouvoir cartographier avec précision de plus grands réseaux. La cartographie de ces réseaux est un enjeu majeur de la gestion de la ressource en eau. En effet le karst représente un immense réservoir d'eau douce, et donc d'eau potable. Encore faut-il savoir où forer afin d'avoir accès à cette ressource. De plus, lors de forte précipitations, le karst peut être victime de cru ou de courants plus violents que d'habitude pouvant abîmer les installations de pompage. L'étude de la topographie et de la dynamique de l'eau dans ce milieu est essentielle avant l'extraction et la gestion de cette ressource, d'autant plus que le pays subit de plus en plus de sécheresses.

En France, la première opération d'exploration karstique à l'aide de robots remonte à 1967 avec le Remotely Operated Underwater vehicle (ROV) "Télénaute" dans la Fontaine de Vaucluse où il atteint 106m de profondeur. Entre 1983 et 1993, 5 robots sont perdus dans la Fontaine de Vaucluse et ont dû être récupérés par des plongeurs. La cause de ces pertes est le cordon reliant les ROV à la surface. Le cordon permet de contrôler le robot depuis la surface, et permet aussi un retour d'informations en temps réel, très utile à l'hydrogéologue. Malheureusement, ce cordon devient un obstacle lors de la remontée, et les robots s'emmêlent souvent dedans, cassant leurs moteurs.

Le projet ALEYIN, porté par l'equipe EXPLORE du LIRMM, propose une nouvelle approche de l'exploration karstique, palliant aux problèmes précédemment rencontrés. La stratégie est la suivante (1) : le robot agit en 2 phases, la phase d'exploration et la phase de retour.

Lors de la phase d'exploration, le robot est téléopéré à l'aide d'un câble qu'il dépose au fur et à mesure. Le robot se centre automatiquement dans la cavité à l'aide de ses capteurs acoustiques qui lui permettent aussi d'acquérir des données sur la topographie du karst.
Une fois la phase d'exploration terminée, le robot se libère de son câble et remonte seul à la surface à la fois à l'aide du câble servant de fil d'Ariane, et de la carte qu'il a construit à l'aide des données relevées lors de la phase d'exploration grâce à un algorithme de Simultaneous Localization And Mapping (SLAM). Le câble déroulé serait un câble actif,

permettant de communiquer avec le robot. Ainsi le câble ne gênerait plus la remontée, et permettrait à de futurs robots de repartir en autonomie complète.

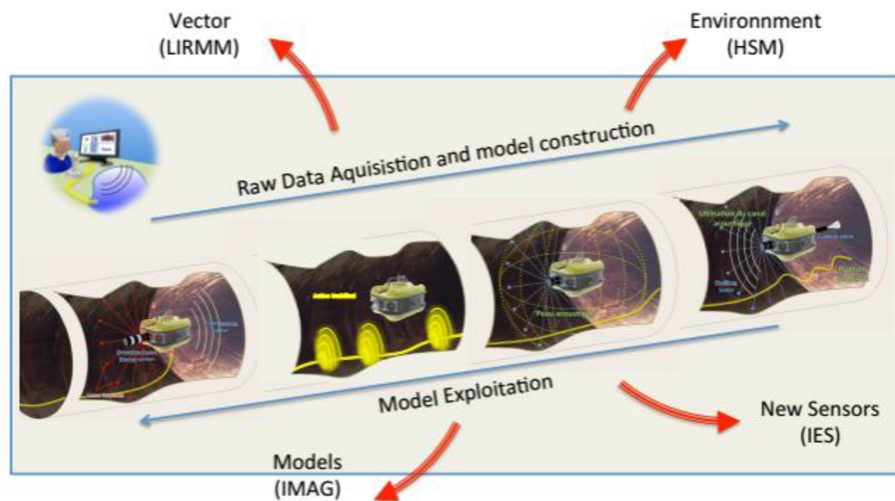


FIGURE 1 – Stratégie d'exploration karstique du projet ALEYIN

1.1.2 Contexte du PFE

Le robot CUBE est un robot expérimental d'exploration karstique développé par le LIRMM. Il est de forme cubique est comporte 8 moteurs, un par sommet, ce qui implique une redondance. L'optimisation de la position des moteurs ainsi que les lois de commandes des moteurs ont été réalisés par Huu-Tho Dang lors de sa thèse. La redondance des moteurs apporte de la robustesse, de la tolérance aux pannes et de la réactivité. Néanmoins cela implique aussi une plus grande difficulté à développer des algorithmes de contrôle [Dan21].

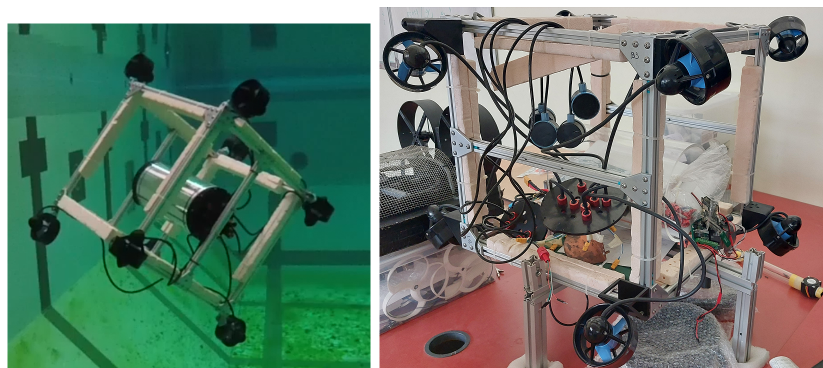


FIGURE 2 – Robot CUBE

Pour que le robot puisse être contrôlé de manière autonome, il a besoin de capteurs extéroceptifs afin d'être en mesure de percevoir son environnement. Le problème est qu'on ne peut pas utiliser n'importe quel type de capteur. L'environnement sous-marin induit l'utilisation de capteurs acoustiques tels que les sonars. Mais l'environnement karstique restreint limite aussi le type de capteurs. En effet, un signal trop longue portée ne ferait que rebondir sur plusieurs parois, faussant ainsi la donnée, mais un sonar avec une portée trop courte pourrait, en cas d'arrivée dans une vaste salle du karst, ne pas détecter les parois. Il faut donc trouver un capteur travaillant à la fois à faible portée mais aussi à portée modérée. Le robot étant expérimental, il ne faut pas non plus que les capteurs coûtent une fortune. Le choix s'est porté sur les échosondeurs Ping de Bluerobotics qui coûtent environ 400€ et ont une portée de 2m à 50m. Il y aura 6 sonars sur le robot, un centré sur chaque face.

Figure 3 – Sonar Ping Bluerobotics

1.2 Objectifs du projet

Le projet d'implémentation matérielle et logicielle des sonars avait déjà été commencé par différents stagiaires, apportant chacun un peu plus au travail du précédent. L'objectif de ce projet est de finir l'implémentation logicielle et matérielle des sonars tout en passant sous Robot Operating System 2 (ROS 2). Dans un second temps, Je devais aussi travailler sur le contrôle du système en prenant en compte les données des capteurs en travaillant notamment sur la simulation MATLAB et en trouvant une stratégie d'évitement d'obstacle viable. Un diagramme de Gantt du projet est disponible en annexe

2 Implémentation logicielle des échosondeurs

2.1 Première approche

Les sonars communiquent via Universal Asynchronous Receiver Transmitter (UART), malheureusement la Raspberry Pi 3 (Pi3) ne comporte que 2 ports UART alors que nous avons 6 sonars soit un besoin de 6 ports UART. Les précédents stagiaires ont eu l'idée d'utiliser une carte NUCLEO-F746ZG comportant un microcontrôleur STM32F746ZG capable de gérer 8 UART. Leur Pi3 utilisait une image Raspbian, le système d'exploitation de Raspberry. Pour traiter les données sonars, ils utilisaient directement le protocole de communication des sonars [1] qu'ils traitaient sur le microcontrôleur, puis envoyaient les données pertinentes sur la Pi3 via UART.

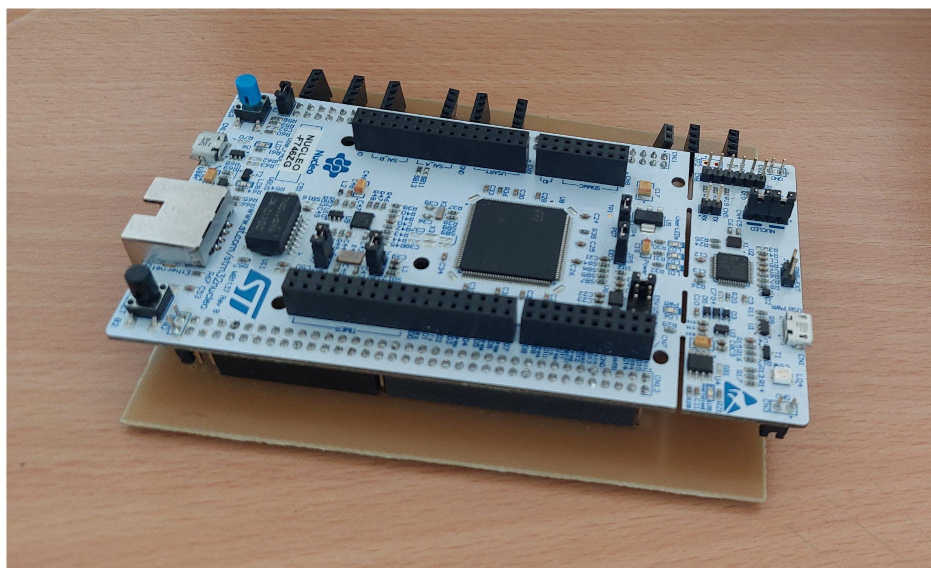


Figure 4 – Nucleo-F746ZG sur son PCB muni des ports UART

Mon premier objectif consistait à reprendre le travail de mes prédécesseurs pour le mettre au propre, l'améliorer et le terminer. Je devais aussi l'adapter afin de recevoir les données sous ROS 2. ROS 2 étant difficile à utiliser sous Raspbian, mais très facile sous Ubuntu, j'ai choisi de changer d'OS et d'utiliser une image Ubuntu sur la Pi3. Lors de la récupération des trames envoyés par la STM32, celles-ci ne correspondaient pas du tout à celles reçues par la configuration sous Raspbian. En effet, l'UART de la Pi3 est un UART logiciel, sa configuration dépend de la configuration de l'OS. De plus, sur la Pi3, l'UART est partagé avec le bluetooth, il faut choisir dans les fichiers de configuration de la Pi3 si l'on veut le bluetooth ou l'UART. Dans notre projet, le bluetooth n'est pas utilisé, mais cela peut être un problème dans d'autres projets. Pour activer l'UART, il faut modifier 2 fichiers de configuration de la carte. Il faut enlever *"console=ttyAMA0,115200"* du fichier "/boot/firmware/cmdline.txt" pour désactiver les messages de débuggage du kernel sur l'UART. Il faut aussi modifier le fichier "/boot/firmware/config.txt" en ajoutant les

1. protocole disponible à cette adresse : https://docs.bluerobotics.com/ping-protocol/

lignes *"enable_uart=1"* et *"dtoverlay=pi3-disable-bt"* pour activer l'UART et désactiver le bluetooth.

Une fois l'UART activé, la commande *stty* permet de vérifier la configuration du port série et de la modifier (baud rate, bit de parité...). Cependant, après avoir mis la même configuration du port série sur Ubuntu que sur Raspbian, la trame reçue par la Pi3 sous Ubuntu ne correspondait à rien. De plus, après avoir discuté avec plusieurs professeurs de Polytech, nous étions d'accord sur le fait qu'utiliser un microcontrôleur pour récupérer des données de capteurs était un peu lourd et que les codes des précédents stagiaires étaient assez illisibles, et qu'il serait donc difficile pour de futurs stagiaires de le reprendre. J'ai donc décidé de changer complètement de méthode, et de tout reprendre à zéro.

2.2 nouvelles approches

L'objectif, en reprenant tout de zéro, était d'arriver à obtenir des données sonars fiables en utilisant une manière simple, propre, et adaptée au systèmes embarqués.

2.2.1 USB/UART

En cherchant sur la documentation du sonar Ping de Bluerobotics, je me suis rendu compte qu'il existait une librairie Python pour configurer et communiquer avec le sonar de manière simple et efficace. La librairie "ping-python" [2] est directement gérée par Bluerobotics, ce qui est un gage de confiance. La librairie propose un ensemble de fonctions pour configurer le sonar (vitesse du son dans l'eau, gain, portée,...), ainsi qu'un ensemble de fonctions pour lire les données souhaitées du sonar (distance, taux de confiance,...). J'ai effectué des tests de cette librairie à l'aide d'une interface Universal Serial Bus (USB)/UART branchée directement sur mon ordinateur.

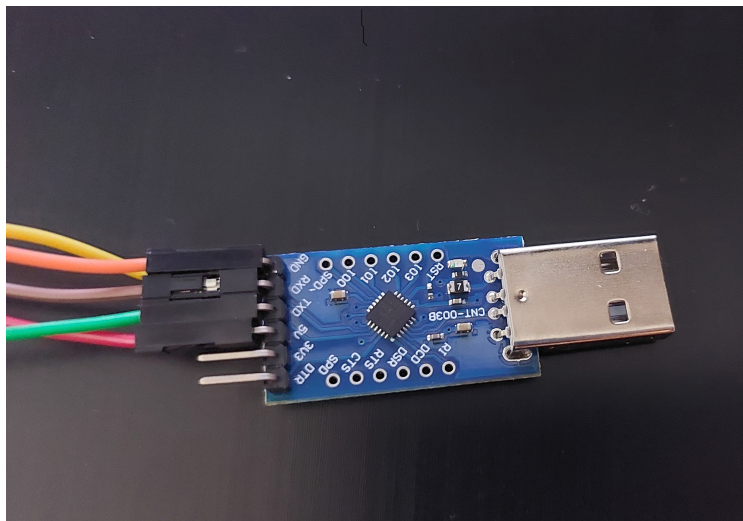


FIGURE 5 – Interface USB/UART

2. librairie disponible sur le git : https://github.com/bluerobotics/ping-python

```
---
data: 856
---
data: 855
---
data: 854
---
data: 854
---
data: 854
---
data: 855
---
data: 855
---
data: 855
---
data: 854
---
data: 854
---
```

FIGURE 6 – Données sonars traitées à l'aide de la librairie "ping-python" et de l'interface USB/UART

Les données récupérées sont les valeurs de distance en millimètres d'un sonar. Pour tester les sonars, je les plaçais dans un bidon en plastique avec de l'eau pour les refroidir. Cette méthode ne permet pas d'obtenir de bonnes données puisque le signal est brouillé par les rebonds sur les parois et la surface trop proche, mais permet de tester si le sonar envoie bien de la donnée, c'est pour cela que lors des tests du taux de confiance, celui-ci ne dépassait pas les 13%. Néanmoins, cela reste plus pertinent que les valeurs que j'obtenais avec la STM32 et la Pi3 sous Raspbian, qui me donnaient des distances d'environ 34000mm avec les sonars dans la même configuration de test.

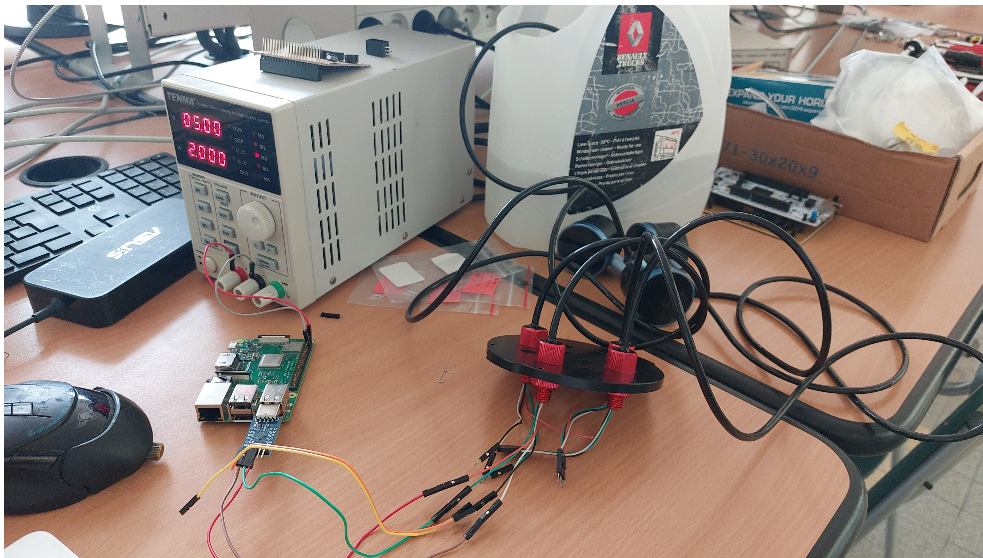


FIGURE 7 – Configuration pour tester un sonar sur la Pi3

Une idée pour obtenir les données des 6 sonars a été de brancher un hub USB sur la Pi3 et d'y brancher 6 interfaces USB/UART comme la précédente. De cette manière, on verrait directement apparaître les chemins des différentes interfaces sous la forme `/dev/tty`. De plus, avec une règle Udev, on peut renommer les chemins des interfaces

afin qu'ils soient propres à chaque interface indépendamment du branchement sur le hub, ce qui faciliterait les opérations de maintenance, où l'opérateur n'aurait pas à connaître les emplacements de chaque USB sur le hub au risque d'intervertir les données. Lorsque l'on branche un périphérique, Udev lit les informations propres au périphérique (numéro constructeur, numéro de série,...), et une règle Udev permet, en fonction de ces informations, de renommer un périphérique. Ci-dessous un exemple de règle Udev pour modifier le nom du périphérique de "/dev/ttySerialx" en "/dev/capt1".

```
  GNU nano 4.8                                              /etc/udev/rules.d/99-usb-serial.rules
SUBSYSTEM=="tty", ATTRS{idVendor}=="10c4", ATTRS{idProduct}=="ea60", ATTRS{serial}=="02ZRD346", SYMLINK+="capt1"
```

FIGURE 8 – Règle Udev pour modifier le nom de l'interface en /dev/capt1

Le principal problème de l'utilisation d'USB est que ce n'est pas un bus de terrain. Il est plus sensible aux perturbations externes et coûte plus cher. De plus les ports USB de la Pi3 servent avant tout au débuggage. Cette solution sera gardée si les autres ne marchent pas ou entraînent des problèmes encore plus importuns.

2.2.2 I2C/UART

Au lieu d'utiliser une interface USB/UART, j'ai tenté d'utiliser une interface Inter-Integrated Circuit (I2C)/dual UART. La Pi3 dispose d'un port I2C sur les pins 3 et 5. L'avantage de l'I2C est que c'est un bus qui fonctionne par adresses. On peut donc connecter plusieurs interfaces sur le même bus, il faudra seulement leur définir des adresses différentes. Il existe dans le commerce différentes interfaces I2C/dual UART. La première que j'ai testé était le module "Gravity : I2C to dual UART", un module chinois de DFRobot proposant une entrée/sortie I2C et deux entrées/sorties UART.

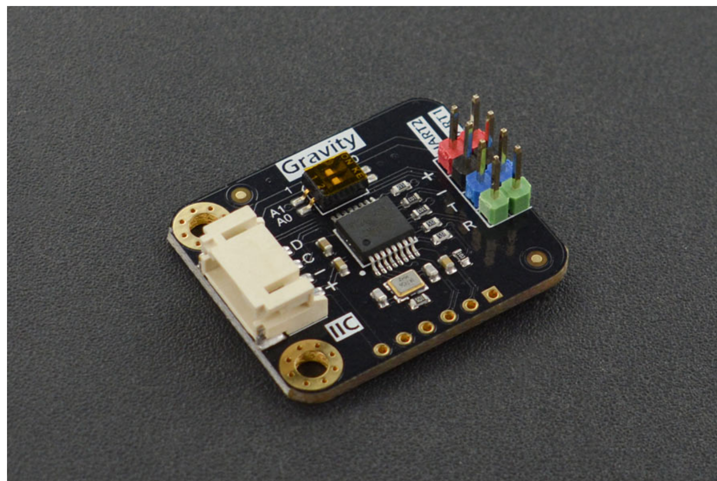


FIGURE 9 – Gravity : I2C to dual UART de DFRobot

L'entreprise propose aussi une librairie python "DFRobot_IIC_Serial" [3] afin de com-

3. librairie disponible sur le git :https://github.com/DFRobot/DFRobot_IICSerial/tree/master/python/raspberrypi

muniquer avec le module et ce qui se trouve branché aux UART. J'ai dans un premier temps tenté d'utiliser cette librairie afin d'envoyer et recevoir les trames des sonars. Pour envoyer une trame, la fonction donnée par la librairie prend en argument une chaîne de caractères, puis transforme ces caractères au format American Standard Code for Information Interchange (ASCII), puis au format binaire, et enfin envoi chaque bit un par un. Le premier problème avec cette méthode est qu'on ne sait pas quelle est la trame allant jusqu'au sonar, le module agit comme une boîte noire, et la documentation étant trouvable uniquement en chinois, cela n'aide pas beaucoup à comprendre ce qui se passe dans le micro-processeur du module. De plus, au lieu de convertir les nombres hexadécimaux de la trame sonar directement en ASCII puis en binaire, la fonction convertit chaque caractère en ASCII avant de convertir en binaire. De plus, les séparateurs (virgules ou espaces) dans la trame sont aussi convertis, ce qui donne au final une trame n'ayant aucune signification pour le sonar.

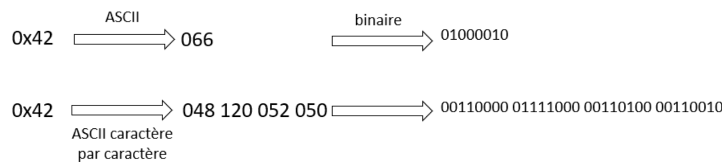


FIGURE 10 – Exemple de conversion avec le première élément de la trame sonar

Étant donné que l'on ne peut pas écrire la trame au format hexadécimal comme on l'aurait fait pour l'UART, j'ai eu l'idée logique d'écrire la trame directement au format ASCII dans la fonction. Or la librairie utilise le format ASCII allant de 0 à 127, soit de 0x00 à 0x7F en hexadécimal. Voici par exemple le protocole pour que le sonar renvoie le message "distance_simple" :

Value (HEX)	Value (decimal)	Type	Name	Notes	Send Bytes
0x42	66	u8	start1	ASCII 'B'	0x42
0x52	82	u8	start2	ASCII 'R'	0x52
0x0002	2	u16	payload_length		0x02, 0x00
0x0006	6	u16	message_id	general_request	0x06, 0x00
0x00	0	u8	src_device_id		0x00
0x00	0	u8	dst_device_id		0x00
0x04BB	1211	u8[2]	payload	requested_id	0xBB 0x04
0x015B	347	u16	checksum		0x5B 0x01

FIGURE 11 – Protocole pour demander l'envoi du message "distance_simple"

Ce qui donne une trame comme ceci :

$$0x42, 0x52, 0x02, 0x00, 0x06, 0x00, 0x00, 0x00, 0xBB, 0x04, 0x5B, 0x01$$

On peut remarquer dans la trame la présence de 0xBB, qui n'est pas compris dans le code ASCII mais dans le code ASCII étendu que ne prend pas en compte la librairie de DFRobot. N'ayant aucun moyen d'utiliser cette librairie pour communiquer avec les sonars, j'ai décidé de changer de module.

Une autre interface I2C/dual UART possible est la "SC16IS752" [4] de NPX-Semiconductors. Cette interface est beaucoup plus documentée sur Internet car beaucoup plus utilisée, et de plus sa documentation est en anglais. L'avantage de cette interface est qu'il existe un module kernel permettant d'outrepasser l'I2C pour venir détecter directement les deux UART via le chemin `/dev/ttyScx`, ce qui permettrait d'utiliser directement la librairie "brping" pour communiquer de manière très simple avec les sonars. Ce module kernel "sc16is7xx.c" [5] est d'ailleurs déjà installée sur les nouvelles versions du kernel Linux.

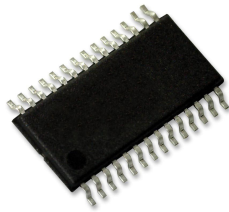


FIGURE 12 – Interface SC16IS752IPW

Cette interface prenant en charge 2 UART, il en faut donc 3 pour gérer tous nos échosondeurs. L'interface possède 2 pins d'adressage A0 et A1, permettant d'avoir 16 adresses possibles suivant les pins sur lesquels ils sont branchés. Par exemple si A0 et A1 sont connectés à V_{DD}, alors l'adresse de l'interface sera 0x90 ; si A0 est connecté à V_{DD} et A1 à SDA, alors l'adresse de l'interface sera 0xA8. Ces multiples adressages permettent d'aligner jusqu'à 16 interfaces sur le même bus I2C.
Pour utiliser ces interfaces, qui sont livrées comme ci-dessus (12), il va falloir les souder à un Printed Circuit Board (PCB). Je reviendrai en détail sur le PCB dans une future partie.

2.3 Utilisation de ROS 2

2.3.1 Publisher et launch

ROS 2 est un middleware open-source qui facilite la création et le contrôle de robots en proposant bibliothèques, outils et normes pour gérer les tâches courantes liées à la robotique. Il permet notamment de faciliter la communication entre les capteurs et les actionneurs par le biais de nœuds indépendants, facilitant ainsi le débuggage et la maintenance. Ce logiciel étant aussi très répandu dans le monde de la robotique, cela facilitera la compréhension du code pour ceux qui reprendront le projet après moi.

ROS 2 fonctionne à l'aide de *Publisher* et de *Subscriber*. Le *Publisher* permet de publier un message auquel le *Subscriber* viens s'abonner pour avoir accès au message. On peut souscrire autant de *Subscriber* que l'on veut à un même message, ce qui permet à une information d'être facilement accessible par un ensemble de programmes.

4. documentation : https://www.nxp.com/docs/en/data-sheet/SC16IS740_750_760.pdf
5. disponible sur le git : https://github.com/torvalds/linux/blob/master/drivers/tty/serial/sc16is7xx.c

Mon objectif est d'avoir un message ROS 2 de type `Int32` par sonar, et non un message comportant les données des 6 sonars qu'il faudra ensuite séparer. Pour cela, j'ai codé un *Publisher* paramétrable utilisant la librairie "brping" de Bluerobotics pour communiquer avec le sonar, lui demander de renvoyer sa mesure de distance et la publier (B.1). Le `Publisher` prend comme paramètres le chemin du périphérique et le taux de confiance désiré. Il vient ensuite ouvrir la communication avec le sonar à l'aide de la librairie "brping", et va effectuer la configuration de la portée du sonar. Dans le `timer_callback` (boucle répétée à intervalles réguliers), on effectue la demande d'envoi de la distance au sonar, et on récupère la réponse si le taux de confiance est supérieur au taux souhaité en paramètre. Le `Publisher` publie ensuite la valeur de distance. S'il n'arrive pas à initialiser le sonar, le `Publisher` s'arrête avec un message d'erreur. Le `Publisher` est le même pour tous les sonars.

Afin que tous les sonars soient traités par le `Publisher`, j'ai réalisé un fichier `launch` avec des `nodes` (B.2). Chaque `node` va appeler le `Publisher` avec ses paramètres, le `launch file` va donc appeler les 6 sonars en même temps. Les `nodes` permettent aussi de renommer les messages pour pouvoir différentier les différents retours des différents sonars.

2.3.2 Lancement du nœud ROS au boot

Les sonars doivent être actifs du démarrage du robot à son extinction, le nœud ROS 2 doit donc lui aussi être actif du démarrage à l'arrêt du robot. Pour ne pas avoir à effectuer de démarrage manuel du nœud, ce qui nécessiterait une connexion au robot et plusieurs manipulations dans le terminal, j'ai choisi de lancer le nœud directement au moment du `boot`, soit au démarrage de la Pi3. Pour cela, j'ai utilisé la suite logicielle `Systemd`. `Systemd` permet de gérer les services de Linux, et notamment les services au démarrage.

`Systemd` a besoin de 2 fichiers pour exécuter un service. Le premier comprend la description ainsi que différentes options de configuration du service, et le second comprend les lignes à exécuter par le service.

Le fichier "/etc/rc.local" est le plus simple à écrire, c'est celui qui comprend les lignes à exécuter par le service (13).

Les deux premières lignes servent à se connecter à un réseau WI-FI particulier défini dans le fichier "/etc/wpa_supplicant/wpa_supplicant.conf", en l'occurrence ici cela permet à la Pi3 de se connecter directement sur le WI-FI de mon téléphone au démarrage. Cela m'évite surtout d'avoir à me reconnecter manuellement à chaque `reboot` de la carte. Les lignes suivantes sont pour le lancement du nœud ROS 2. Je commence par sourcer la ditribution de ROS 2, ici `Foxy`, puis définir l'ID du domaine sur lequel je me place, permettant à tous les appareils sur la même connexion et le même domaine d'être sur le même réseau ROS 2 et donc de pouvoir communiquer avec les autres appareils du réseau. J'affiche ensuite la distribution de ROS 2, ce qui me permet de vérifier au moment du `boot` que `Foxy` soit bien sourcé. Les lignes suivantes permettent de lancer le `launch file` que l'on a vu précédemment. Pour éviter de rester bloqué dans le `boot`,

```
GNU nano 4.8                                                              /etc/rc.local
#!/bin/bash
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other value error.
#
# In order to enable or disable this script just change the execution bits.
#
# By default this script does nothing.

wpa_supplicant -B -c /etc/wpa_supplicant/wpa_supplicant.conf -i wlan0
dhclient wlan0

source /opt/ros/foxy/setup.bash && export ROS_LOCALHOST_ONLY=0 && export ROS_DOMAIN_ID=12
printenv ROS_DISTRO
cd /home/ubuntu/dev_ws
source install/setup.bash
ros2 launch sonars_test sonars_test_launch.py &
exit 0
```

FIGURE 13 – Fichier /etc/rc.local

quelques précautions s'imposent. Il faut enlever tous les `print` présent dans des boucles infinies (par exemple ici dans le "timer_callback"), ou le programme ne cessera jamais d'afficher des lignes, bloquant ainsi le `boot`, ce qui nécessitera de débrancher la carte, de modifier directement le code problématique sur la carte SD et de tout relancer. Il faut aussi impérativement rajouter le "&" à la fin de la ligne lançant le nœud. Le symbole "&" à la fin d'une ligne de commande permet d'exécuter la commande en tâche de fond, ce qui permet au `shell` de reprendre la main et de finir le `boot`.

Le deuxième fichier, "etc/systemd/system/rc-local.service", permet de configurer le service (14).

```
GNU nano 4.8                                      /etc/systemd/system/rc-local.service
[Unit]
 Description=/etc/rc.local Compatibility
 ConditionPathExists=/etc/rc.local

[Service]
# Environment="LD_LIBRARY_PATH=/home/ubuntu/dev_ws/install/sonars/lib:/opt/ros/foxy/lib"
 Environment="LD_LIBRARY_PATH=/opt/ros/foxy/opt/yaml_cpp_vendor/lib:/opt/ros/foxy/lib/aarch64-linux-gnu:/opt/ros/foxy/lib"
 Environment="PYTHON_PATH=/home/ubuntu/dev_ws/install/sonars/lib/python3.8/site-packages:/opt/ros/foxy/lib/python3.8/site-packages"
 Environment="AMENT_PREFIX_PATH=/home/ubuntu/dev_ws/install/sonars:/opt/ros/foxy"
 Environment="HOME=/root"
 Type=forking
 ExecStart=/etc/rc.local
 TimeoutSec=0
 StandardOutput=tty
 RemainAfterExit=yes
 SysVStartPriority=99

[Install]
 WantedBy=multi-user.target
```

FIGURE 14 – Fichier rc-local.service de systemd

Dans ce fichier, je viens configurer le type de service, ici `forking`, qui permet au programme de rester actif. Je viens aussi introduire le paramètre `target Multi-user`, qui spécifie au service de commencer avant l'apparition de l'interface utilisateur, soit avant le `login`. C'est aussi dans ce fichier que je viens spécifier les différents chemins de `Foxy` et de ses librairies. En effet le chemin de base des distributions ROS 2 et les chemins que ces distributions utilisent pour avoir accès à leurs librairies passent par "/home", or à ce moment là, "/home" n'est pas encore défini, il n'est défini qu'après avoir rentré le `login` utilisateur. Il faut donc spécifier au système les chemins parallèles

dans le `root`, le super-utilisateur de Linux, qui lui est défini pendant le `boot` du système.

Une fois les 2 fichiers édités, on rend le fichier "/etc/rc.local" exécutable à l'aide de la commande *chmod* et on active le service avec la commande *"sudo systemctl enable rc-local"*. Ainsi le nœud ROS 2 se lance automatiquement au démarrage de la Pi3 avant de se connecter à l'utilisateur, de plus il reste actif jusqu'à l'extinction de la carte.

3 Implémentation matérielle des échosondeurs

L'implémentation matérielle des échosondeurs est décomposée en deux parties. La première traitera de la conception d'un support pour les échosondeurs afin de pouvoir les monter sur le CUBE. La seconde partie traitera de la conception d'un PCB pour l'intégration des interfaces I2C/dual UART.

3.1 Support pour échosondeurs

Afin de monter les échosondeurs sur le robot, j'ai dû concevoir un support facile à monter et démonter et sécuritaire pour les échosondeurs.

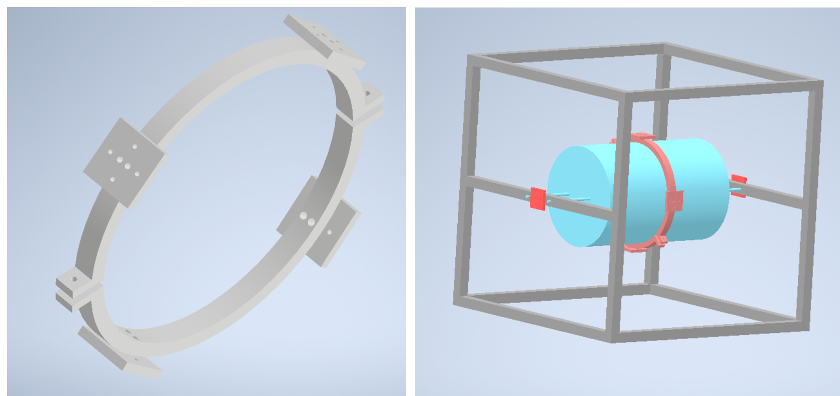


FIGURE 15 – Supports pour échosondeurs

Sur la figure de droite (15), on retrouve en gris le corps du CUBE, en bleu le cylindre étanche et en rouge les supports pour les échosondeurs. Les plaques support des échosondeurs se fixent à l'aide de deux vis, que ce soit sur le corps ou sur l'anneau. L'anneau est en deux parties que l'on vient serrer autour du cylindre à l'aide de 2 vis/écrous.

Pour valider le support proposé, je propose de vérifier 2 facteurs. Le premier est que les sonars positionnés sur l'anneau ne soient pas gênés par les arrêtes du cube. Dans un deuxième temps, il me faut m'assurer que la force de serrage des vis permet de maintenir l'anneau par frottements sur le cylindre.

3.1.1 Vérification de la position du sonar

Le sonar est, à quelques degrés près, centré sur la face du Robot CUBE. Celui-ci a une largeur de faisceau de 30°.

La face du sonar ce trouve à 98mm de la face du cube, et étant centrée sur la face du cube, les arrêtes du cube sont à 230mm du centre du faisceau. En utilisant un simple sinus :

$$sin(\hat{\text{A}}) = \frac{\text{coté opposé}}{\text{hypoténuse}}$$

Je trouve que le faisceau a un rayon d'environ 25mm au passage de la face du cube. Quand bien même le sonar ne serait pas positionné parfaitement centré sur la face, le faisceau ne rencontrera jamais les arrêtes du cube.

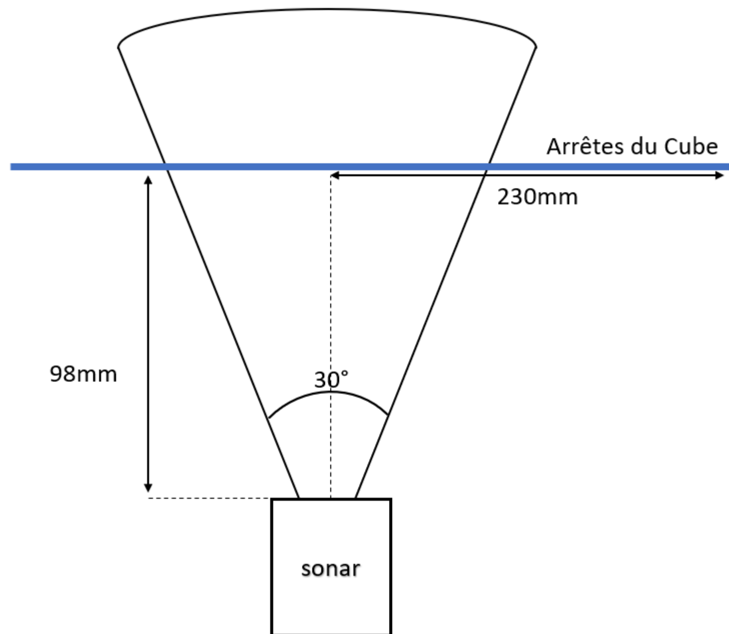


Figure 16 – Schéma position échosondeur

3.1.2 Vérification du maintien en position de l'anneau par serrage

L'anneau n'est maintenu sur le cylindre que par le serrage des vis, venant plaquer et serrer l'anneau sur le cylindre. La force de serrage doit être suffisante pour empêcher l'anneau de glisser sur le cylindre lorsque le robot se déplace, ou qu'il doit faire face au courant.

Pour simplifier l'étude, je pose les hypothèses simplificatrices suivantes :

- Je n'étudierai que la force exercée sur un demi anneau, le raisonnement étant le même pour l'autre demi anneau, symétrique au premier.
- L'anneau est fabriqué en POM-C, un plastique usinable utilisé dans la conception de pièces à hautes propriétés mécaniques. Sa masse volumique est de 1410 kg/m^3 [Bas06].
- Le poids du demi anneau est négligeable par rapport à la force de serrage du fait de son faible volume et de la masse volumique du POM-C, en effet le volume du demi anneau est d'environ $70cm^3$ soit a une masse d'environ 100g, de même la poussée d'Archimède est négligeable du fait du faible volume.
- Les seules forces extérieures sont les forces de frottement fluide.
- Le système est placé dans un conduit de 3m de diamètre rempli d'eau douce à $20°C$ s'écoulant à une vitesse de 6m/s.
- Le robot n'avance pas, il reste immobile, luttant juste contre le courant.
- Les vis utilisées seront des vis CHC M6 de classe 8.8.

Considérons le demi anneau sur le cylindre. Considérons qu'une seule vis effectue le serrage, l'autre servant juste à maintenir l'ensemble. La formule de Kellerman et Klein[Foi+21] nous donne la relation suivante entre le couple de serrage et la force de serrage :

$$C_s = F_s(\frac{p}{2\pi} + \frac{\mu_f r_f}{cos\beta} + \mu_e r_e) \tag{1}$$

avec :

C_s	:	couple de serrage en Nm
F_s	:	force de serrage en N
p	:	pas de la vis en m
β	:	angle des filets
μ_f	:	coefficient de frottement dans les filets
r_f	:	rayon moyen des filets en m
μ_e	:	coefficient de frottement sous l'écrou
r_e	:	rayon moyen sous l'écrou en m

Cette équation (1) est généraliste, elle ne prend pas en compte certains facteurs comme les matériaux mis en jeux ou la taille de la vis, mais elle permet d'avoir une idée globale de la force de serrage.

Pour une vis CHC M6 ISO 4762-DIN 912 de classe 8.8, on a :

$$\begin{aligned} p &= 1mm \\ \beta &= 60^\circ \\ r_e &= 5mm \end{aligned}$$

Pour μ_f et μ_e, je décide de prendre la valeur 0.2, correspondant à une vis neuve sans lubrification. Ces coefficients sont approximatifs et changent avec l'usure de la vis. Pour une vis classe 8.8 non lubrifiée, ces coefficients évoluent entre 0.2 et 0.4 en moyenne.
Pour calculer r_f, j'utilise la formule suivante :

$$\begin{aligned} &d2 = d - 0.6495 \times p \\ &rf = d2/2 \end{aligned} \tag{2}$$

avec :

d	:	diamètre nominale en mm
$d2$	:	diamètre moyen en mm
p	:	pas en mm

Pour cette vis, $r_f = 5.3505mm$. Pour le couple de serrage, je prends le couple de serrage standard pour une vis M6 de classe 8.8 donné dans le tableau en annexe C.1, soit $C_s = 10,25Nm$.
Ainsi à l'aide de l'équation 1, j'obtiens $F_s \approx 3107N$, ce qui serait équivalent à un anneau d'environ 317Kg.

Le calcul des forces de frottement fluide dépend du type d'écoulement. Je calcul donc d'abord le nombre de Reynolds pour déterminer la nature de l'écoulement.

$$Re = \frac{\rho V L}{\eta} \tag{3}$$

avec :

Re	:	nombre de Reynolds
ρ	:	masse volumique du fluide en kgm^{-3}
V	:	vitesse du fluide en $m.s^{-1}$
η	:	viscosité dynamique du fluide en $kg.m^{-1}.s^{-1}$

La viscosité cinématique ν de l'eau douce à $20°C$ étant connue ($10^{-6}m^2.s^{-1}$), on peut simplifier l'équation 3 en :

$$Re = \frac{LV}{\nu} \tag{4}$$

Après l'application numérique de l'équation 4, je trouve Re de l'ordre de 10^7 ce qui indique un écoulement turbulent.

Puisque l'écoulement est turbulent, l'équation des forces de frottement fluide, découlant des équations de Navier-Stokes, peut s'écrire :

$$F_v = \frac{1}{2}\rho V^2 C_x S \tag{5}$$

avec :

F_v	:	forces de frottement fluide en N
ρ	:	masse volumique du fluide en $kg.m^{-3}$
V	:	vitesse du fluide par rapport à l'objet en $m.s^{-1}$
C_x	:	coefficient de trainée
S	:	surface de l'objet perpendiculaire au à la vitesse du fluide en m^2

Pour le C_x, n'ayant pas trouvé de valeur pour un cylindre orienté base perpendiculaire au fluide, je me suis basé sur les travaux de Sighard F.Hoerner [FHo65] où il a calculé un C_x de 1,17 pour les séparations ainsi que pour les demi-sphères orientées face plane perpendiculaire au fluide. Étant les formes qui se rapprochent le plus du cylindre, je prends donc $C_x = 1.17$.
Après application numérique, je trouve $F_v \approx 84N$, c'est assez faible malgré la grande vitesse de l'eau car la surface de contact est très réduite, environ $40cm^2$.

Il y a adhérence d'un objet sur un autre si :

$$F < F_g \qquad \text{avec} \qquad f_g = \frac{F_g}{P} \tag{6}$$

avec :

F	:	force de poussée appliquée à l'objet, ici les force de frottement fluide, en N
F_g	:	force limite de glissement en N
f_g	:	coefficient de glissement statique
P	:	poids, ici la force de serrage de la vis, en N

Malheureusement, la donnée du coefficient de frottement statique du POM-C sur le PMMA (le matériau du cylindre) est inconnue, soit jamais calculée, soit jamais publiée. Néanmoins, le coefficient de frottement dynamique du POM-C est connu, il varie suivant les matériaux de contact entre 0.15 et 0.35. Alors que le frottement statique est la force qui empêche de mettre en mouvement un objet sur un autre, le frottement dynamique est la force de frottement induite par le mouvement d'un objet sur un autre. La force de frottement dynamique est toujours inférieure à la force de frottement statique limite. Ainsi, si la force de poussée est toujours inférieure à la force de frottement dynamique, l'anneau ne pourra pas glisser sur le cylindre et gardera son adhérence.

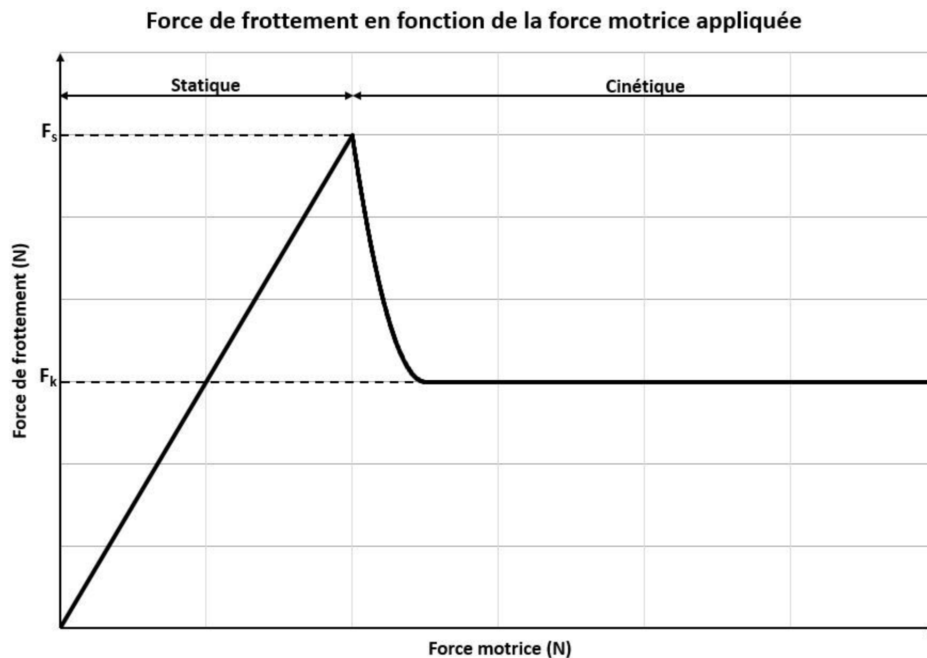


FIGURE 17 – Force de frottement statique et cinétique (dynamique)

Ne connaissant pas précisément le coefficient de frottement dynamique du POM-C sur le PMMA, j'ai calculé les forces de frottement dynamique à partir des coefficients de frottement dynamique limites du POM-C présents dans la littérature. Ainsi après application numérique :

$$\text{Pour} f_g = 0.35, \quad F_g \approx 1087N > F_v$$
$$\text{Pour} f_g = 0.15, \quad F_g \approx 466N > F_v$$

Conclusion

Dans ce cas là, je peux conclure que l'anneau ne glissera pas sur le cylindre car les forces de frottement fluide ne dépassent jamais la force de frottement dynamique moyenne. Néanmoins, il se peut que le coefficient de frottement dynamique, ainsi que le coefficient de frottement statique du POM-C sur le PMMA soit plus petit que 0.15, et ainsi les forces de frottement fluide viendraient faire glisser l'anneau sur le cylindre et casser le système. Il faudrait, dans ce cas de figure, un coefficient de frottement statique inférieur à 0.027, ce qui est assez peu probable. De plus, même si $6m.s^{-1}$ est déjà rapide pour un cours d'eau, il se peut que la vitesse de l'eau, dans certains karst, soient plus grande, engendrant des forces de frottement fluide plus grandes. De plus, avec les hypothèses faites ici, l'eau est considérée comme de l'eau douce pure, or dans les karst, l'eau peut être chargée de sédiments plus ou moins gros, surtout après de fortes pluies en surface, ce qui augmenterait aussi les forces de frottement fluide.

3.2 Conception du PCB

Un PCB est un circuit imprimé sur lequel on vient souder des composant électroniques afin d'obtenir une carte avec des branchements faciles à réaliser pour pouvoir être intégrée dans un système de manière simple, efficace et peu encombrante.

Le PCB que je devais concevoir doit pouvoir accueillir 4 interfaces "SC16IS752IPW", être relié aux pins de la Pi3, pouvoir accueillir les branchements de 8 échosondeurs et pouvoir être branché aux batteries pour l'alimentation 5V. Avoir une interface de plus permettrait, en cas de défaillance de l'une d'entre elles, d'en avoir une autre de secours prête à l'emploi.

Pour réaliser le PCB, j'ai utilisé le logiciel KiCad 7.0, une suite logicielle gratuite de conception de circuits électroniques.

La première étape est la création d'un schéma électronique du système. Cela permet d'avoir une vision d'ensemble du système et de placer les connexions entre les composants (D.1). Les schémas de certains composants, comme l'interface I2C/dual UART sont disponibles directement sur internet. Pour d'autres, il faut les créer à l'aide de l'outil de création de symboles.

Une fois le schéma électronique réalisé, on peut réaliser la conception du PCB. A l'aide de l'outil de conversion de schéma en PCB, le logiciel place directement les `footprints` des éléments du schéma dans l'éditeur de PCB. Les `footprints` sont soit disponibles dans la bibliothèque présente sur le logiciel, soit trouvables sur internet et importables. S'ils n'existent pas, on peut toujours en créer à l'aide de l'outil de conception de `footprint`. Une fois les `footprints` placés comme souhaité, il faut réaliser les connexions entre les composants. Les PCB comportent une ou deux faces actives suivant les utilisations. Les liaisons sur une même face ne peuvent pas se croiser, auquel cas elles seraient considérées comme un même fil. Dans mon cas, il était impossible de faire toutes

les connexions sur une même couche, j'ai donc conçu un PCB double face (D.2).

J'ai ensuite fabriqué le PCB à l'aide d'une machine présente à Polytech Montpellier. Une fois celui-ci fabriqué, il faut souder les différents éléments à la plaque. C'est un travail minutieux et difficile, surtout pour souder les "SC16IS752IPW", qui font environ 8*6.5 mm^2 et dont les pins, qui doivent être soudés sur la face supérieure, font 0.3mm de large.

Une fois les éléments soudés, j'ai pu tester le PCB. Malheureusement celui-ci ne marchait pas. Pour vérifier d'où venait le problème, j'ai utilisé un voltmètre afin de tester les connexions. Il s'est avéré que le problème venait des `vials`, les passages d'une couche à l'autre, qui ne transmettaient pas le courant. Afin de quand même pouvoir tester le système, j'ai tenté de passer outre la deuxième couche en soudant des fils sur le PCB, malheureusement sans succès. J'ai donc effectué quelques changement sur le PCB afin de faciliter l'opération de soudage, mais aucun personnel compétent à utiliser la machine à PCB n'étant présent jusqu'à la fin de mon stage, je n'ai pas pu le refabriquer ni le tester.

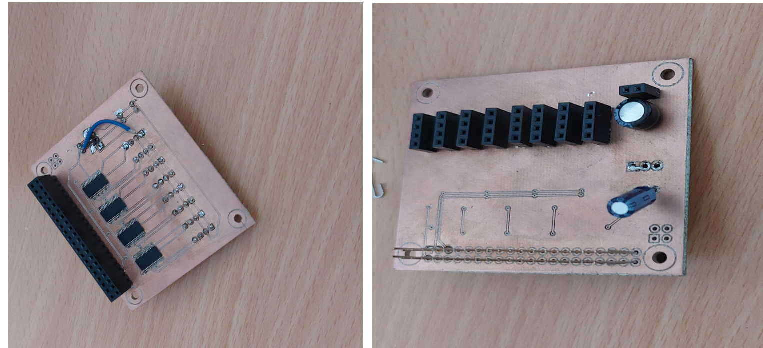


FIGURE 18 – PCB final

4 Évitement d'obstacle

Dans cette partie, je vais traiter de la stratégie d'évitement d'obstacle retenue pour le Robot CUBE ainsi que la simulation du Robot CUBE dans un karst.

4.1 Simulation

Avant de définir la stratégie d'évitement d'obstacle, je devais reprendre la simulation fournie par mon maître de stage et la modifier afin qu'elle coïncide avec le modèle du CUBE actuel. La simulation à été réalisée sous MATLAB. Dans la simulation originelle, le Robot CUBE est équipé d'un sonar rotatif à 360° similaire au sonar Ping360 de Bluerobotics. Le robot naviguait dans la cavité en suivant un point lièvre, point se déplaçant sur un chemin prédéfini, et le robot calculant sa direction pour rejoindre le lièvre. En se déplaçant, le robot scanne la cavité à l'aide de son sonar profilométrique. Le karst est simulé par un ensemble de triangles limitrophes dont les sommets sont contenus dans 3 matrices $vert1$, $vert2$, $vert3$. En associant $vert1(1)$, $vert2(1)$ et $vert3(1)$ on obtient un des triangles du karst.

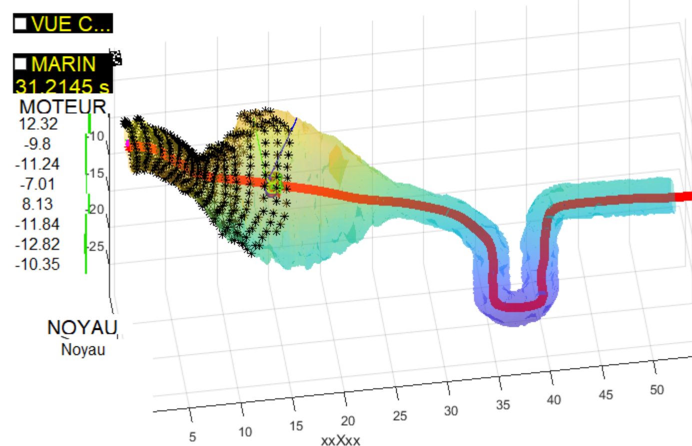


FIGURE 19 – Simulation originelle avec sonar profilométrique

Dans un premier temps, je devais modifier la fonction simulant le sonar rotatif afin de simuler les 6 échosondeurs présents sur ma version du Robot CUBE. Le sonar rotatif est simulé par la fonction "SONAR_PROFILO", qui détermine la direction du faisceau du sonar dans le repère du robot comme :

```
DIRECTION_RAYON_B=[0,cos(Alpha),sin(Alpha)]
```

où Alpha est un paramètre de la fonction définissant l'angle du faisceau du sonar. Pour simuler les 6 échosondeurs, j'ai ajouté un paramètre "nb" à la fonction "SONAR_PROFILO" représentant le numéro du sonar, et j'ai modifié la valeur de "DIRECTION_RAYON_B" en fonction du numéro du sonar :

```
function [Disstance,Alpha,point_inters]=SONAR_PROFILO(nb,
   vert1,vert2,vert3,JS,jd,X,Alpha,Q_B_0,PRM_SNR_PROF)

% DIRECTION_RAYON_B = [0,cos(Alpha),sin(Alpha)];
if nb==1
    DIRECTION_RAYON_B = [1,0,0];
else
    if nb==2
        DIRECTION_RAYON_B = [-1,0,0];
    else
        if nb==3
            DIRECTION_RAYON_B = [0,0,1];
        else
            if nb==4
                DIRECTION_RAYON_B = [0,0,-1];
            else
                if nb==5
                    DIRECTION_RAYON_B = [0,-1,0];
                else
                    if nb==6
                        DIRECTION_RAYON_B = [0,1,0];
                    end
                end
            end
        end
    end
end
```

Cette modification permet bien de faire apparaître les sonar comme s'ils étaient sur les faces du cube.

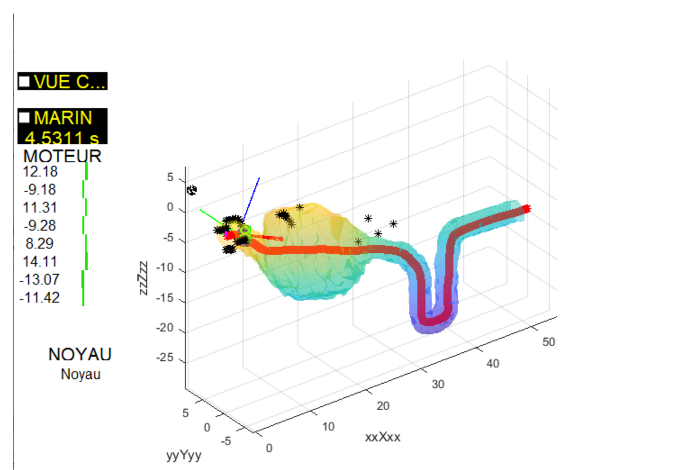


FIGURE 20 – Simulation avec les 6 échosondeurs

Cette simple modification marche très bien pour les sonars pointant vers les côtés de la cavité (soit les sonars suivant y, -y, z et -z dans le repère du robot). Pour le sonar situé à l'avant, on remarque que celui-ci place des points dans le vide ou derrière une première paroi. Penchons nous alors sur l'algorithme d'intersection ligne/triangle.

La fonction "TriangleRayIntersection" est basée sur l'algorithme proposé par Môller et Trumbore [MT05]. La fonction prend comme arguments l'origine des raies, la direction des raies, les sommets des triangles et des paramètres intrinsèques à la fonction. Elle retourne ensuite un tableau "INTERSECT" de booléens donnant quelles paires de triangles et de raies se coupent, un tableau "t" de flottants donnant les distances entre l'origine des raies et les points d'intersections correspondants, "u,v" les coordonnées barycentriques des points d'intersections et "xcoor" le tableau des coordonnés cartésiennes des points d'intersections (E.1).

Dans la simulation originelle, après avoir déterminé les points d'intersections entre les raies et les triangles, si un faisceau coupe plusieurs triangles, la fonction "SONAR_PROFILO" fait la moyenne des coordonnées des points d'intersections pour obtenir le point final. Cette méthode marche très bien pour les sonars sur les axes y et z car leurs faisceaux ne coupent généralement que 1 ou deux triangles, mais pour les sonars situés sur l'axe x, leurs rayons coupent une multitude de triangles à la fois proches et lointains. La méthode par la moyenne n'est donc pas adaptée. Au lieu de faire la moyenne des coordonnées des points d'intersection pour un faisceau, je ne garde que le point d'intersection ayant la distance à l'origine du faisceau la plus faible, ce qui permet d'éliminer les points lointains que le sonar n'aurait pas dû capter. Cette donnée est fournie par la fonction "TriangleRayIntersection".

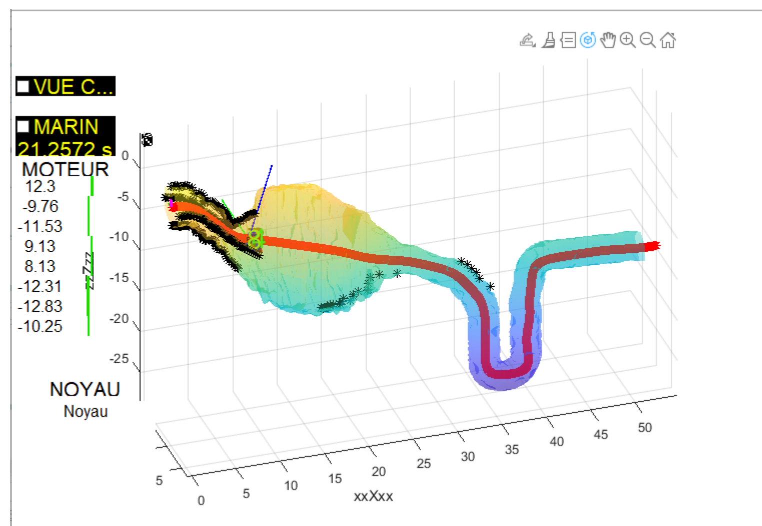


FIGURE 21 – Simulation avec méthode du point le plus proche

La fonction "SONAR_PROFILO" (E.2) permet maintenant la simulation des 6 échosondeurs de manière fiable, même si le robot venait à effectuer une rotation sur lui-même et que le sonar "frontal" venait à changer. Néanmoins la simulation ne reflète pas par-

faitement la réalité, en effet ici les faisceaux des échosondeurs sont modélisé par des segments, tandis que le faisceau réel comporte un angle d'ouverture et devrait donc être modélisé par un cône. Une telle modélisation serait très difficile à mettre en place mathématiquement, notamment pour la gestion des intersections avec les triangles. Nous garderons donc le modèle simplifié de faisceaux rectilignes.

4.2 Contrôle et évitement d'obstacle

Le contrôle du robot se fait par l'intermédiaire d'une "virtual target", ou lièvre. Le robot doit pouvoir converger vers un chemin en poursuivant un point se déplaçant le long du chemin. La méthode 2D de contrôle a été définie dans les publications L.Lapierre et al. [LSP06] et L.Lapierre et B.Jouvencel [LJ08].

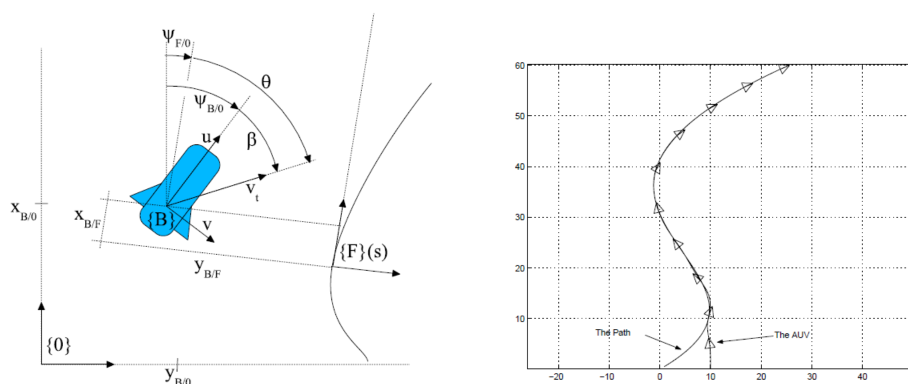


FIGURE 22 – Exemple de paramétrage et résultat d'une simulation avec contrôle par lièvre ([LJ08])

Un repère de Frenet $\{F\}$ est attaché au lièvre. Le but de ce contrôle est de minimiser la distance au chemin ($X_{B/F} et Y_{B/F}$) et de minimiser l'angle entre le vecteur vitesse du robot et la tangente au chemin ($\psi_{B/F}$) [LSP06].

Pour utiliser cette méthode en 3D, on utilise des quaternions afin de contrôler l'attitude du robot comme défini dans L.Lapierre et al. [Lap+21]. Le quaternion Q définit l'attitude du robot par rapport au repère de référence par une rotation d'un angle α autour d'un vecteur normalisé **n**. Q s'exprime de la manière suivante :

$$Q = [cos(\frac{\alpha}{2}), sin(\frac{\alpha}{2}).n^T]^T \tag{7}$$

On définir Q_d le quaternion d'attitude désirée, qui est le quaternion d'attitude du lièvre. La méthode pour trouver les paramètres à donner au CUBE pour suivre le quaternion d'attitude est donnée et expliquée dans L.Lapierre et al. [Lap+21].

La simulation associe ces deux méthodes afin d'obtenir un contrôle du robot en 3D robuste et efficace.

L'algorithme de contrôle ne comporte pas de méthode d'évitement d'obstacle. Si le chemin traverse un obstacle, le robot rentrera dedans. Un algorithme d'évitement d'obstacle basé sur l'algorithme de contrôle par lièvre est proposé par L.Lapierre et R.Zapata [LZ12]. Au moment où le robot détecte un obstacle à l'aide de ses capteurs, celui-ci va suivre un autre chemin permettant d'éviter l'obstacle avant de retrouver le chemin originel une fois l'obstacle franchi. Pour cela, lorsqu'un capteur de proximité détecte un obstacle, on définit C_{OA} le point d'intersection capteur/obstacle le plus proche, puis on définit la Safety Maneuvering Zone (SMZ), notée $S_{OA}(r_S, C_{OA})$ comme un cercle de rayon r_S et de centre C_{OA}. On définit ensuite $P_{OA} = [x_{OA}, y_{OA}]^T$ le point d'intersection entre S_{OA} et $\overrightarrow{P_R C_{OA}}$. On peut enfin définir $T_{OA} = [P_{OA}, \psi_{OA}]$ le lièvre d'évitement d'obstacle avec ψ_{OA} l'angle entre x et la tangente à S_{OA} en P_{OA}. Toute la méthode est donnée et expliquée dans L.Lapierre et R.Zapata [LZ12].

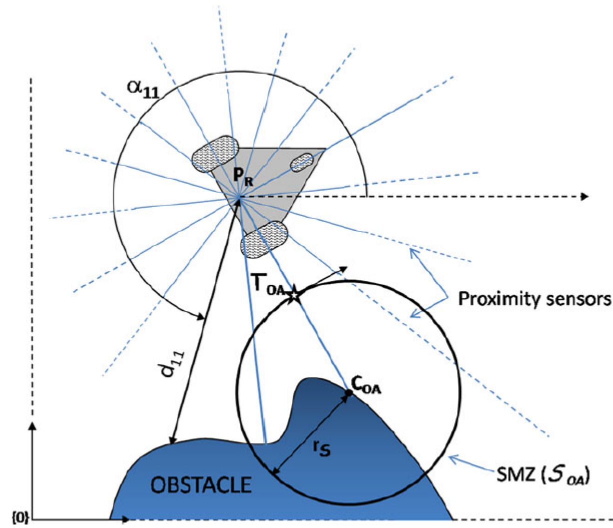


Figure 23 – Exemple de SMZ ([LZ12])

Dans le cas de notre robot ne possédant qu'un échosondeur par face, et donc par conséquent beaucoup d'angles morts, tourner autour de l'obstacle comme décrit dans L.Lapierre et R.Zapata [LZ12] impliquerait de perdre de vue l'obstacle à un moment donné, mettant à mal la méthode d'évitement d'obstacle. Pour éviter cette perte de vue, il faut

$$\begin{aligned} &\psi - \psi_{RC_{OA}} = 0 \\ \text{avec} \quad &\psi_{RC_{OA}} = atan2((y_{OA} - y), (x_{OA} - x)) \end{aligned} \tag{8}$$

avec ψ et $\psi_{RC_{OA}}$ tels que décrit dans L.Lapierre et R.Zapata ([LZ12]) et sur la figure 24.

L'objectif sera donc de minimiser l'équation 8. Pour définir le sens de rotation de T_{OA} sur S_{OA}, on pose

$$\begin{aligned} d_v \quad &: \quad \text{distance retournée par le sonar orienté vers } \overrightarrow{v} \\ d_{-v} \quad &: \quad \text{distance retournée par le sonar orienté vers -}\overrightarrow{v} \end{aligned}$$

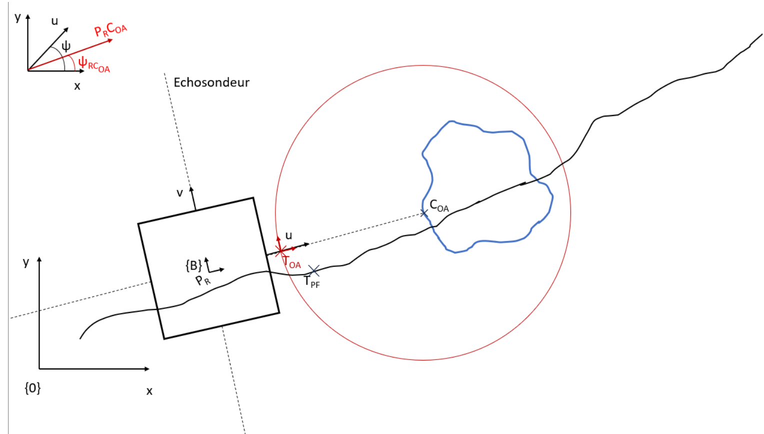


FIGURE 24 – Définition des paramètres et de la SMZ pour le cas du Robot CUBE

On pose alors

$$\begin{aligned} \Sigma_{OA} &= 1 \quad \text{si} \quad [(d_v > d_{-v})(\sigma_{OA} = 1)] \\ \Sigma_{OA} &= 2 \quad \text{si} \quad [(d_v < d_{-v})(\sigma_{OA} = 1)] \\ \Sigma_{OA} &= 0 \quad \text{sinon} \end{aligned} \tag{9}$$

où σ_{OA} est un booléen signalant si l'on se trouve en mode évitement d'obstacle ou suivi du chemin. La direction de l'angle ψ_{OA} est donnée par

$$\begin{aligned} \psi_{OA} &= \psi_{RC_{OA}} + \pi/2 \quad \text{si} \quad [\Sigma_{OA} = 1] \\ \psi_{OA} &= \psi_{RC_{OA}} - \pi/2 \quad \text{si} \quad [\Sigma_{OA} = 2] \end{aligned} \tag{10}$$

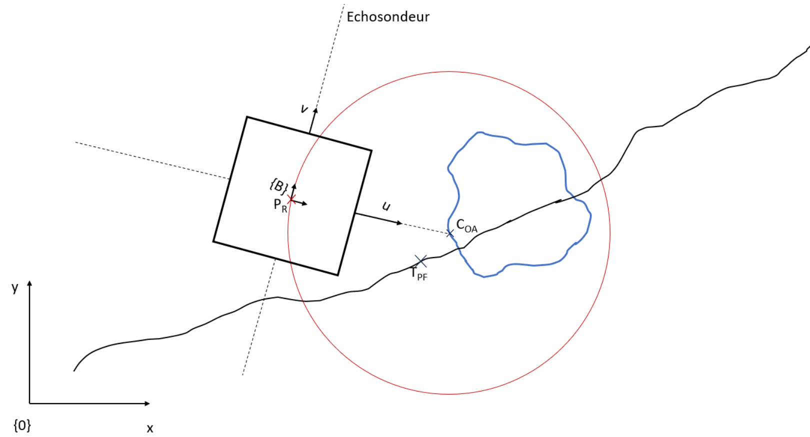


FIGURE 25 – Evitement de l'obstacle

Conclusion

Cette stratégie d'évitement 2D permet au robot d'éviter les obstacles de type stalactites ou stalagmites affleurant du sol ou du plafond de la cavité. La même méthode appliquée aux autres axes du robot permettrait d'avoir une stratégie d'évitement 3D, néanmoins elle ne serait pas assez efficace ni assez robuste comparée à une méthode prenant en compte les quaternions d'attitude et le système 3D global.

5 Conclusion

L'exploration et la cartographie des milieux karstiques par des robots sous-marins est un enjeu majeur. Le Robot CUBE est un robot expérimental ayant pour objectif de tester et d'améliorer différentes approches afin d'obtenir un système robuste et fiable lors de ces explorations.

Lors de ce stage, j'ai pu réaliser l'implémentation logicielle des sonars sur le Robot CUBE, néanmoins par manque de temps, je n'ai pas pu finir l'implémentation matérielle et je n'ai donc pas pu tester la dernière version de l'implémentation logicielle. Ce manque de temps vient en partie de la commande et de la livraison de composants, que l'on ne maîtrise pas mais pouvant être anticipées. Le plus gros manque de temps vient du fait de devoir reprendre à zéro un projet s'étalant sur 4 différents stages. Alors que je devais avancer sur la suite du projet, j'ai perdu un temps considérable à essayer de faire marcher le travail de mes prédécesseurs qui était sensé être fiable. Bien qu'il m'ait fait perdre du temps sur le projet, cela m'a quand même permis de m'intéresser à la configuration bas-niveau de la Raspberry Pi3 et des systèmes Linux, notamment Ubuntu, ce qui me sera sûrement utile plus tard.

Ce stage m'a aussi permis de découvrir un aspect de l'électronique que je ne connaissais pas encore, la réalisation de PCB. Bien que celui-ci n'ait pas marché à cause d'un problème lors de sa création par la machine, venant soit de la machine en elle-même soit des fichiers implémentés dans la machine, cela m'a permis de comprendre comment réaliser un PCB, essentiel à la réduction de l'encombrement d'un système embarqué, et de prendre en main un outil permettant d'aider à la réalisation de celui-ci.

Ayant travaillé sur l'implémentation logicielle et matérielle des échosondeurs sur le robot jusqu'à la fermeture de la Faculté des Sciences de Montpellier, lieu d'entreposage du Robot CUBE, cela ne m'a laissé que peu de temps à accorder à la simulation et à la stratégie d'évitement d'obstacle. J'aurais aimé pouvoir me pencher plus sur l'aspect 3D utilisant les quaternions afin d'arriver à une méthode plus robuste, ainsi que de pouvoir l'implémenter sur la simulation.

Un futur stage pourrait donc voir le jour, où il faudrait finir l'intégration des échosondeurs en fabricant les supports et réaliser un nouveau PCB plus optimisé, et revoir la stratégie d'évitement afin de l'appliquer en 3D en s'appuyant sur l'utilisation des quaternions d'attitude, et enfin implémenter l'algorithme de contrôle sur le robot réel.

Liste des Acronymes

ASCII American Standard Code for Information Interchange

I2C Inter-Integrated Circuit

PCB Printed Circuit Board

Pi3 Raspberry Pi 3

ROS 2 Robot Operating System 2

ROV Remotely Operated Underwater vehicle

SLAM Simultaneous Localization And Mapping

SMZ Safety Maneuvering Zone

UART Universal Asynchronous Receiver Transmitter

USB Universal Serial Bus

Table des figures

1 Stratégie d'exploration karstique du projet ALEYIN 7
2 Robot CUBE 7
3 Sonar Ping Bluerobotics 8
4 Nucleo-F746ZG sur son PCB muni des ports UART 9
5 Interface USB/UART 10
6 Données sonars traitées à l'aide de la librairie "ping-python" et de l'interface USB/UART 11
7 Configuration pour tester un sonar sur la Pi3 11
8 Règle Udev pour modifier le nom de l'interface en /dev/capt1 12
9 Gravity : I2C to dual UART de DFRobot 12
10 Exemple de conversion avec le première élément de la trame sonar 13
11 Protocole pour demander l'envoi du message "distance_simple" 13
12 Interface SC16IS752IPW 14
13 Fichier /etc/rc.local 16
14 Fichier rc-local.service de systemd 16
15 Supports pour échosondeurs 18
16 Schéma position échosondeur 19
17 Force de frottement statique et cinétique (dynamique) 22
18 PCB final 24
19 Simulation originelle avec sonar profilométrique 25
20 Simulation avec les 6 échosondeurs 26
21 Simulation avec méthode du point le plus proche 27
22 Exemple de paramétrage et résultat d'une simulation avec contrôle par lièvre ([LJ08]) 28
23 Exemple de SMZ ([LZ12]) 29
24 Définition des paramètres et de la SMZ pour le cas du Robot CUBE . . . 30
25 Evitement de l'obstacle 30

A Déroulement du projet

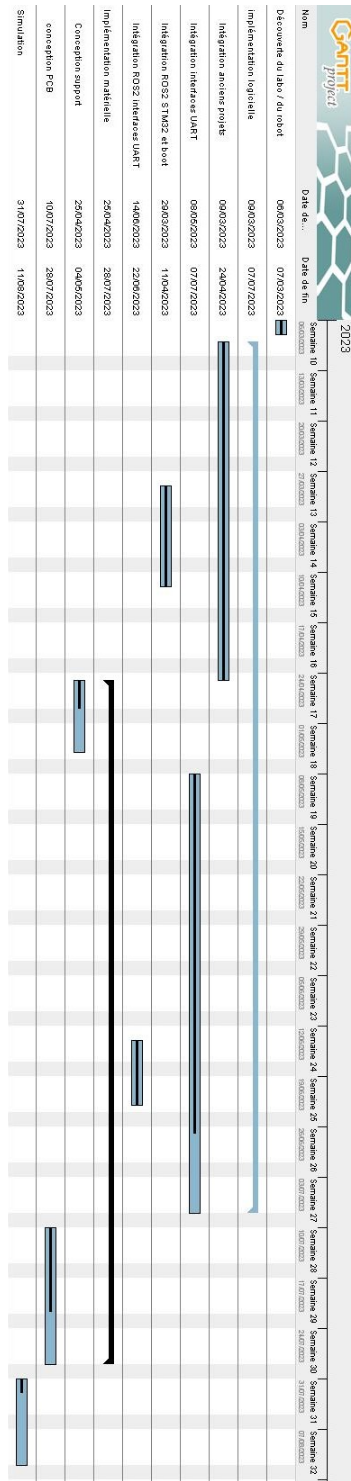


Figure 26 – Diagramme de Gantt

B codes

B.1 Publisher

```
#! /usr/bin/env python3.8

import rclpy
from rclpy.node import Node
from brping import Ping1D
import time
from std_msgs.msg import Int32

class SonarPublisher(Node):

        def __init__(self):
                super().__init__('sonar_publisher')
                self.declare_parameter('serial_path', '/dev/ttySC0')
                self.declare_parameter('confidence_rate', 50)
                self.publisher_=self.create_publisher(Int32, 'data_sonar', 10)
                timer_period=0.5

                param_serial_path = self.get_parameter('serial_path').get_parameter_value().string_value
                param_confidence_rate = self.get_parameter('confidence_rate').get_parameter_value()

                self.Sonar = Ping1D()
                self.Sonar.connect_serial(param_serial_path, 115200)
                if self.Sonar.initialize() is False:
                        print("failed to initialize ", param_serial_path)
                        exit(1)
                self.Sonar.set_mode_auto(0)
                self.Sonar.set_range(0,1000)
                self.timer=self.create_timer(timer_period, self.timer_callback)

        def timer_callback(self):
                msg = Int32()
                sonar_data = self.Sonar.get_distance()
                if sonar_data["confidence"] >= param_confidence_rate:
                        msg.data = sonar_data["distance"]
                self.publisher_.publish(msg)
                self.get_logger().info('Publishing: "%s"' % msg.data)

def main(args=None):
        rclpy.init(args=args)

        sonar_publisher = SonarPublisher()

        rclpy.spin(sonar_publisher)

        sonar_publisher.destroy_node()
        rclpy.shutdown()

if __name__=='__main__':
        main()
```

B.2 Launch file

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
        return LaunchDescription([
                Node(
                        package='sonars_test',
                        executable='test_capt.py',
                        name='test_capt_1',
                        output='screen',
                        emulate_tty=True,
                        parameters=[
                                {'serial_path': '/dev/capt1'}
                                {'confidence_rate': 60}
                        ],
                        remappings=[
                                ('/data_sonar', '/data/sonar1')
                        ]
                ),

                Node(
                        package='sonars_test',
                        executable='test_capt.py',
                        name='test_capt_2',
                        output='screen',
                        emulate_tty=True,
                        parameters=[
```

```
                {'serial_path': '/dev/capt2'},
                {'confidence_rate': 60}
            ],
            remappings=[
                ('/data_sonar', '/data/sonar2')
            ]
        ),

        Node(
            package='sonars_test',
            executable='test_capt.py',
            name='test_capt_3',
            output='screen',
            emulate_tty=True,
            parameters=[
                {'serial_path': '/dev/capt3'},
                {'confidence_rate': 60}
            ],
            remappings=[
                ('/data_sonar', '/data/sonar3')
            ]
        ),

        Node(
            package='sonars_test',
            executable='test_capt.py',
            name='test_capt_4',
            output='screen',
            emulate_tty=True,
            parameters=[
                {'serial_path': '/dev/capt4'},
                {'confidence_rate': 60}
            ],
            remappings=[
                ('/data_sonar', '/data/sonar4')
            ]
        ),

        Node(
            package='sonars_test',
            executable='test_capt.py',
            name='test_capt_5',
            output='screen',
            emulate_tty=True,
            parameters=[
                {'serial_path': '/dev/capt5'},
                {'confidence_rate': 60}
            ],
            remappings=[
                ('/data_sonar', '/data/sonar5')
            ]
        ),

        Node(
            package='sonars_test',
            executable='test_capt.py',
            name='test_capt_6',
            output='screen',
            emulate_tty=True,
            parameters=[
                {'serial_path': '/dev/capt6'},
                {'confidence_rate': 60}
            ],
            remappings=[
                ('/data_sonar', '/data/sonar6')
            ]
        )

    ])
```

C Support

C.1 Couple serrage standard

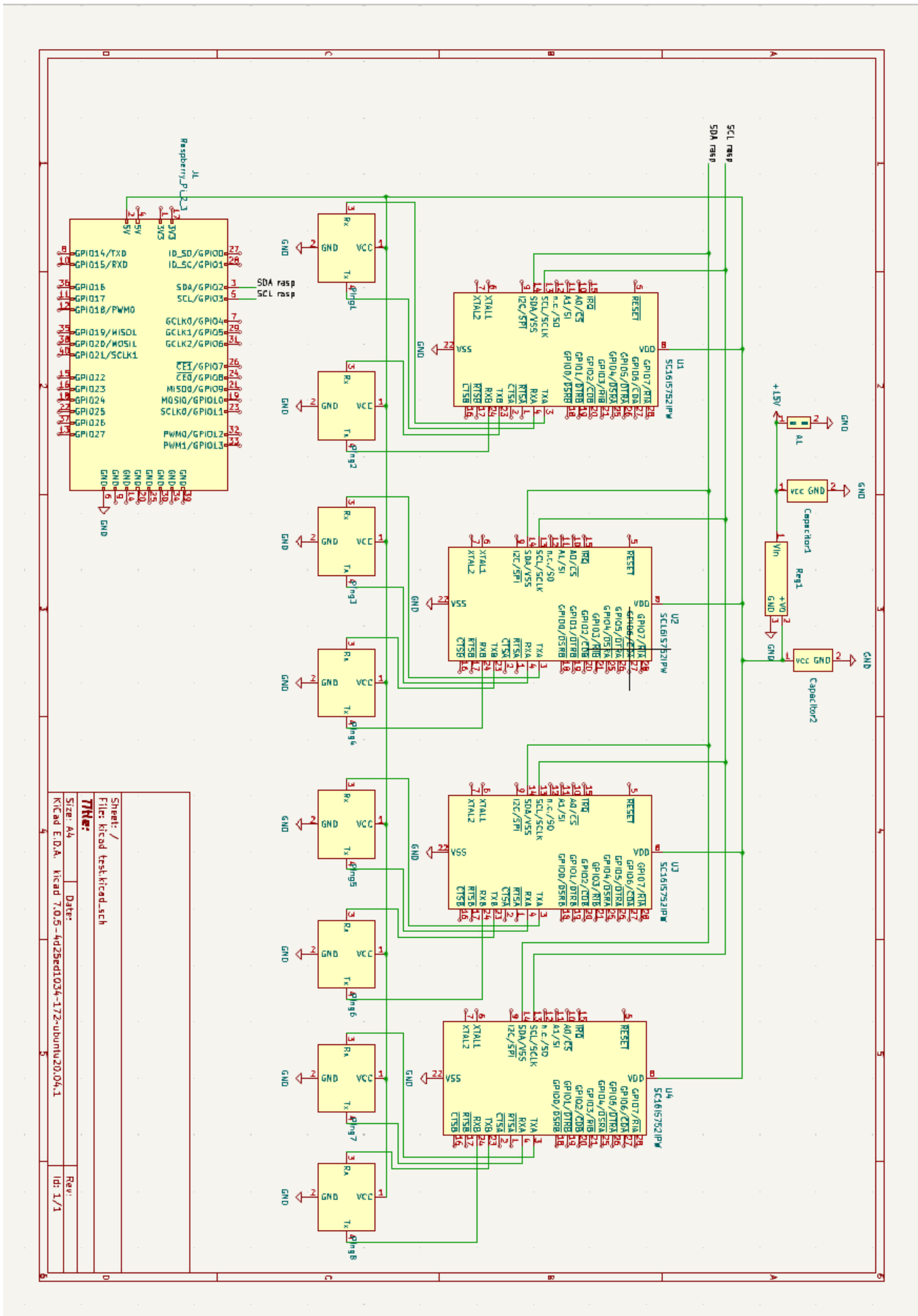
Taille de filetage	Couple de serrage Tableau pour standard Couple de serrage en (Nm)					
Classe de résistance	4.6	5.6	6.8	8.8	10.9	12.9
M2	0,13	0,16	0,26	0,35	0,49	0,59
M2,5	0,27	0,34	0,54	0,72	1,01	1,21
M3	0,48	0,60	0,96	1,28	1,80	2,16
M4	1,12	1,39	2,23	2,97	4,18	5,02
M5	2,26	2,83	4,52	6,03	8,48	10,18
M6	3,84	4,80	7,69	10,25	14,41	17,29
M7	5,13	6,42	10,27	13,70	19,25	23,10
M8	9,35	11,69	18,70	24,93	35,06	42,07
M10	18	23	37	49	70	83
M12	32	40	65	86	121	146
M14	52	65	104	138	194	233
M16	81	101	161	215	302	363
M18	112	139	222	296	417	500
M20	157	197	315	420	590	709
M22	215	269	430	574	807	968
M24	272	340	544	726	1020	1224
M27	400	500	800	1067	1500	1800
M30	542	677	1083	1445	2032	2438
M33	739	923	1477	1969	2770	3323
M36	948	1185	1896	2528	3555	4266
M39	1229	1536	2457	3276	4607	5529

FIGURE 27 – Tableau des couples de serrages standard

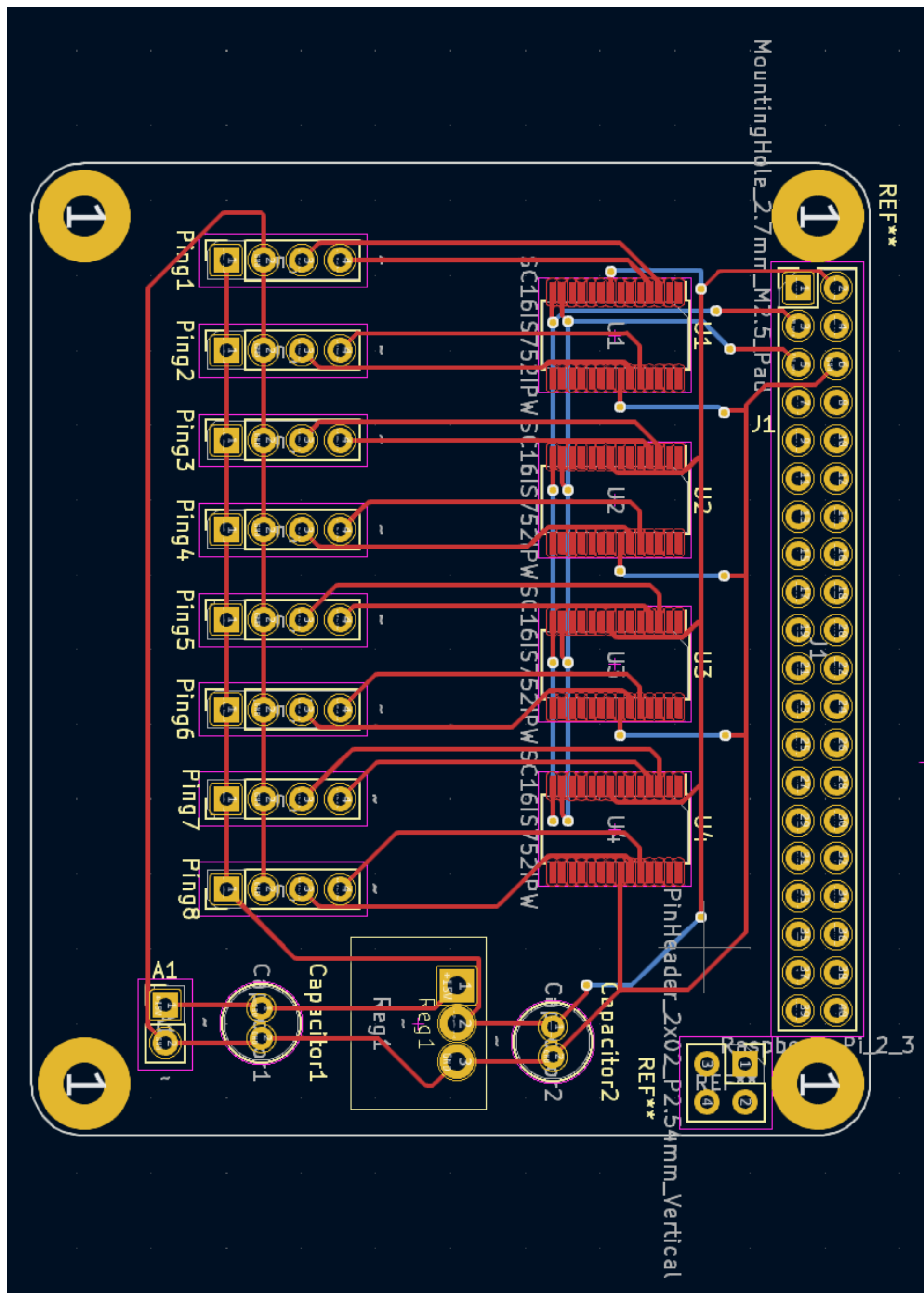
source : https://www.couple-de-serrage.fr/filetages-m%C3%A9triques/

D PCB

D.1 Schéma électronique



D.2 Conception du PCB



En rouge les éléments sur la face supérieure, en bleu les éléments sur la face inférieure et en jaune les trous traversants la carte.

E Simulation

E.1 Fonction "TriangleRayIntersection"

```
function [intersect, t, u, v, xcoor] =
   TriangleRayIntersection (...
  orig, dir, vert0, vert1, vert2, varargin)
%TRIANGLERAYINTERSECTION Ray/triangle intersection.
%    INTERSECT = TriangleRayIntersection(ORIG, DIR, VERT1,
   VERT2, VERT3)
%      calculates ray/triangle intersections using the
   algorithm proposed
%      BY Mller and Trumbore (1997), implemented as highly
   vectorized
%      MATLAB code. The ray starts at ORIG and points toward
    DIR. The
%      triangle is defined by vertix points: VERT1, VERT2,
   VERT3. All input
%      arrays are in Nx3 or 1x3 format, where N is number of
    triangles or
%      rays.
%
%   [INTERSECT, T, U, V, XCOOR] = TriangleRayIntersection
   (...)
%     Returns:
%     * Intersect - boolean array of length N informing
   which line and
%                  triangle pair intersect
%     * t   - distance from the ray origin to the
   intersection point in
%             units of |dir|. Provided only for line/
   triangle pair that
%             intersect unless 'fullReturn' parameter is
   true.
%     * u,v - barycentric coordinates of the intersection
   point
%     * xcoor - carthesian coordinates of the intersection
   point
%
%   TriangleRayIntersection(...,'param','value','param','
   value'...) allows
%    additional param/value pairs to be used. Allowed
   parameters:
%    * planeType - 'one sided' or 'two sided' (default) -
   how to treat
%       triangles. In 'one sided' version only
```

```
   intersections in single
%        direction are counted and intersections with back
   facing
%           tringles are ignored
%    * lineType - 'ray' (default), 'line' or 'segment' - how
    to treat rays:
%        - 'line' means infinite (on both sides) line;
%        - 'ray' means infinite (on one side) ray comming
   out of origin;
%        - 'segment' means line segment bounded on both
   sides
%    * border - controls border handling:
%        - 'normal'(default) border - triangle is exactly as
    defined.
%           Intersections with border points can be easily
   lost due to
%           rounding errors.
%        - 'inclusive' border - triangle is marginally
   larger.
%           Intersections with border points are always
   captured but can
%           lead to double counting when working with
   surfaces.
%        - 'exclusive' border - triangle is marginally
   smaller.
%           Intersections with border points are not
   captured and can
%           lead to under-counting when working with
   surfaces.
%    * epsilon - (default = 1e-5) controls border size
%    * fullReturn - (default = false) controls returned
   variables t, u, v,
%        and xcoor. By default in order to save time, not
   all t, u & v are
%        calculated, only t, u & v for intersections can be
   expected.
%        fullReturn set to true will force the calculation
   of them all.
%
% ALGORITHM:
%  Function solves
%        |t|
%    M * |u| = (o-v0)
%        |v|
%  for [t; u; v] where M = [-d, v1-v0, v2-v0]. u,v are
   barycentric coordinates
```

```
%  and t - the distance from the ray origin in |d| units
%  ray/triangle intersect if u>=0, v>=0 and u+v<=1
%
% NOTE:
%  The algorithm is able to solve several types of problems:
%  * many faces / single ray  intersection
%  * one  face  / many   rays intersection
%  * one  face  / one    ray  intersection
%  * many faces / many   rays intersection
%  In order to allow that to happen all imput arrays are
   expected in Nx3
%  format, where N is number of vertices or rays. In most
   cases number of
%  vertices is different than number of rays, so one of the
   imputs will
%  have to be cloned to have the right size. Use "repmat(A,
   size(B,1),1)".
%
% Based on:
%  *"Fast, minimum storage ray-triangle intersection". Tomas
    Mller and
%    Ben Trumbore. Journal of Graphics Tools, 2(1):21--28,
   1997.
%    http://www.graphics.cornell.edu/pubs/1997/MT97.pdf
%  * http://fileadmin.cs.lth.se/cs/Personal/Tomas_Akenine-
   Moller/raytri/
%  * http://fileadmin.cs.lth.se/cs/Personal/Tomas_Akenine-
   Moller/raytri/raytri.c
%
% Author:
%    Jarek Tuszynski (jaroslaw.w.tuszynski@leidos.com)
%
% License: BSD license (http://en.wikipedia.org/wiki/
   BSD_licenses)

%% Transpose inputs if needed
if (size(orig ,1)==3 && size(orig ,2)~=3), orig =orig' ; end
if (size(dir  ,1)==3 && size(dir  ,2)~=3), dir  =dir'  ; end
if (size(vert0,1)==3 && size(vert0,2)~=3), vert0=vert0'; end
if (size(vert1,1)==3 && size(vert1,2)~=3), vert1=vert1'; end
if (size(vert2,1)==3 && size(vert2,2)~=3), vert2=vert2'; end

%% In case of single points clone them to the same size as
   the rest
N = max([size(orig,1), size(dir,1), size(vert0,1), size(
   vert1,1), size(vert2,1)]);
```

```
if (size(orig ,1)==1 && N>1 && size(orig ,2)==3), orig  =
   repmat(orig , N, 1); end
if (size(dir  ,1)==1 && N>1 && size(dir  ,2)==3), dir   =
   repmat(dir  , N, 1); end
if (size(vert0,1)==1 && N>1 && size(vert0,2)==3), vert0 =
   repmat(vert0, N, 1); end
if (size(vert1,1)==1 && N>1 && size(vert1,2)==3), vert1 =
   repmat(vert1, N, 1); end
if (size(vert2,1)==1 && N>1 && size(vert2,2)==3), vert2 =
   repmat(vert2, N, 1); end

%% Check if all the sizes match
SameSize = (any(size(orig)==size(vert0)) && ...
  any(size(orig)==size(vert1)) && ...
  any(size(orig)==size(vert2)) && ...
  any(size(orig)==size(dir  )) );
assert(SameSize && size(orig,2)==3, ...
  'All input vectors have to be in Nx3 format.');

%% Read user preferences
eps        = 1e-5;
planeType  = 'two sided';
lineType   = 'ray';
border     = 'normal';
fullReturn = false;
nVarargs   = length(varargin);
k = 1;
if nVarargs>0 && isstruct(varargin{1})
  % This section is provided for backward compability only
  options = varargin{1};
  if (isfield(options, 'eps'     )), eps      = options.eps;
           end
  if (isfield(options, 'triangle')), planeType= options.
     triangle; end
  if (isfield(options, 'ray'     )), lineType = options.ray;
           end
  if (isfield(options, 'border'  )), border   = options.
     border;   end
else
  while (k<=nVarargs)
    assert(ischar(varargin{k}), 'Incorrect input parameters'
       )
    switch lower(varargin{k})
      case 'eps'
        eps = abs(varargin{k+1});
        k = k+1;
```

```
      case 'planetype'
        planeType = lower(strtrim(varargin{k+1}));
        k = k+1;
      case 'border'
        border = lower(strtrim(varargin{k+1}));
        k = k+1;
      case 'linetype'
        lineType = lower(strtrim(varargin{k+1}));
        k = k+1;
      case 'fullreturn'
        fullReturn = (double(varargin{k+1})~=0);
        k = k+1;
    end
    k = k+1;
  end
end

%% Set up border parameter
switch border
  case 'normal'
    zero=0.0;
  case 'inclusive'
    zero=eps;
  case 'exclusive'
    zero=-eps;
  otherwise
    error('Border parameter must be either "normal", "
       inclusive" or "exclusive"')
end

%% initialize default output
intersect = false(size(orig,1),1); % by default there are no
    intersections
t = inf+zeros(size(orig,1),1); u=t; v=t;
xcoor = nan+zeros(size(orig));

%% Find faces parallel to the ray
edge1 = vert1-vert0;          % find vectors for two edges
   sharing vert0
edge2 = vert2-vert0;
tvec  = orig -vert0;          % vector from vert0 to ray
   origin
pvec  = cross(dir, edge2,2);  % begin calculating
   determinant - also used to calculate U parameter
det   = sum(edge1.*pvec,2);   % determinant of the matrix M
   = dot(edge1,pvec)
```

```
switch planeType
  case 'two sided'              % treats triangles as two
     sided
    angleOK = (abs(det)>eps); % if determinant is near zero
       then ray lies in the plane of the triangle
  case 'one sided'              % treats triangles as one
     sided
    angleOK = (det>eps);
  otherwise
    error('Triangle parameter must be either "one sided" or
       "two sided"');
end
if all(~angleOK), return; end % if all parallel than no
   intersections

%% Different behavior depending on one or two sided
   triangles
det(~angleOK) = nan;              % change to avoid division
    by zero
u    = sum(tvec.*pvec,2)./det;    % 1st barycentric
   coordinate
if fullReturn
  % calculate all variables for all line/triangle pairs
  qvec = cross(tvec, edge1,2);    % prepare to test V
     parameter
  v    = sum(dir  .*qvec,2)./det; % 2nd barycentric
     coordinate
  t    = sum(edge2.*qvec,2)./det; % 'position on the line'
     coordinate
  % test if line/plane intersection is within the triangle
  ok   = (angleOK & u>=-zero & v>=-zero & u+v<=1.0+zero);
else
  % limit some calculations only to line/triangle pairs
     where it makes
  % a difference. It is tempting to try to push this concept
      of
  % limiting the number of calculations to only the
     necessary to "u"
  % and "t" but that produces slower code
  v = nan+zeros(size(u)); t=v;
  ok = (angleOK & u>=-zero & u<=1.0+zero); % mask
  % if all line/plane intersections are outside the triangle
      than no intersections
  if ~any(ok), intersect = ok; return; end
  qvec = cross(tvec(ok,:), edge1(ok,:),2); % prepare to test
      V parameter
```

```
  v(ok,:) = sum(dir(ok,:).*qvec,2) ./ det(ok,:); % 2nd
     barycentric coordinate
  if (~strcmpi(lineType,'line')) % 'position on the line'
     coordinate
    t(ok,:) = sum(edge2(ok,:).*qvec,2)./det(ok,:);
  end
  % test if line/plane intersection is within the triangle
  ok = (ok & v>=-zero & u+v<=1.0+zero);
end

%% Test where along the line the line/plane intersection
   occurs
switch lineType
  case 'line'      % infinite line
    intersect = ok;
  case 'ray'       % ray is bound on one side
    intersect = (ok & t>=-zero); % intersection on the
       correct side of the origin
  case 'segment'   % segment is bound on two sides
    intersect = (ok & t>=-zero & t<=1.0+zero); %
       intersection between origin and destination
  otherwise
    error('lineType parameter must be either "line", "ray"
       or "segment"');
end

%% calculate intersection coordinates if requested
if (nargout>4)
  ok = intersect | fullReturn;
  xcoor(ok,:) = vert0(ok,:) ...
    + edge1(ok,:).*repmat(u(ok,1),1,3) ...
    + edge2(ok,:).*repmat(v(ok,1),1,3);
end
```

E.2 Fonction "SONAR_PROFILO"

```
function [Distance,Alpha,point_inters]=SONAR_PROFILO(nb,
   vert1,vert2,vert3,JS,jd,X,Alpha,Q_B_0,PRM_SNR_PROF)


Longueur_RAYON = PRM_SNR_PROF.Portee_SNR;
POINT_INIT_Rayon_B = PRM_SNR_PROF.Pos_B';
% DIRECTION_RAYON_B = [0,cos(Alpha),sin(Alpha)];
if nb==1
    DIRECTION_RAYON_B = [1,0,0];
else
    if nb==2
        DIRECTION_RAYON_B = [-1,0,0];
    else
        if nb==3
            DIRECTION_RAYON_B = [0,0,1];
        else
            if nb==4
                DIRECTION_RAYON_B = [0,0,-1];
            else
                if nb==5
                    DIRECTION_RAYON_B = [0,-1,0];
                else
                    if nb==6
                        DIRECTION_RAYON_B = [0,1,0];
                    end
                end
            end
        end
    end
end


POINT_INIT_Rayon_0 = POINT_INIT_Rayon_B + X;
Q_Temp = quatmultiply(quatmultiply(Q_B_0,[0
   DIRECTION_RAYON_B]),quatconj(Q_B_0));
DIRECTION_RAYON_0 = Q_Temp(2:4);
% POINT_ARRIVEE_Rayon_0 = POINT_INIT_Rayon_0 +
   DIRECTION_RAYON_0*Longueur_RAYON;

orig   = POINT_INIT_Rayon_0;                      % ray's origin
dir    = DIRECTION_RAYON_0*Longueur_RAYON ;          % ray's
   direction
%tic;
```

```
% Methode par la plus courte distance
[intersect,t,~,~,xcoor] = TriangleRayIntersection(orig, dir,
    vert1, vert2, vert3,'lineType', 'segment', 'planeType',
   'two sided');

Xxcoor=xcoor(intersect,1); % coordonnees en x des points d'
   intersections
Yxcoor=xcoor(intersect,2); % coordonnees en y des points d'
   intersections
Zxcoor=xcoor(intersect,3); % coordonnees en z des points d'
   intersections
[M,I]=min(t(intersect)); % valeur de la plus courte distance
    et indice du point correspondant


JX=sum(isnan(xcoor(:,1)));
if JX==JS
    J(jd,1)=NaN;%si il ne detecte aucune surface
else
    if JX==(JS-1)
        J(jd,1)=xcoor(intersect,1);
    else
        J(jd,1)=Xxcoor(I,1);
    end
end

JY=sum(isnan(xcoor(:,2)));
if JY==JS
    J(jd,2)=NaN;
else
    if JY==(JS-1)
        J(jd,2)=xcoor(intersect,2);
    else
        J(jd,2)=Yxcoor(I,1);
    end
end

JZ=sum(isnan(xcoor(:,3)));
if JZ==JS
    J(jd,3)=NaN;
else
    if JZ==(JS-1)
        J(jd,3)=xcoor(intersect,3);
    else
        J(jd,3)=Zxcoor(I,1);
    end
```

```
end


% Methode par la moyenne
%{
[intersect,~,~,~,xcoor] = TriangleRayIntersection(orig, dir,
    vert1, vert2, vert3,'lineType', 'segment', 'planeType',
   'two sided');

JX=sum(isnan(xcoor(:,1)));
if JX==JS
    J(jd,1)=NaN;%si il ne detecte aucune surface
else
    if JX==(JS-1)
        J(jd,1)=xcoor(intersect,1);
    else
        J(jd,1)=sum(xcoor(intersect,1))/(JS-JX);
    end
end

JY=sum(isnan(xcoor(:,2)));
if JY==JS
    J(jd,2)=NaN;
else
    if JY==(JS-1)
        J(jd,2)=xcoor(intersect,2);
    else
        J(jd,2)=sum(xcoor(intersect,2))/(JS-JY);
    end
end

JZ=sum(isnan(xcoor(:,3)));
if JZ==JS
    J(jd,3)=NaN;
else
    if JZ==(JS-1)
        J(jd,3)=xcoor(intersect,3);
    else
        J(jd,3)=sum(xcoor(intersect,3))/(JS-JZ);
    end
end
%}

Distance=sqrt((J(jd,1)-POINT_INIT_Rayon_0(1))^2+(J(jd,2)-
   POINT_INIT_Rayon_0(2))^2+(J(jd,3)-POINT_INIT_Rayon_0(3))
```

```
    ^2);

point_inters = J(jd,1:3);
```

Références

[FHo65] Sighard F.Hoerner. *Fluid-dynamic drag.* Sighard F.Hoerner, 1965. URL : https://ia600707.us.archive.org/13/items/FluidDynamicDragHoerner1965/Fluid-dynamic_drag__Hoerner__1965_text.pdf.

[Bak03] Michel Bakalowicz. « Karst et érosion karstique ». In : *Planet Terre* (nov. 2003). ISSN : 2552-9250. URL : https://planet-terre.ens-lyon.fr/ressource/erosion-karstique.xml.

[MT05] Tomas Möller et Ben Trumbore. « Fast, Minimum Storage Ray-Triangle Intersection ». In : *Journal of Graphics Tools* 2 (août 2005). DOI : 10.1145/1198555.1198746.

[Bas06] Myriam Bastard. « Etude de la durabilité de pièces thermoplastiques. Application au polyoxyméthylène. » Theses. Arts et Métiers ParisTech, jan. 2006. URL : https://pastel.hal.science/tel-00076835.

[LSP06] Lionel Lapierre, D. Soetanto et Antonio Pascoal. « Nonsingular Path Following Control of a Unicycle in the Presence of Parametric Modelling Uncertainties ». In : *International Journal of Robust and Nonlinear Control* 16 (juill. 2006), 485â504. DOI : 10.1002/rnc.1075.

[LJ08] Lionel Lapierre et Bruno Jouvencel. « Robust Nonlinear Path-Following Control of an AUV ». In : *Oceanic Engineering, IEEE Journal of* 33 (mai 2008), p. 89-102. DOI : 10.1109/JOE.2008.923554.

[LZ12] Lionel Lapierre et Rene Zapata. « A guaranteed obstacle avoidance guidance system : The safe maneuvering zone ». In : *Autonomous Robots* 32 (avr. 2012). DOI : 10.1007/s10514-011-9269-5.

[Dan21] Huu tho Dang. « Underwater robots for karst and marine exploration : A study ofredundant AUVs ». 2021MONTS038. Thèse de doct. 2021. URL : http://www.theses.fr/2021MONTS038/document.

[Foi+21] Charly Foissac et al. « Vers un serrage intelligent dans les assemblages aéronautiques ». In : *17ème colloque national S-mart AIP-PRIMECA*. Université Polytechnique Hauts-de-France [UPHF]. LAVAL VIRTUAL WORLD, France, mars 2021. URL : https://hal.science/hal-03296145.

[Lap+21] Lionel Lapierre et al. « Karst exploration : Unconstrained attitude dynamic control for an AUV ». In : *Ocean Engineering* 219 (jan. 2021), #108321. DOI : 10.1016/j.oceaneng.2020.108321. URL : https://hal-lirmm.ccsd.cnrs.fr/lirmm-03477919.