Profil Robotique Promotion 2020

UV 5.7 – Architecture robotique

Projet GO-ZONE

Encadrant : Luc JAULIN

2019-2020

Remerciements

Nous remercions Thomas Le Mézo et Benoit Zerr pour leur aide concernant l'étude du Riptide et Simon Rohou pour son aide concernant la librairie TUBEX. Nous remercions également Henrique Fagundes Gasparoto pour son soutien lors de l'expérimentation menée au Lac de Ty Colo. Enfin nous remercions Luc Jaulin pour la mise en place de ce projet et son aide tout au long des séances.

Résumé

Chaque année l'ensemble de la promotion du profil robotique travaille sur un projet commun. Cette année il s'agissait de permettre à des robots sous-marins autonomes (Autonomous Underwater Vehicle - AUV) de couvrir une zone en utilisant seulement des suivis de cap consécutifs en fonction d'une mesure de profondeur du fond marin. Nous parlons ainsi de cycles stables. Le projet fût donc intitulé Go Zone et avait pour but d'aboutir sur une preuve de concept pouvant être réutilisée par la suite dans la recherche de la Cordelière.
Afin de mener à bien cet objectif, nous avons du répartir le travail en 12 tâches interdépendantes. Le travail consistait donc à définir cette notion de cycle stable, à utiliser ainsi que modéliser les AUV à dispositions et enfin à implémenter des cycles stables au sein des AUV. Une fois ce travail effectué nous avons mené à bien une expérimentation au lac de Ty Colo prouvant ainsi ce concept de navigation.

Abstract

Each year the whole promotion of the robotic profile works on a common project. This year it was to allow autonomous underwater robots (Autonomous Underwater Vehicle - AUV) to cover an area using only consecutive heading tracking based on a seabed depth measurement. We are talking about stable cycles. The project was therefore entitled Go Zone and aimed to lead to a proof of concept that could be reused later in the search for the Cordière.
In order to carry out this objective, we had to divide the work into 12 interdependent tasks. The work therefore consisted in defining this notion of stable cycle, in using as well as modeling the available AUVs and finally in implementing stable cycles within the AUVs. Once this work was done, we carried out an experiment at Lake Ty Colo, thus proving this concept of navigation.

Table des matières

Table des figures

Annexes

Chapitre 1

Introduction

Dans le cadre de l'UV 5.7 Architecture Robotique, l'ensemble de la promotion du profil robotique travaille sur un projet commun. Cette année ce projet a été intitulé "Go Zone". Il s'agit de prouver que des robots sous-marins autonomes (Autonomous Underwater Vehicle - AUV) peuvent couvrir une zone en parcourant des "cycles stables". La réalisation d'un cycle stable par un AUV consiste à lui faire effectuer des suivis de cap consécutifs en fonction d'une mesure de profondeur du fond marin. Cette notion de cycle stable sera explicitée dans la suite de ce rapport.

Afin de mener à bien ce projet, 11 séances pour un total de 56 heures ont été dédiées. Le travail à effectuer a du être divisé en différentes tâches afin d'optimiser le temps qui nous a été imparti. De plus, les AUV à notre disposition sont des AUV de la société Riptide ainsi que le Folaga de la société Kopadia.

Ce présent rapport présentera ainsi le travail effectué par l'ensemble de la promotion et décrira également une expérimentation constituant l'aboutissement des travaux menés conjointement.

Chapitre 2

Gestion des tâches à réaliser

2.1 Identification des tâches à réaliser

Le projet étant réalisé par 18 personnes, il a été nécessaire de répartir le travail sous forme de tâches. C'est tâches ont été identifiées lors des premières séances. Il en est de même pour leur attribution aux membres du groupe. De plus, une personne a été désignée comme architecte du projet. Son rôle étant de s'assurer de l'avancement général du projet en ayant une vue globale ainsi que de donner les directives permettant à tous d'avancer dans le même sens sans redondances. De plus il soutien les différents groupes et participe aux réflexions lors du choix d'une solution technique
Voici la liste des tâches ayant été réalisées :

- **Tâche 1 : Simulation de la commande**
  Le but est de simuler le principe de l'automate sous Python ainsi que d'implémenter les équations d'état des AUV.
- **Tâche 2 : Utilisation des AUV Riptide**
  Le but est de savoir utiliser le Riptide en tant qu'utilisateur et de comprendre son fonctionnement.
- **Tâche 3 : Utilisation du Folaga**
  Le but est de savoir utiliser le Folaga en tant qu'utilisateur et de comprendre son fonctionnement.
- **Tâche 4 : Prise d'information sur les capteurs**
  Le but étant d'identifier les capteurs nécessaires à la réalisation du projet ainsi que d'obtenir la documentation technique associée.
- **Tâche 5 : Définition des cycles stables**
  Le but étant, à partir d'une bathymétrie connue, de définir des zones constituées de cycles stables de façon la plus automatique possible.
- **Tâche 6 : Simulateur 3D**
  Le but étant de mettre en place d'un simulateur 3D pour le rejeu d'une mission ou bien pour visualiser le comportement de l'automate sur un modèle dynamique.
- **Tâche 7 : Intelligence de la gestion des zones**
  Le but étant de faire un algorithme de graphe capable de répondre à une requête imprévue du type aller à la zone C, visiter tous les noeuds sans jamais passer par la zone

D.

— **Tâche 8 : Utilisation du Tubex**
Le but étant de prouver la stabilité des cycles stables via Tubex ainsi que les liens entre les différentes zones.

— **Tâche 9 : Identification des paramètres des AUV**
Le but étant de déterminer les paramètres physiques des AUV mis en évidence par la modélisation des AUV.

— **Tâche 10 : Gestion des AUV sous ROS**
Le but étant de permettre le contrôle et la mise en place d'une mission via ROS.

— **Tâche 11 : Modélisation des AUV**
Le but étant de déterminer les équations d'états qui régissent les AUV.

— **Tâche 12 : Expérimentation**
Le but étant de mener une expérimentation pour mettre à l'épreuve le travail ayant été réalisé dans ce projet.

La numérotation des tâches n'a ni de signification chronologique, ni même de signification prioritaire. Ainsi certaines des tâches sont étroitement liées entre elles.

2.2 Répartition des tâches

Voici sous forme de tableau la répartition des tâches suivant les membres du groupe :

Tâche	**Membres**
Tâche 1	Matthieu Bouveron, Clément Bichat
Tâche 2	Philibert Adam, Nathan Fourniol, Alexandre Courjaud
Tâche 3	Corentin Jegat, Yann Musellec
Tâche 4	Philibert Adam, Matthieu Bouveron, Driss Tayebi
Tâche 5	David Brellmann, Aurelien Lebrun, Alexandre Argento
Tâche 6	Cyril Cotsaftis, Quentin Cardinal
Tâche 7	Anouar Mahla, Axel Porlan
Tâche 8	Aurelien Grenier, Driss Tayebi
Tâche 9	Erwann Landais, Yann Musellec
Tâche 10	Nathan Fourniol, Corentin Jegat, Alexandre Courjaud, Philibert Adam
Tâche 11	Quentin Cardinal, Cyril Cotsaftis, Erwann Landais
Tâche 12	Kévin Bedin
Architecte	Kévin Bedin

2.3 Lien entre les tâches

Voici sous forme schématique les relations identifiées entre les différentes tâches :

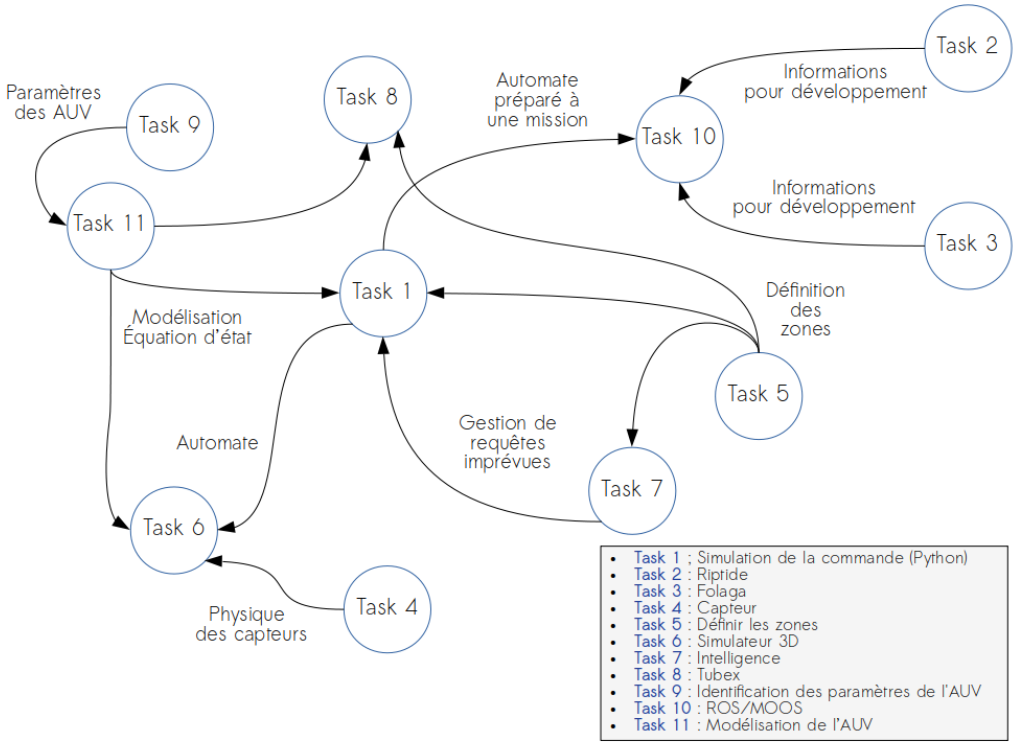

FIGURE 2.1 – Interdépendance des tâches

De part ces interdépendances, ce présent rapport ne sera pas constitué d'une partie par tâche.

2.4 Agencement des tâches dans ce rapport

Chaque partie pourra être constituée d'une ou plusieurs tâches. Cela permet de rendre le rapport plus cohérent et plus logique dans son agencement.
Voici sous la forme d'un tableau la répartition des tâches au sein des parties de ce rapport :

Tâche	**Partie**
Tâche 1	3.4.1 Simulation de la commande
Tâche 2, Tâche 4 (Riptide), Tâche 10 (Riptide)	3.1.1 Utilisation de l'AUV uuV de Riptide
Tâche 3, Tâche 4 (Folaga), Tâche 10 (Folaga)	3.1.1 Utilisation de l'AUV Folaga de Kopadia
Tâche 5	3.3.1 Définition des cycles stables
Tâche 6	3.4.2 Simulateur 3D
Tâche 7	3.3.2 Intelligence
Tâche 8	3.3.3 Tubex
Tâche 9	3.2.2 Identification des paramètres des AUV
Tâche 11	3.2.1 Modèles des AUV
Tâche 12	4 Expérimentation

2.5 Outils utilisés pour le développement du projet

Afin de mener à bien le projet nous avons utilisé la plateforme collaborative GitLab. Chaque tâche s'est vue dédier un répertoire GitLab au sein du projet global Go Zone. Les droits d'accès ont été aménagés en fonction des répartitions du travail au sein de la promotion. Cette méthode de fonctionnement a permis à l'architecte d'avoir une vue globale sur l'avancée du projet et un développement AGILE des différents travaux. Enfin vous trouverez en annexe le Diagramme de Gantt décrivant l'avancée du projet.

Chapitre 3

Travail réalisé

3.1 Utilisation des AUV

3.1.1 Utilisation de l'AUV MKII microUUV de Riptide

3.1.1.1 Riptide Autonomous Solutions

Riptide Autonomous Solutions est une entreprise américaine axée sur le développement et la production de véhicules maritimes sans pilote (UMV), véhicules sous-marins autonomes (AUV), véhicules sous-marins sans pilote (UUV) et de véhicules de surface autonomes (ASV). Elle a été rachetée par BAE Systems en Juin 2019.

Leurs produits sont présentés comme techniquement supérieurs, à prix nettement inférieur par rapport à la concurrence, telle que Edge Tech. L'électronique, les logiciels open source ainsi que les techniques de fabrication récentes sont sensées avoir pour résultat de produire de meilleures performances. Leurs produits, au premier coup d'oeil simples et flexibles, sont sensé être personnalisables pour adapter la charge utile ou les paramètres opérationnels. Leur gamme d'AUV est variée et permet l'ajout d'un sonar, de DVL ou encore d'une antenne Iridium selon les besoins.

3.1.1.2 Le Riptide MKII microUUV

Le premier AUV utilisé est le Riptide MKII microUUV de Riptide Autonomous Solutions. Pour des soucis d'écriture, celui-ci sera appelé "Riptide" dans la suite de ce rapport. Il s'agit d'un AUV low-cost modulable. L'ENSTA Bretagne en possède quatre modèles basiques, non utilisés jusque là.

FIGURE 3.1 – Options de modules Riptide

Le Riptide peut être divisé en trois parties distinctes :

— la tête (navigation)
— le payload central (alimentation)
— la queue (actionneurs)

La tête contient tous les capteurs utiles à la navigation ainsi que une grande partie de l'électronique du système. Le Riptide est équipé d'une IMU, d'un capteur de température/-pression et d'un écho-sondeur. Nous n'avons pas accès à la documentation de ces capteurs, ce qui nous pose de sérieux problèmes. Une carte Beaglebone Black sert de PC embarqué et gère les communications ainsi que le contrôle de l'AUV. La plupart des communications se font par I2C, or sans documentation il est difficile d'obtenir les adresses des capteurs et donc d'avoir accès à leur données sans passer par l'interface proposée. De plus, pour éviter toute influence électro-magnétique de la part de la batterie, un I2C différentiel a été mis en place, ce qui a pour effet de rendre presque impossible toute tentative d'espionnage des bus.

La queue abrite le moteur principal ainsi que les trois ailerons de direction, disposés à 120 degrés les uns de autres. L'aileron dorsal contient aussi une antenne et son module GPS ainsi que la clé WiFi permettant d'y accéder à distance.

Le payload central contient quand à lui les packs de batterie ainsi que les poids nécessaires pour la flottabilité et l'équilibrage. Pour l'instant, il s'agit de 144 piles A4 permettant d'obtenir une autonomie de 15/20 heures selon utilisation.

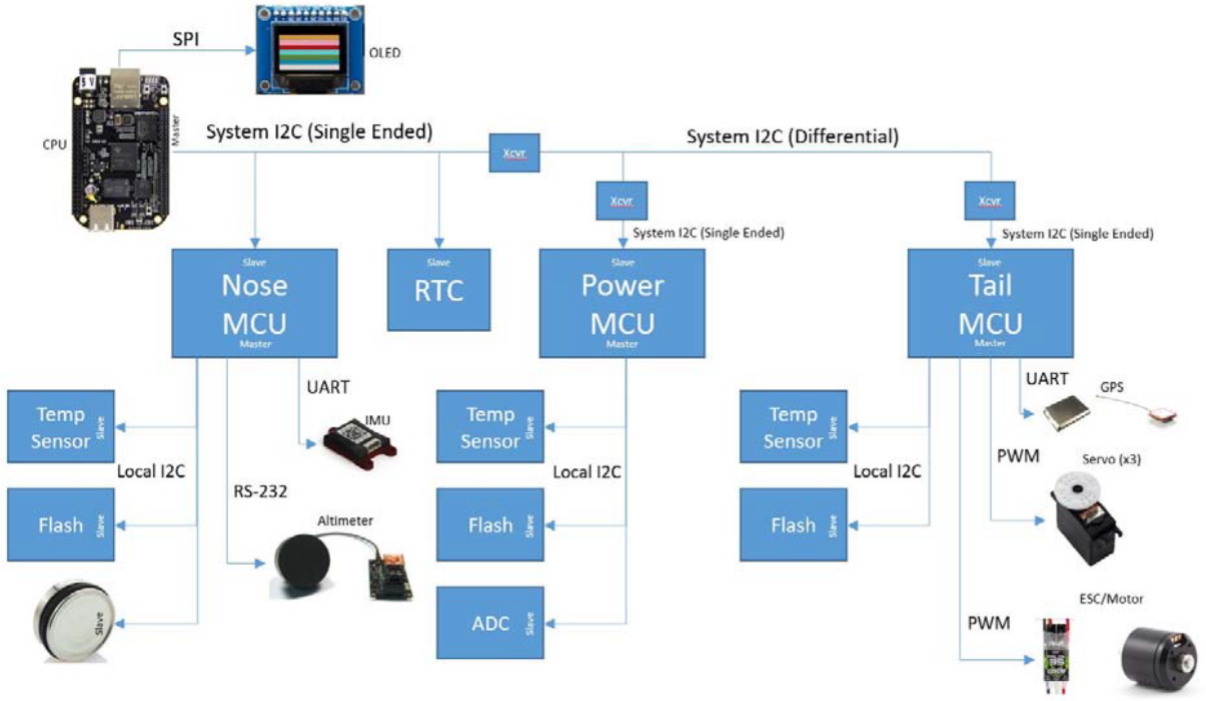

FIGURE 3.2 – Architecture électronique du Riptide

De plus, un écran de status se situe au niveau de la tête du Riptide. Il permet un aperçu rapide de l'état de l'AUV, des données des capteurs ainsi que son adresse IP sur le réseau. Il est allumé en permanence, même en mission à 300m de fond.

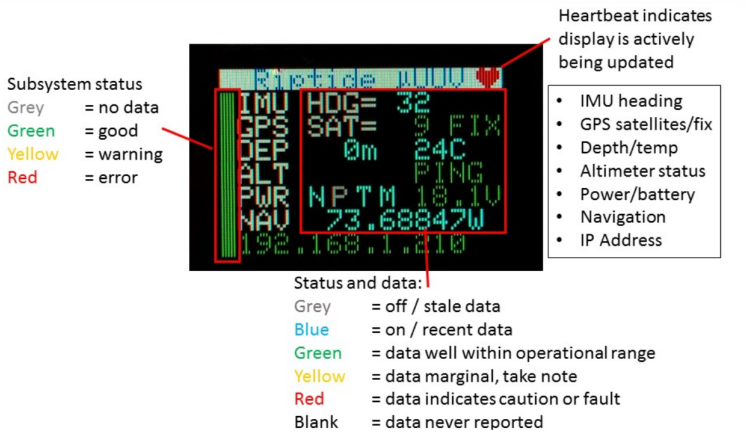

FIGURE 3.3 – Ecran de status du Riptide

Dans le cadre du projet GoZone, il nous faut un contrôle total des actionneurs ainsi que l'accès aux données des capteurs de navigation. Un contrôleur PID est déjà installé sur les Riptides et seule la commande en cap et en vitesse est nécessaire. Le contrôle se fait à travers trois couches, du haut au bas niveau. Il s'effectue par flux «down» des couches supérieures, avec le statut et les données «up» des couches inférieures. En l'état, il n'y a qu'un accès à la couche "Mission Control" par le biais d'une interface web présentée par la suite.

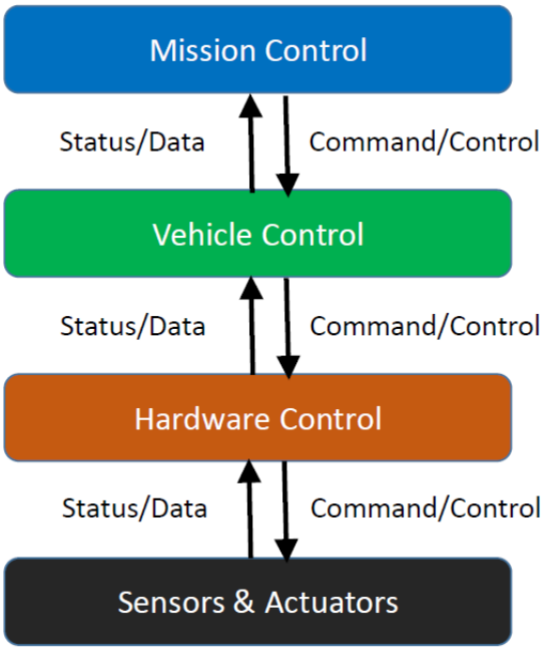

FIGURE 3.4 – Architecture de contrôle du Riptide

Pour son fonctionnement, avec les paramètres d'usine, la BeagleBone Black fait tourner un système d'exploitation Debian. Sur celui-ci est installé une couche supplémentaire d'abstraction avec le middleware MOOS (Mission Oriented Operating System) ainsi que tous les scripts binaires décrivant les différentes couches et permettant le bon déroulement des missions. En effet, cet appareil est vendu pour pouvoir effectuer des missions de l'ordre d'une suite finie de consignes comme rejoindre une certaine profondeur, suivre un cap ou alors une liste de poins GPS à suivre. Pour les définir, l'entreprise a prévu une page web à laquelle on peut accéder une fois connecté au même WiFi que le Riptide en renseignant l'adresse IP de l'AUV dans un navigateur. Ainsi, on obtient une page qui comporte un onglet pour définir les missions, "Mission Manager" mais aussi un pour la visualisation des valeurs des capteurs et les test unitaires du Riptide, "Checkout". Un troisième onglet, "MOOS Variables" permet de visualiser toutes les variables définies dans MOOS pour le bon fonctionnement du robot.

FIGURE 3.5 – Interface de gestion des missions

FIGURE 3.6 – Interface de test et de visualisation des données renvoyées par les capteurs

3.1.1.3 Passage de MOOS à ROS

Nous avons choisis de développer ce projet sous ROS. Il a donc fallu mettre au point une solution de développement ROS sur le Riptide. Pour cela nous avions deux solutions :

— Re-coder totalement le robot avec ROS
— Mettre en place un bridge entre MOOS et ROS

La première solution apporte l'avantage d'avoir la main sur tous les niveaux de développement de l'AUV alors que le second permet justement de ne pas avoir à refaire les régulateurs bas niveau déjà implémentés sous MOOS. Cependant, nous avons rapidement abandonné l'idée de réécrire les traitements bas niveau avec ROS par manque de documentation disponible. Nous avons essayé d'espionner les communications I2C grâce à un oscilloscope pour reproduire les commandes envoyées avec les programme MOOS, cependant nous n'avons pas obtenu de résultats exploitables de cette manière.

Nous nous sommes donc penché sur le bridge entre MOOS et ROS. Pour cela nous avons choisi de nous servir des commandes MOOS compilées sur le Riptide :

— uPoke permet de remplacer une valeur dans la base de donnée MOOS depuis un terminal.
— uXMS permet de lire la valeur d'une ou de plusieurs valeurs de la base de données depuis un terminal.

Nous avons donc réalisé deux noeuds ROS. Le premier consiste à récupérer les commandes en cap, tangage, roulis, vitesse et profondeur depuis des topics ROS puis à écrire ces valeurs dans la base de donnée MOOS grâce à uPoke, ainsi les régulateurs MOOS prennent le relais et envoient les commandes vers les moteurs.

Le deuxième utilise la commande uXMS pour obtenir les valeurs des capteurs depuis la base de donnée MOOS pour les renvoyer vers des topics ROS.

3.1.1.4 Résultat

En fin de projet les résultats de ce bridge ne sont pas satisfaisant :
— La commande uPoke n'arrive pas toujours à écrire des valeurs dans la base de données et une absence de commande pendant 3 secondes remet les moteurs à zéro.
— La récupération du résultat de uXMS pour extraire la valeurs des variables n'est pas assez performante, ce qui entraîne un retard qui s'accumule au cours du temps.

Nous n'avons donc pas pu réaliser de test en conditions réelles avec ROS sur le Riptide.

3.1.2 Utilisation de l'AUV Folaga de Kopadia

Le Folaga étant déjà connu de l'ENSTA Bretagne, nous ne présenterons donc pas ses caractéristiques ainsi que l'entreprise Kopadia comme nous l'avons fait pour les microUUV de Riptide. Ici, seule la partie technique développée et utilisée lors de ce projet sera décrite.

3.1.2.1 Le bas niveau : communication entre la raspberry et l'ordinateur Graaltech

Ce qui est défini par "bas niveau" est l'ensemble des outils/codes de communication entre les capteurs et/ou les actionneurs. Toute cette partie est réalisée par l'ordinateur embarqué présent dans l'AUV. Néanmoins, pour acquérir une valeur voulue sur la carte Raspberry où se

situe notre architecture ROS, il est nécessaire d'interroger l'ordinateur Graaltech. Ainsi, un code permet d'aller chercher les données capteurs et un autre sert à envoyer des commandes moteurs ou des commandes de calibration. Pour obtenir les valeurs de l'echosondeur, la procédure est différente. Il est branché directement en USB sur la carte Raspberry. Un troisième scripte s'occupe alors de lire les valeurs renvoyées sur ce port.

3.1.2.2 Calibration ou propulsion, il faut choisir.

Une des contraintes du Folaga réside dans le choix de piloter les moteurs et les pompes (pour la propulsion) ou de faire varier la position du pack batterie et le remplissage du balaste (pour la calibration et l'équilibrage). Les deux choix ne pouvant pas être simultanés, un programme s'occupe de tenir compte ou non des données de propulsion et de calibration qu'il reçoit en fonction du mode dans lequel il est.

3.1.2.3 Le mode manuel, un ajout incontournable

Avant de commencer les explications, il est important de préciser que le contrôle mauel n'est disponible qu'en surface car il utilise l'interface WiFi. Après avoir codé le bas niveau, un besoin s'est rapidement fait sentir : contrôler le Folaga en mode manuel. L'exemple le plus parlant est de pouvoir faire revenir l'AUV à un point donné après l'exécution d'une mission. Cela permet également de reprendre la main sur l'AUV à tout instant si le comportement n'est pas celui attendu, d'amener l'AUV dans une certaine zone, etc. Pour contrôler le Folaga le choix a été d'utiliser une manette. Avoir deux joysticks, des gâchettes et des boutons est très pratique afin d'utiliser de l'analogique et du on/off. A noter qu'il est possible de switcher entre le mode propulsion et calibration à la manette afin de réaliser une calibration manuelle.

3.1.2.4 Asservissement en cap, du capteur au contrôleur.

Dans cette partie, on part du principe que l'on ne connaît pas parfaitement les équations d'évolution du Folaga en fonction de ses actionneurs. De plus, aucun gyroscope est à notre disposition, c'est à dire que la vitesse de rotation n'est pas connue. C'est pourquoi il a fallut réaliser un filtre de Kalman basé sur les données récupérées lors d'essais pour lisser les données capteurs et estimer la vitesse de rotation. Cela donne le filtre de Kalman étendu suivant comme estimateur du cap et de sa dérivée :

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \sim \begin{bmatrix} \cos\theta \\ \sin\theta \\ \dot{\theta} \end{bmatrix}, x_{k+1} = \begin{bmatrix} x_1 - dt * x_3 * x_2 \\ x_2 + dt * x_3 * x_1 \\ x_3 \end{bmatrix}, y = \begin{bmatrix} \cos\theta_{mes} \\ \sin\theta_{mes} \\ sawtooth(\theta_{mes} - \theta_{mes-1})/dt \\ 1 \end{bmatrix} \sim \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_1^2 + x_2^2 \end{bmatrix}, \tag{3.1}$$

Contrôleur :

On peut désormais réaliser un correcteur proportionnel dérivé en sachant que le terme dérivé ne sera plus aberrant. La consigne envoyée aux pompes suit la loi suivante :

$$u = k_p * sawtooth(\theta_{consigne} - \theta_{kaman}) + k_d * \dot{\theta}_{kalman} \tag{3.2}$$

Le terme proportionnel va permettre à l'AUV de converger vers la consigne et le terme dérivé va permettre d'anticiper la forte inertie du Folaga afin d'éviter les oscillations autour de la valeur souhaitée.

3.1.2.5 API ROS développée

Voici l'architecture ROS développée permettant d'interagir avec le Folaga :

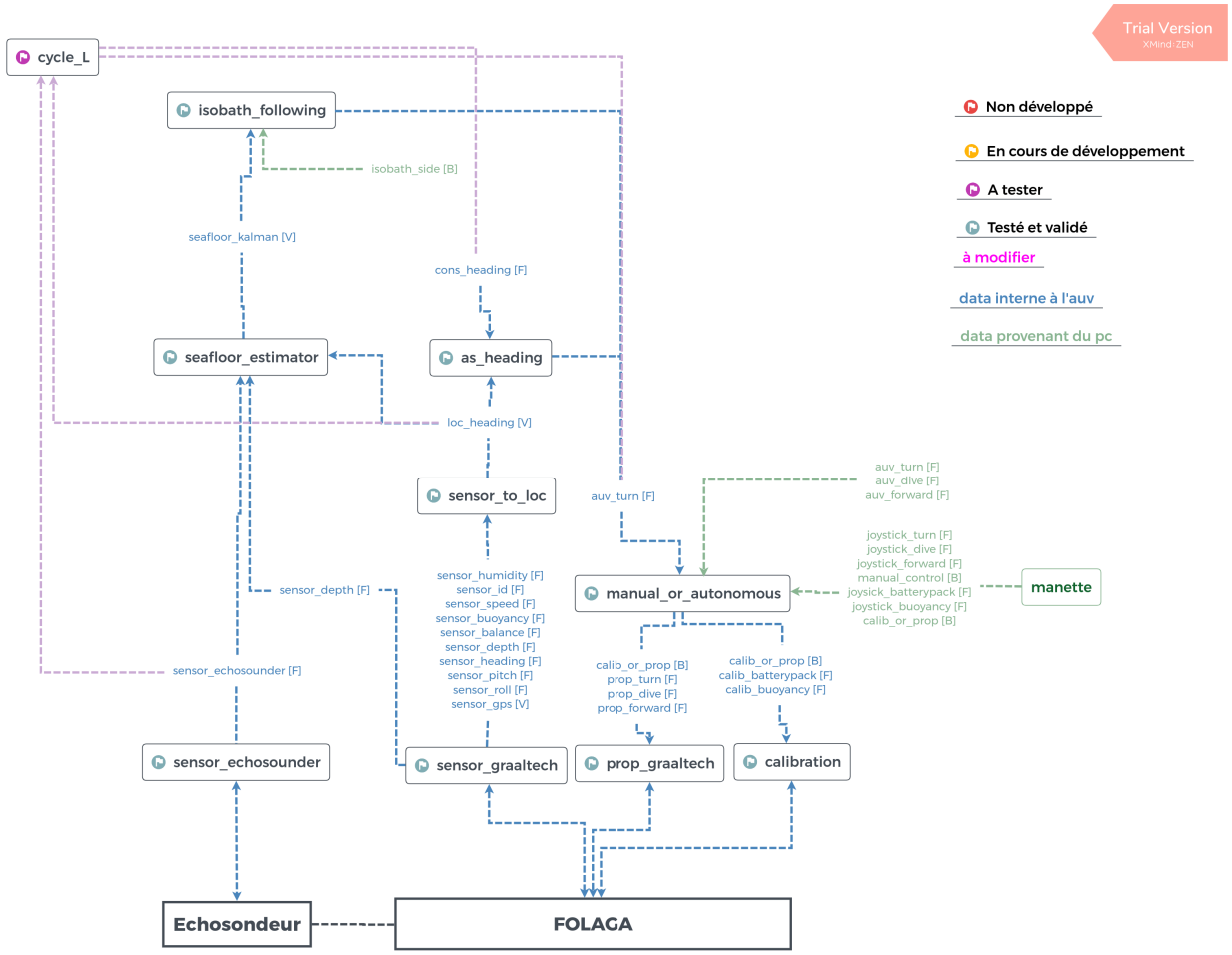

FIGURE 3.7 – Architecture ROS du Folaga

3.2 Modélisation des AUV

3.2.1 Modèles des AUV

3.2.1.1 Choix du modèle pour le Riptide

Les actionneurs du Riptide sont les suivants : une hélice à l'arrière pour la propulsion, trois servomoteurs contrôlant les ailerons, placés à 120° les uns des autres à l'arrière du Riptide. Pour notre application, on ne considère pas la capacité du Riptide à se déplacer suivant l'axe z.

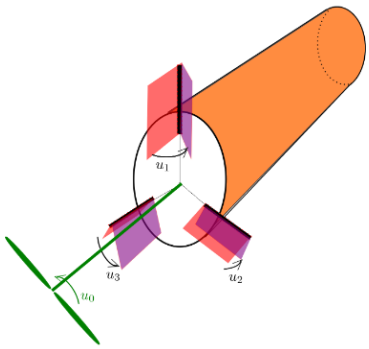

FIGURE 3.8 – Modélisation du Riptide avec ses actionneurs

D'après les équations de Fossen [2], la force de propulsion exercée par l'hélice est proportionnelle au carré de la commande envoyée ; le coefficient de proportionnalité dépendant alors notamment du diamètre de l'hélice et de la masse volumique de l'eau. On suppose aussi que la force de frottement de l'eau sur le Folaga est un frottement quadratique. En appliquant le principe fondamental de la dynamique au système Riptide, dans le référentiel lié à celui-ci, on a alors :

$$\begin{aligned}(m + m_a) * a &= \alpha * u_0 * |u_0| - \beta * v_x * |v_x| \\ \iff a &= p_1 * u_0 * |u_0| - p_2 * v_x * |v_x|\end{aligned}$$

Avec p_1 et p_2 des constantes dépendant de la surface normale du Riptide à l'axe x, des coefficients de frottement de l'eau, et de la masse ajoutée m_a de l'eau.

Pour la rotation, on sait que chacun des ailerons du Riptide possède une force de trainée et de portance. On choisit ici de négliger la force de traînée pour la force de portance, dépendante de la vitesse du Riptide selon l'axe x et contrôlant la rotation de l'AUV.
D'après [2], la force de portance d'un aileron est de la forme :

$$F_{pa} = \gamma * v * \sin u * \vec{ax}$$

Avec $\vec{ax}$ l'axe de l'aileron considéré, et u l'angle entre cet aileron autour de cet axe.
Puisque la commande des ailerons ne permet que de faibles variations d'angle, on se place ici dans l'approximation des petits angles : on considère donc $u \equiv \sin u$

En exprimant chacun des ailerons dans le référentiel associé au Riptide, on a alors :

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = v.B. \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{3.3}$$

$$\text{avec } B = \begin{pmatrix} p_3 & 0 & 0 \\ 0 & p_4 & 0 \\ 0 & 0 & p_4 \end{pmatrix} . \begin{pmatrix} -1 & -1 & -1 \\ 0 & \sin(2\pi/3) & -\sin(2\pi/3) \\ 1 & \cos(2\pi/3) & \cos(2\pi/3) \end{pmatrix} ;$$

p_3 et p_4 dépendent ici des paramètres d'inertie du Riptide (masse, rayon, ...). On notera qu'on choisit ici qu'on approxime l'inertie du Riptide comme correspondant à celle d'un cylindre plein suivant l'axe x, ce qui explique la forme diagonale de la matrice et la répétition de p_4.

Finalement, en utilisant les équations de changement de référentiel [3] entre le référentiel du Riptide et le référentiel observateur, le modèle global associé au Riptide est le suivant :

$$\begin{cases} \dot{p} = R(\phi, \theta, \psi) * (v_x, 0, 0)^T \\ \dot{v}_x = p_1 * u_0 * |u_0| - p_2 * v_x * |v_x| \\ \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \tan(\theta).\sin(\phi) & \tan(\theta).\cos(\phi) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \dfrac{\sin(\phi)}{\cos(\theta)} & \dfrac{\cos(\phi)}{\cos(\theta)} \end{bmatrix} .v_x.B(p3, p4). \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \end{cases} \tag{3.4}$$

3.2.1.2 Choix du modèle pour le Folaga

Les actionneurs du Folaga sont les suivants : une hélice à l'arrière pour la propulsion, quatre pompes sur les côtés pour la rotation, et quatre pompes pour la plongée ou la remontée. Pour notre application, on ne s'intéressera qu'aux pompes utilisées pour la rotation et à l'hélice. Les codes bas-niveaux du Folaga gérant directement l'activation des quatre pompes pour une commande en rotation donnée, on peut donc réduire la commande en rotation à une commande u_1.
Ayant une forme de torpille, l'AUV ne peut se déplacer que suivant son axe x en propulsion. Il garde cependant trois degrés de liberté en rotation sur chacun de ses axes.
D'après [2], la force de propulsion exercée par une hélice est proportionnelle au carré de la commande envoyée. On supposera qu'il en est de même pour la force de pression exercée par les pompes motorisées du Folaga. On a donc :

$$F_{px} = \alpha * u_0 * |u_0|$$
$$F_{rz} = p_3 * u_1 * |u_1|$$

Avec F_{px} la force de propulsion selon x, et F_{rz} la force de rotation selon z.

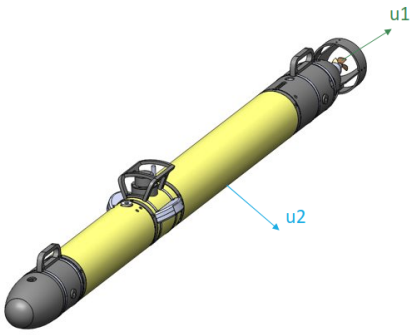

FIGURE 3.9 – Modélisation du Folaga avec ses actionneurs

Dès lors, et avec la même démarche que pour le Riptide, on utilise le modèle suivant pour modéliser le Folaga :

$$\begin{cases} \dot{p} = R(\phi, \theta, \psi) * (v_x, 0, 0)^T \\ \dot{v}_x = p_1 * u_0 * |u_0| - p_2 * v_x * |v_x| \\ \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \tan(\theta).\sin(\phi) & \tan(\theta).\cos(\phi) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \dfrac{\sin(\phi)}{\cos(\theta)} & \dfrac{\cos(\phi)}{\cos(\theta)} \end{bmatrix} . \begin{bmatrix} w_{rx} \\ w_{ry} \\ w_{rz} \end{bmatrix} \\ \begin{bmatrix} \dot{w}_{rx} \\ \dot{w}_{ry} \\ \dot{w}_{rz} \end{bmatrix} = (I^{-1}). \begin{bmatrix} 0 \\ 0 \\ p_3 * u_1 * |u_1| - p_4 * w_{rz} * |w_{rz}| \end{bmatrix} \end{cases} \tag{3.5}$$

Avec I la matrice d'inertie de l'AUV, u_0 la commande en propulsion sur l'hélice, et u_1 la commande en rotation selon z. De même que pour le Riptide, les coefficients p_1 et p_2 dépendent de la surface normale du Folaga à l'axe x, des coefficients de frottement de l'eau, et de la masse ajoutée de l'eau.

On se place pour cette application dans l'hypothèse où le Folaga n'a qu'une rotation selon l'axe z. On suppose également qu'il est parfaitement équilibré, et qu'il ne peut pas se déplacer selon l'axe y (pas de roulis ou de tangage). On suppose enfin (au vu de la forme cylindrique de l'AUV) que la matrice d'inertie de celui-ci correspond à une matrice diagonale. On peut alors simplifier le modèle de la façon suivante :

$$\begin{cases} \dot{p} = R(\phi, \theta, \psi) * (v_x, 0, 0)^T \\ \dot{v}_x = p_1 * u_0 * |u_0| - p_2 * v_x * |v_x| \\ \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} w_{rx} \\ w_{ry} \\ w_{rz} \end{bmatrix} \\ \begin{bmatrix} \dot{w}_{rx} \\ \dot{w}_{ry} \\ \dot{w}_{rz} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ p_5 * u_1 * |u_1| - p_6 * w_{rz} * |w_{rz}| \end{bmatrix} \end{cases} \tag{3.6}$$

Avec p5 et p6 correspondant respectivement à p3 et p4, multipliés par les termes de la matrice d'inertie de l'AUV.

3.2.2 Identification des paramètres des AUV

3.2.2.1 Détermination des paramètres dynamiques des AUV

Afin d'obtenir une simulation conforme à la réalité et à notre modèle, il est nécessaire de déterminer des protocoles pour obtenir les coefficients en propulsion et en frottement sur chacun des axes de l'AUV. Soit p_{prop} et p_{frot} les coefficients de propulsion associés à une commande et de frottement sur l'un des axes de déplacement de l'AUV (soit selon l'axe x, soit en rotation).

On a alors :

$$a = p_{prop} * u * |u| - p_{frot} * V * |V| \tag{3.7}$$

Avec a, V et u correspondant respectivement à l'accélération, la vitesse et la commande envoyée pour le déplacement considéré (rotation ou propulsion selon x).

Plusieurs méthodes ont été envisagées pour déterminer les paramètres dynamiques des AUV. La méthode exploitée dans ce rapport se base sur le protocole suivant :

1. Donner une commande constante à l'AUV pendant un certain temps, afin d'atteindre le régime permanent en vitesse. L'AUV atteint alors la vitesse V_{cst}
2. Couper la commande, attendre un certain temps afin d'être sûr d'atteindre une vitesse nulle.

Au cours de la partie 2, seuls les frottements s'appliquent sur l'AUV. On a donc :

$$-p_{frot}.V.|V| = a = \dot{V}$$

En intégrant, on obtient alors :

$$V(t) = \frac{V_{cst}}{V_{cst}.p_{frot}.t + 1} \tag{3.8}$$

Avec V_{cst} la vitesse de l'AUV à l'instant où on coupe la commande.

En intégrant une dernière fois, on a enfin :

$$M(t) = M(0) + \frac{log(V_{cte}.p_{frot}.t + 1)}{p_{frot}} \tag{3.9}$$

Avec M(t) la position associée à l'AUV à un instant t. Cette position correspond soit à un angle pour une commande en rotation, soit à un déplacement suivant x pour une commande en propulsion.
En connaissant V_{cst}, puis M(t) (ou V(t)), on peut alors en déduire p_{frot}. Deux méthodes ont été exploitées ici pour obtenir ce coefficient : soit avec un algorithme de minimisation, soit en utilisant un algorithme basé sur la théorie des intervalles.

L'étape suivante est ensuite d'obtenir p_{prop}. Pour cela, on peut utiliser l'estimation de p_{frot} et les données de la première partie. En effet, on a alors :

$$p_{prop} * u * |u| - p_{frot} * V * |V| = a \tag{3.10}$$

On peut autrement utiliser le fait qu'avec une commande constante, l'accélération de l'AUV est nulle à partir d'un certain temps. Dans ce cas, on a alors la relation :

$$p_{prop} * u * |u| - p_{frot} * V * |V| = 0 \tag{3.11}$$

A l'aide des données V issues de la première partie, on peut obtenir une estimation de p_{prop}. Là aussi, ce coefficient peut être obtenu soit via un algorithme de minimisation, soit via un algorithme basé sur la théorie des intervalles. Dans ce dernier cas, en connaissant l'intervalle de p_{prop}, on retrouve p_{frot} en calculant pour toutes les données avec une accélération nulle l'intervalle dans lequel rentre p_{frot}. On utilise donc l'équation (3.11).

Grâce à l'analyse par intervalle, une autre approche était possible pour calculer les coefficients. Au lieu de calculer d'abord p_{prop} puis p_{frot}, on peut calculer directement les deux coefficients en utilisant directement l'équation (3.10). Pour cela, on estime la valeur de l'accélération et construire un intervalle pour cette valeur.

On peut aussi compléter l'obtention de p_{prop} en mesurant la force déployée par l'AUV pour une commande constante, à l'aide de dynamomètres.
Pour la propulsion, celui-ci peut être fixé sur l'arrière de l'AUV. Pour la rotation, celui-ci peut être placé sur une extrémité de l'AUV, l'autre extrémité devant être fixé à un bâti.

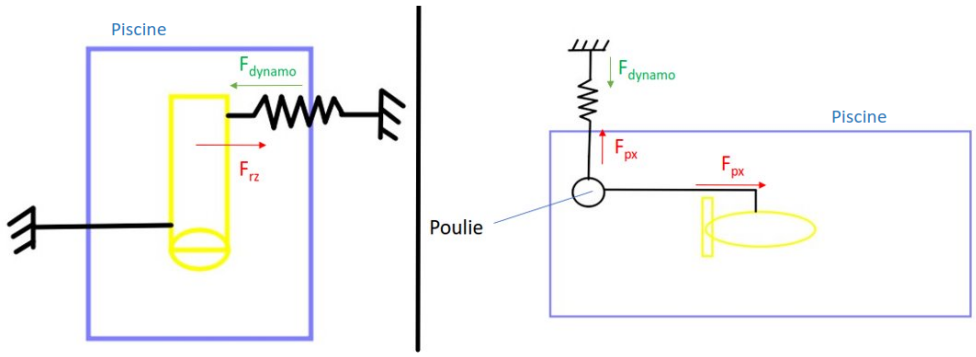

FIGURE 3.10 – Schémas pour la mesure du coefficient de propulsion en rotation (gauche) et suivant x (droite) d'un AUV

3.2.2.2 Exploitation des données

En utilisant les équations (3.8), (3.9) et (3.10), on peut utiliser les données prises sur le Folaga pour déterminer ces coefficients.

Dans le cas du Folaga, les données sont enregistrées dans différents topics ROS. Pour les coefficients en rotation, on utilise les données concernant la commande en rotation envoyée à l'AUV ($/auv\_turn$), le cap de l'AUV ($/sensor\_heading$), le cap et la vitesse de l'AUV estimée par Kalman ($/loc\_heading$), ainsi que la reprise en main éventuelle d'un opérateur, stoppant la manipulation ($/manual\_control$).
Pour les coefficients en propulsion, on utilise les données concernant la commande en propulsion envoyée à l'AUV ($/auv\_forward$), le déplacement de l'AUV via le gps ($/sensor\_gps$), ainsi que la reprise en main éventuelle d'un opérateur, stoppant la manipulation ($/manual\_control$). Il faut cependant noter que les données GPS issues de l'AUV sont ici brutes (lattitude et longitude) : un traitement est nécessaire pour obtenir le déplacement de l'AUV au cours du temps.

Dans le cadre du Riptide, les données sont enregistrées dans un fichier .xlog, contenant l'évolution des variables MOOS au cours du temps. On peut donc théoriquement obtenir les commandes, le cap (pouvant être déterminé grâce aux données de l'IMU) et la position de l'AUV (grâce au GPS) au cours des missions effectués. Cependant, en raison de la difficulté à utiliser les Riptides, aucune mission ayant pour but la détermination des paramètres dynamiques n'a pu être effectuée. Les logs issus de l'expérimentation du lac Ty Colo n'ont pas non plus pu être exploitées pour estimer ces paramètres, faute de temps.
Pour autant, les coefficients d'inertie de celui-ci ont pu être déterminés. Grâce à la connaissance de la masse, de la longueur et du rayon du Riptide,il est possible d'obtenir les paramètres p_3 et p_4 liés à l'inertie du Riptide.

3.2.2.3 Comparaison avec une simulation

Plusieurs expérimentations ont pu être réalisés avec le Folaga : une première dans la piscine de l'ENSTA Bretagne (dans laquelle seules des commandes en rotation ont pu être effectuées), une autre au lac de la Penfeld au cours du mois de décembre, et enfin une dernière dans le lac Ty Colo fin février.

Au cours de l'expérimentation du mois de décembre, il a été relevé que la fréquence du GPS du Folaga (0.2 Hz) était trop faible pour obtenir une estimation satisfaisante de sa trajectoire au cours du temps. En effet, la portée de la télécommande du Folaga réduit son contrôle à quelques dizaines de mètres, l'empêchant d'effectuer sous le contrôle d'un opérateur une trajectoire suffisamment grande pour pouvoir obtenir un nombre de points satisfaisant à exploiter. A cause de ce problème, les données brutes issues du GPS du Folaga n'étaient pas utilisables pour obtenir une estimation des coefficients de propulsion et de frottement suivant l'axe x.
Plusieurs solutions sont cependant envisageables face à ce problème. On peut imaginer exploiter les données du GPS à l'aide d'un algorithme de krigeage, ou multiplier les expérimentations pour une commande identique, afin de pouvoir obtenir un nombre suffisant de données pour estimer ces coefficients. De plus, la fréquence de l'échosondeur du Folaga rend possible d'obtenir une estimée de sa position en fonction du temps. Pour cela, l'exploitation des données du lac de Ty Colo, dans laquelle le Folaga a pu effectuer une longue trajectoire à commande constante, paraît prometteuse pour notre estimation.

Face à ce problème, nous avons choisi de nous concentrer sur la détermination des coefficients liés à la rotation du Folaga. Nous avons alors pu être capable d'exploiter les données issus du cap de l'AUV pour l'ensemble des expérimentations réalisées.

Dans un premier temps, ces coefficients ont été obtenus en appliquant un algorithme de minimisation aux équations (3.8), (3.9) et (3.10). L'algorithme choisi ici est un algorithme appliquant la méthode des moindres carrés.
On obtient alors une expérimentation extrêmement fidèle par rapport aux données issues de l'expérimentation de la Penfeld, mais très peu fidèle par rapport aux données issues des expérimentations dans la piscine de l'ENSTA et dans le lac Ty Colo.

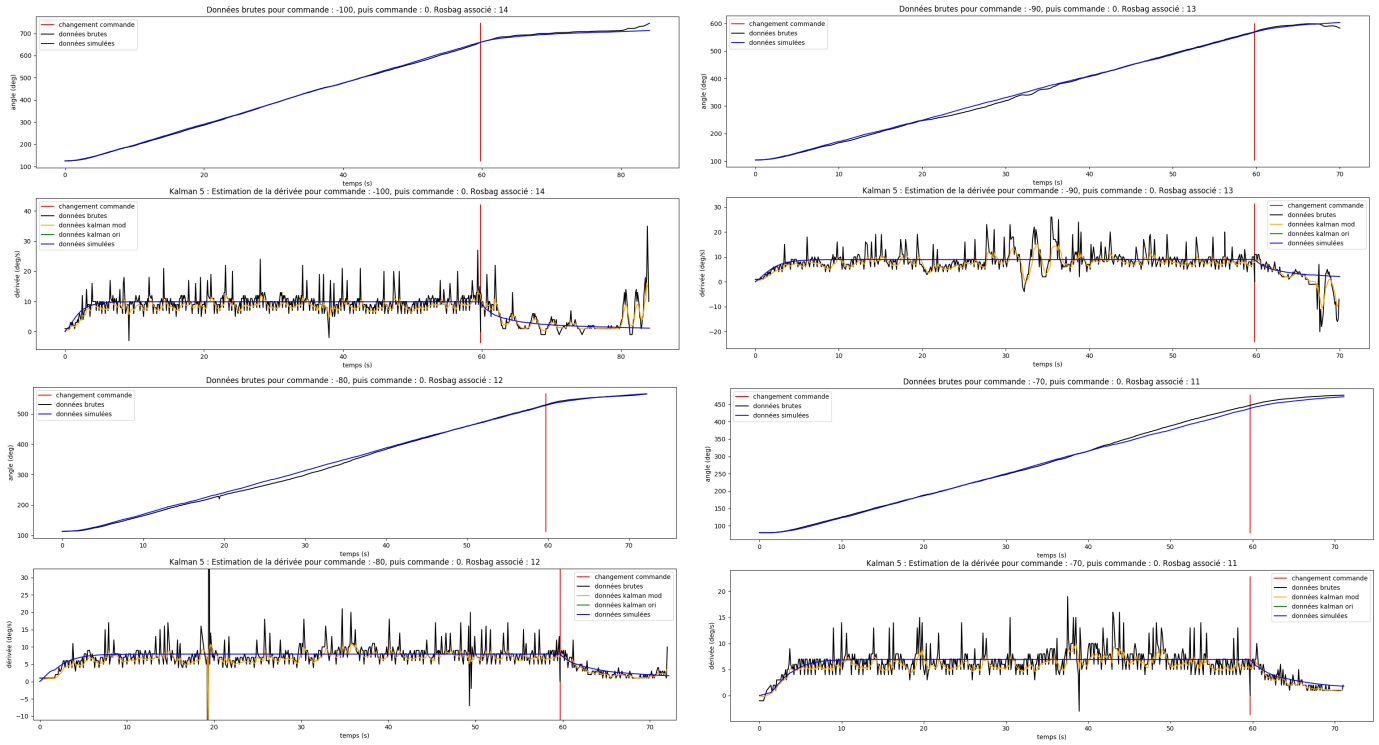

FIGURE 3.11 – Comparaison entre les données issues de la simulation et de l'expérimentation de la Penfeld, pour différentes commandes

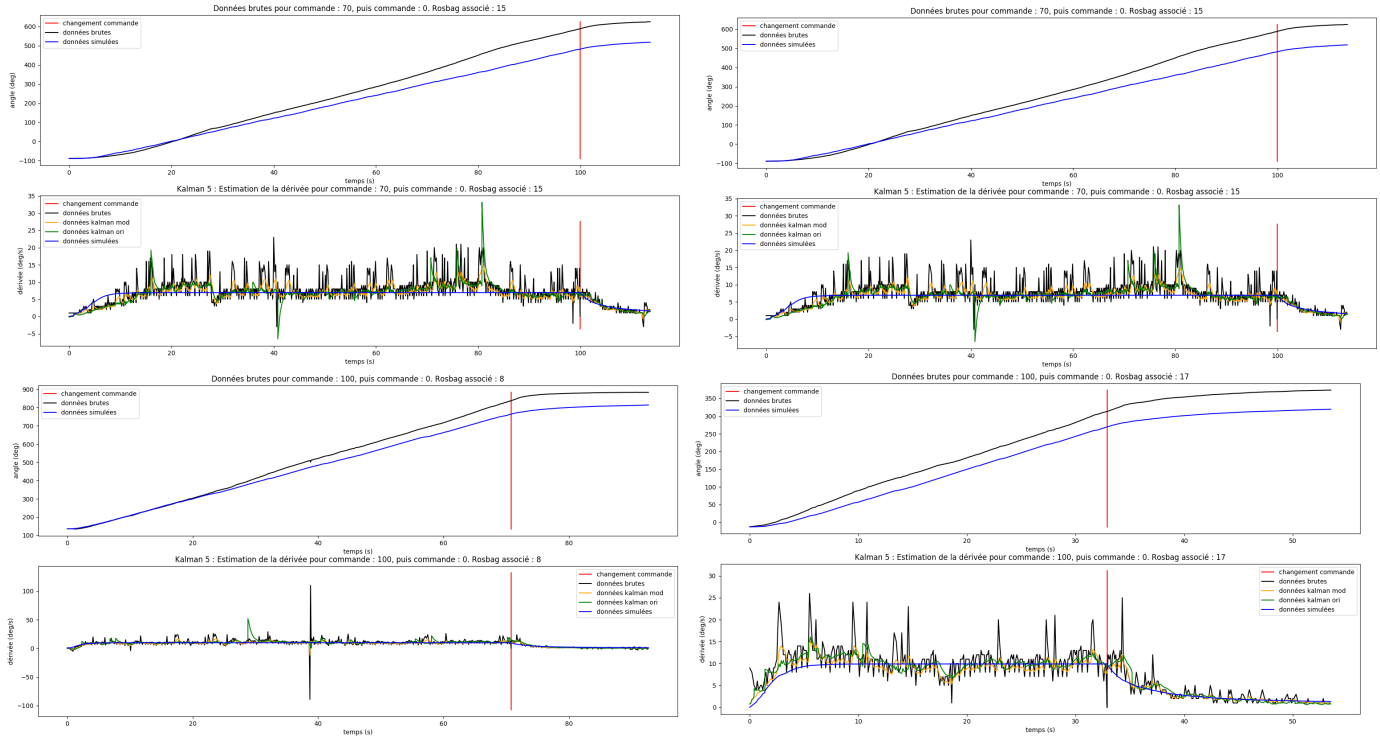

FIGURE 3.12 – Comparaison entre les données issues de la simulation et des expérimentations du lac Ty Colo (gauche) et de la piscine de l'ENSTA Bretagne (droite), pour différentes commandes

Une telle différence peut être due aux différences de condition entre l'expérimentation dans

la Penfeld et les autres expérimentations. Dans le cadre de la Penfeld, l'expérimentation a eu lieu dans un espace ouvert, par vent faible. A l'ENSTA Bretagne, l'expérimentation a eu lieu dans un espace de faible volume, conduisant le Folaga à se cogner régulièrement contre les bords de la piscine (ce qui faussait alors les mesures de cap et de vitesse). Enfin, l'expérimentation dans le lac Ty Colo a eu lieu sous un vent violent ; étant donné que le Folaga se trouvait à la surface de l'eau au cours des expérimentations, cela a donc pu influencer sa vitesse de rotation.

On remarque aussi qu'entre l'expérimentation dans le lac Ty Colo et l'expérimentation dans le lac de la Penfeld, la vitesse maximale du Folaga semble changer pour une commande identique. Une telle différence peut aussi être liée aux changements mécaniques apportés au Folaga suite aux expérimentations dans le lac de Guerlédan (ajout de flotteurs sur le côté notamment, afin d'améliorer sa flotabilité), ayant eu lieu après l'expérimentation dans le lac de la Penfeld.

Avec l'utilisation des coefficients déterminés par la méthode des intervalles, on obtient les courbes suivantes.

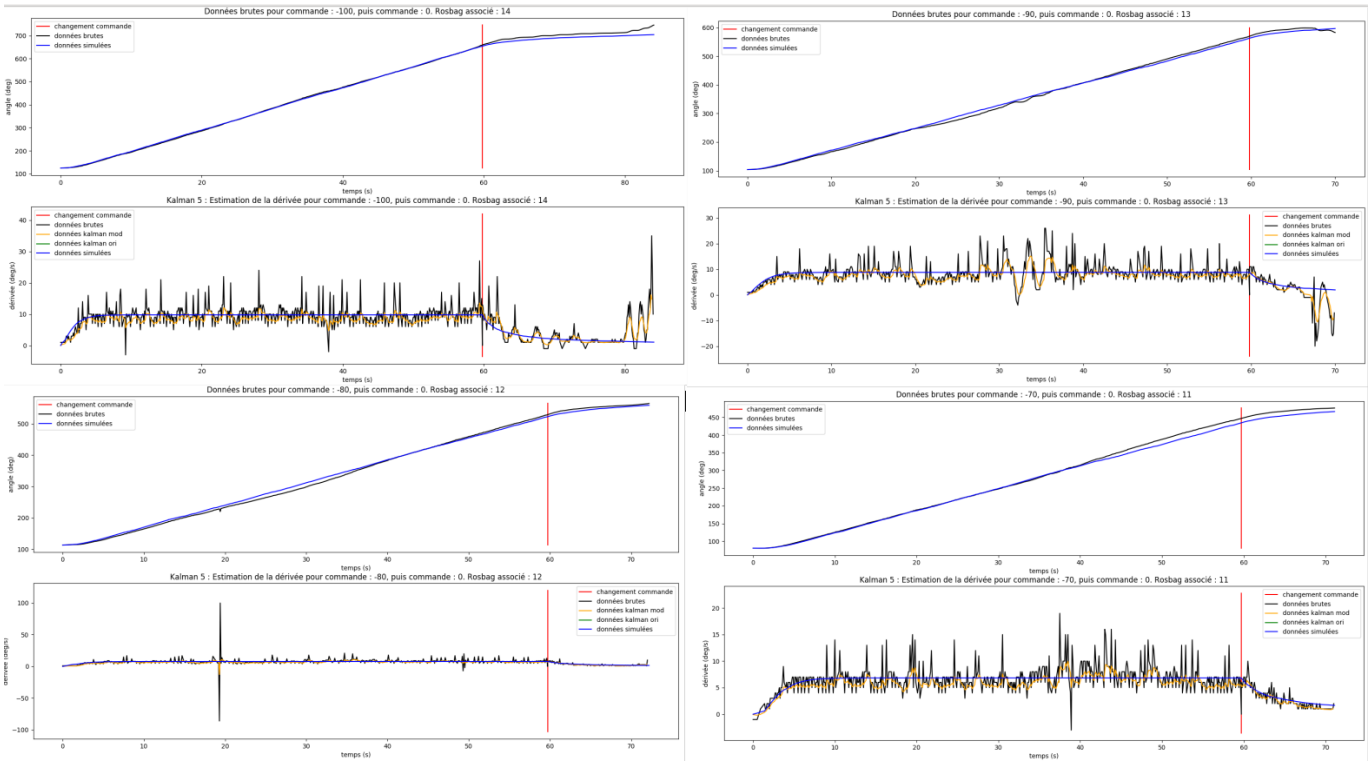

FIGURE 3.13 – Comparaison entre les données issues de la simulation et de l'expérimentation de la Penfeld, pour différentes commandes, avec des coefficients issus de la méthode par intervalle

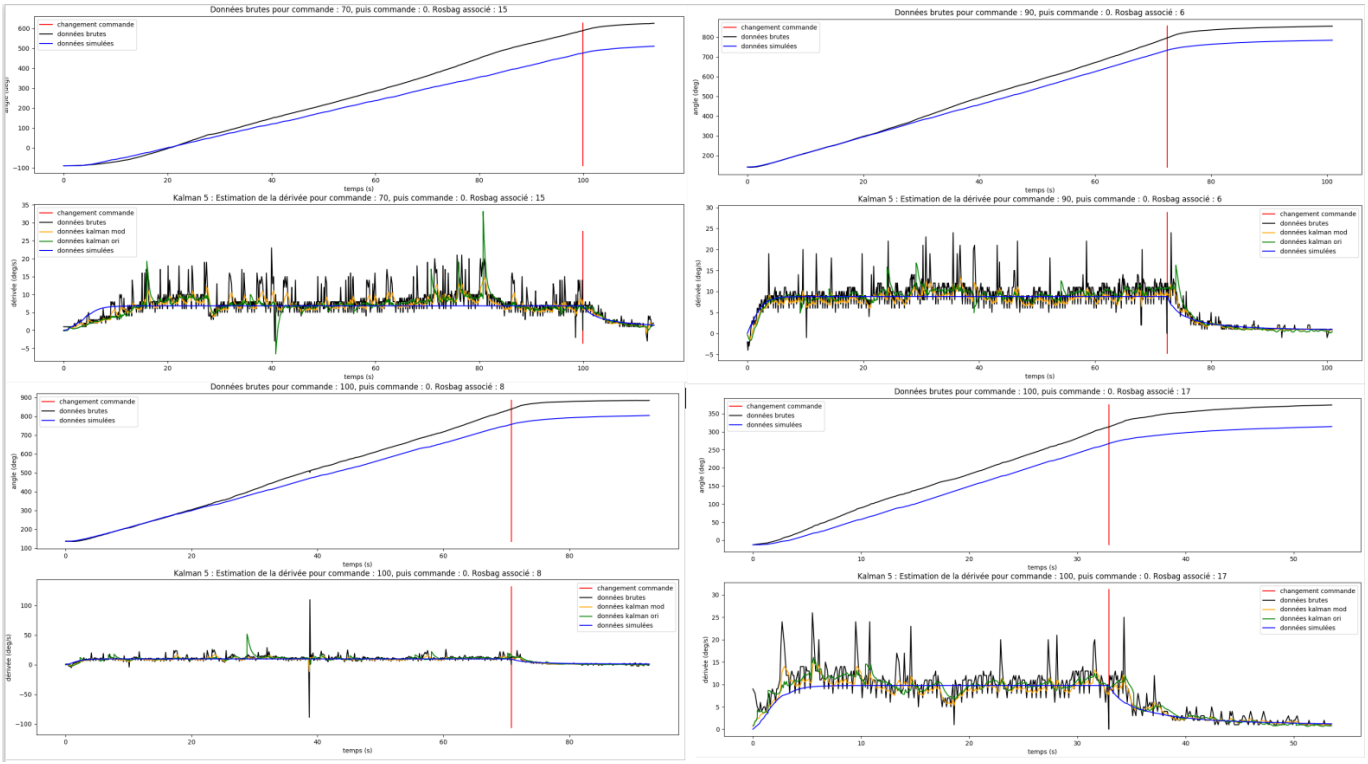

FIGURE 3.14 – Comparaison entre les données issues de la simulation et des expérimentations du lac Ty Colo (gauche) et de la piscine de l'ENSTA Bretagne (droite), pour différentes commandes, avec des coefficients issus de la méthode par intervalle

On constate ici aussi un décalage, entre les différentes simulations, de plus, le calcul du terme p_{frot} par la première méthode des intervalles donne un très grand intervalle. En effet, le coefficient p_{frot} aurait pu être plus contracté mais par manque de temps, nous avons un résultat un peu moins bon que la méthode des moindres carrés. Pour la seconde méthode des intervalles, la différence des ordres de grandeur ne donnait pas d'approximation assez exacte de la solution.

De nouvelles expérimentations au lac de la Penfeld seraient pertinentes pour valider ou infirmer ces hypothèses, et pouvoir obtenir une meilleure estimation des paramètres dynamiques du Folaga.

3.3 Gestion des cycles stables

3.3.1 Définition des cycles stables

3.3.1.1 Définitions

Dans notre projet, un AUV est amené à "rebondir", c'est-à-dire changer de cap, dès qu'il croise une isobathe définie. Il y a alors trois possibilités. Premièrement, l'AUV peut finir par ne plus croiser d'isobathe et donc ne jamais finir les ordres qui lui sont envoyés. Dans ce cas, le robot se perd et le cycle n'est pas considéré comme stable. Deuxièmement, l'AUV peut réussir à toujours recontrer une isobathe et donc toujours rebondir mais jamais aux mêmes endroits

de l'isobathe. Le robot va alors se balader sur la carte sans jamais converger vers un cycle fixe et nous n'aurons alors pas de cycle stable. Troisièmement, l'AUV peut converger vers une zone dans laquelle il rencontrera des isobathes qui lui permettront d'effectuer de manière récurrente les ordres de changement de cap qui lui sont envoyés. On observera alors une convergence de l'AUV vers une zone fixe et on parlera alors de "cycle stable".
Un cycle stable peut être représenté par l'image suivante :

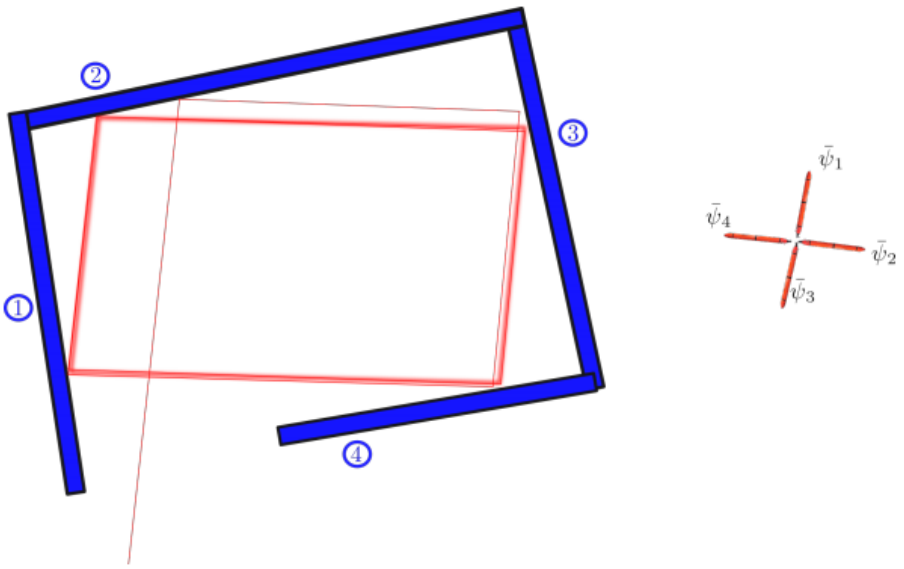

FIGURE 3.15 – Exemple d'un cycle stable

3.3.1.2 Implémentation d'une solution

Introduction

Nous avons choisi d'opter pour une méthode de type "*brute-force*", c'est à dire de chercher toutes les combinaisons possibles de cycles stables sur la carte.
L'algorithme de recherche de cycles stables est implémenté dans le script *treatment.py* dans l'objet **Map**.

Cet objet prend pour entrée une liste de caps à suivre et un tableau binaire du terrain représentant la carte pour une isobathe sélectionnée. Un élément $a_{i,j}$ de ce tableau est de valeur supérieure à 0 si l'isobathe passe par le point de coordonnées (i, j). A noter qu'un MNT peut être chargée directement dans l'objet **Map**, avec l'opération de krigeage et de filtre d'isobathes, par la méthode *read_mnt*.
L'algorithme est en mesure de trouver les cycles stables sur ce tableau binaire, de les visualiser et de nous en fournir les coordonnées pour chaque étape du cycle.

Définitions

Nous définissons dans cette section par le nombre n, le nombre de caps prédéfinis par l'utilisateur en entrée. Il est important de rappeler que les caps $[\Psi_1, \Psi_2, ..., \Psi_n]$ ont été préalablement

fixé au début de l'algorithme par l'utilisateur, ce n'est pas à l'algorithme de les choisir.

Nous appelons par **séquence complète** une liste de n tuples : $[p_1, p_2, ..., p_n]$. Chaque tuple p_i pour $i > 1$ de cette liste représente les coordonnées (dans le tableau) d'un rebond de l'algorithme sur l'isobathe. p_1 modélise la position initiale du robot au début de la séquence.

Une séquence complète est dite être un **pseudo-cycle** si p_1 est voisin de p_n. p_1 et p_n sont dit ϵ-voisins si et seulement si :

$$\|p_n - p_1\| \leqslant \epsilon \tag{3.12}$$

où ϵ est un seuil prédéfini par l'utilisateur.

On définit un **m-pseudo-cycle** par un pseudo-cycle $[p_1, p_2, ..., p_n]$ qui peut être **itéré** m fois et qui à chaque itération $i \in [|1, m|]$ nous fournit un pseudo-cycle.
Pour chaque itération $i \in [|1, m|]$, on obtient un pseudo-cycle dont le point de d'arrivée $p_{n,i}$ est le point de départ $p_{1,i+1}$ du pseudo-cycle de la $i + 1^{\text{ème}}$ itération.

Un **cycle** peut être défini comme un ∞-pseudo-cycle avec la contrainte suivante : $\forall m \in \mathbb{N}^*$: $p_{m,n}$ est ϵ-voisin de $p_{1,1}$

Un **cycle stable** est un cycle $[p_0, p_1, ..., p_n]$ vérifiant deux critères supplémentaires :

- Pour chaque rebond k du robot, il faut que : $\forall m \in \mathbb{N}^*$: $p_{m,k}$ soit ϵ-voisin de $p_{1,k}$
- Chaque séquence complète voisine de $p_{1,1}$ et réalisée avec les mêmes caps $[\Psi_1, \Psi_2, ..., \Psi_n]$ que le cycle stable forment des m-pseudo-cycles et convergent vers le cycle stable. Par convergence, on entend que à chaque itérations, les rebonds de ces séquences complètes se rapprochent des rebonds du cycle stable. Par application du théorème du point fixe de Banach, dont vous trouverez des démonstrations et applications ici [1], on sait alors que ce cycle va converger vers des points fixes sur chaque rebond.

Étapes de l'algorithme implémenté

Voici sous forme schématisée l'algorithme que nous avons implémenté :

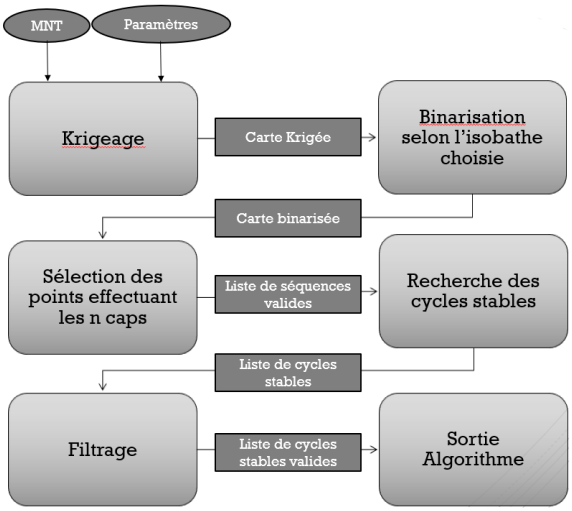

FIGURE 3.16 – Fonctionnement de l'algorithme de recherche de cycles stables

Nous allons revenir sur chacune des étapes identifiées sur le schéma ci dessus.

Extraction des données de mesures

Cette étape est la première couche de traitement du MNT afin d'obtenir un format standard de points de mesures pour la suite du traitement. Les formats de MNT pouvant parfois varier, nous faisons toutefois l'hypothèse raisonnable qu'il s'agit d'un fichier non crypté, où chaque point de mesure correspond à une ligne et chaque coordonnée du point est séparée par un unique séparateur. Ce séparateur doit être distinct des caractères utilisés pour l'écriture des coordonnées.
Le format choisi pour stocker les données est le dictionnaire python pour sa rapidité de remplissage. Les points de mesure y sont stockés de façon à avoir les coordonnées $[x, y]$ en tant que clés, et la mesure z en tant que valeur associée. Les points peuvent être exprimés dans un système de coordonnées au choix, pourvu que la zone dans laquelle s'effectuera la mission n'impliquera pas de trop grandes erreurs de mesures dans le système choisi. Une conversion devra être faite si nécessaire, pour ramener l'écriture des points dans un système plan.

Ce traitement possède donc les paramètres suivant :
— le nom du fichier MNT,
— le séparateur entre les coordonnées des points de mesure,
— le système de coordonnées utilisé.

Interpolation

L'objectif de cette étape est de s'adapter à la contrainte d'affichage des données de terrain dans un tableau, d'une part pour l'affichage en tant qu'image et d'autre part pour la détection

numérique d'une isobathe. En entrée, nous avons un dictionnaire de points généralement désordonnés, et en sortie, il nous faut des points ordonnés selon un quadrillage orthogonal caractérisé par un pas suivant x et un pas suivant y.
Heureusement, une méthode appelée **Krigeage** nous permet de faire une interpolation des points du quadrillage, à partir des mesures, tout en minimisant la variance de l'erreur commise. En effectuant un krigeage de type ordinaire, il faudra choisir un modèle de dépendance spatiale que l'on appelle semi-variogramme. Quelques modèles prédéfinis existent déjà tels que le "linéaire", "gaussien", "sphérique", "exponentiel" ou "puissance". Il est important d'en choisir un donnant une représentation vraisemblable des données géographiques. Il sera d'ailleurs possiblement nécessaire de filtrer un certain nombre de points de mesures afin de limiter le temps de calcul de l'interpolation.
Voici donc les paramètres de cette étape :

- le type de krigeage,
- le dimensionnement du quadrillage par ses pas dx et dy,
- le filtrage des données de mesure pour réduire le temps de calcul.

Binarisation selon l'isobathe choisie

Cette étape transforme tous les points de la carte ayant la profondeur choisie en 1 et tous les autres points en 0. C'est une étape de filtrage nécessaire pour pouvoir travailler simplement sur une isobathe.
Les deux paramètres nécessaires ici seront les suivants :

- l'isobathe,
- la marge de précision sur l'isobathe.

Notons qu'il est important de régler une isobathe intéressante suivant la zone de mission afin de permettre la détection de cycles stables. De plus, la marge de précision a une influence sur l'épaisseur en pixels de l'isobathe sur l'image binarisée. Certains contours peuvent disparaître si cette marge est trop faible tandis que d'autres peuvent s'étendre sur de grandes zones dans le cas inverse.

Sélection des points de départ de séquences complètes

La génération de séquences se fait avec la méthode *do_sequence*. Cette méthode va chercher pour un point et une liste de caps donnés la séquence générée.
Le calcul de la position des rebonds d'un robot pour une position et une direction données se fait avec la méthode *find_next_step*. Le rebonds est trouvé en générant une droite, avec la fonction de *skimage* qui est plus rapide que celle sur opencv, sur un tableau de même dimension que la carte binaire. La multiplication de notre carte avec le tableau de cette droite nous fournit la position des intersections donc la position du rebond du robot sur l'isobathe.

Maitenant que nous somme en mesure de générer des séquences sur notre carte binaire, nous sélectionnons uniquement toutes les séquences complètes que nous pouvons trouver via la méthode *find_complete_sequences*. Nous stockons les coordonnées de ces séquences complètes

dans un dictionnaire qui a pour clés les coordonnées du point de départ de nos séquences.

Recherche des cycles stables

Dans cette étape, nous recherchons les cycles stables. Un tel cycle est identifié si deux séquences complètes présentent dans un même voisinage convergent vers une même séquence complète.
On peut se représenter cela grâce à ce schéma :

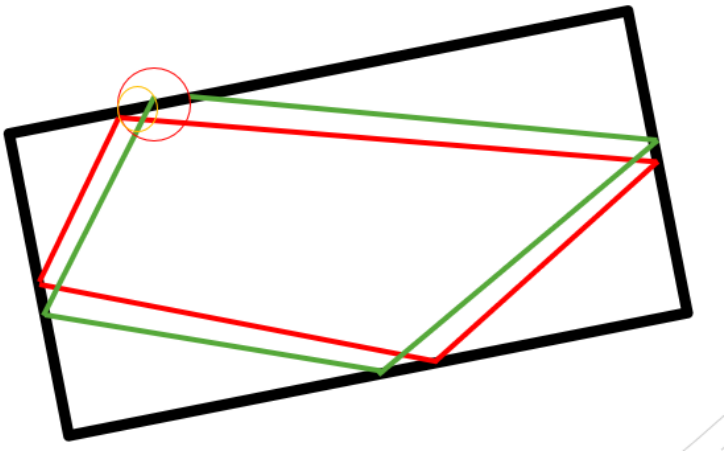

FIGURE 3.17 – Schéma convergence de cycle

On observe une mécanique d'encapsulation qui représente la convergence définie précédemment. La recherche des cycles stables se fait par la méthode *find_ stable_ cycles* qui va enregistrer les cycles stables dans une liste *stable_ cycles* et les séquences convergeant vers ces cycles dans la liste *seq_ conv*.

Filtrage

Dans cette étape nous réalisons trois filtrages. Ces opérations dépendent des paramètres choisis.

1. Les cycles ayant des côtés trop petits sont éliminés
2. Les cycles trop proches sont éliminés
3. On affiche seulement les n plus gros ou ceux choisis manuellement

Sortie de l'algorithme

L'algorithme ressort les cycles stables ainsi que les points fixes des différentes zones. Il ressort aussi une partie graphique qui nous permet de mieux visualiser les cycles stables sur la carte. Voici sous forme de schéma les entrés/sorties de l'algorithme ainsi développé :

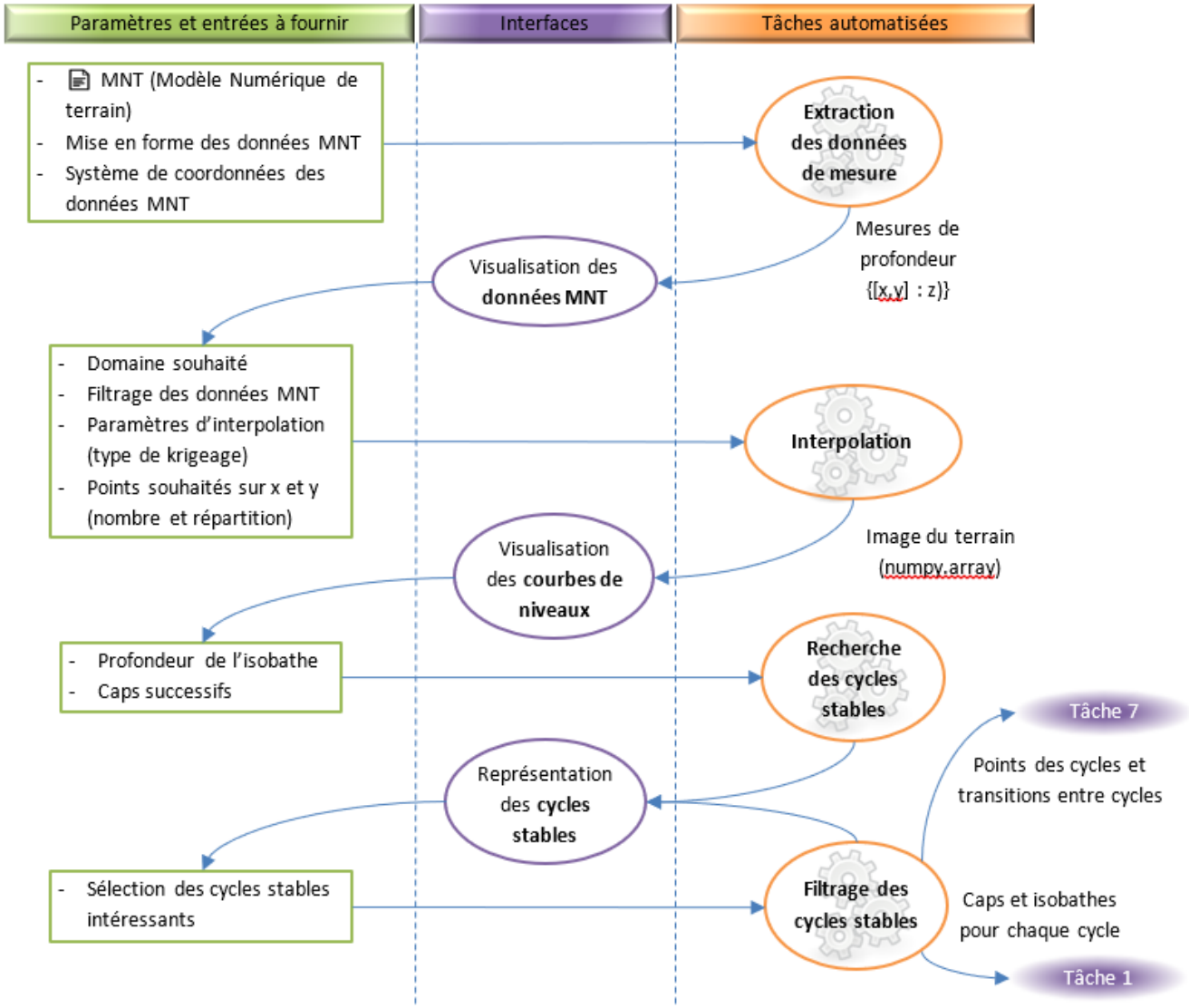

FIGURE 3.18 – Schéma des paramètres et des sorties de l'algorithme de détection de cycles stables

On voit par cette représentation les diverses interactions nécessaires entre le programme et l'ingénieur, ces interactions se font via les interfaces (en violet) générées par l'algorithme afin de déterminer certains paramètres.

Chaque interface sert de la manière suivante :

1. Visualisation des données MNT : représenter la totalité de la zone où des relevés sont disponibles pour préciser le domaine d'exploration sous-marine souhaité, doser la réduction éventuelle du nombre de points utilisés pour effectuer l'interpolation et ainsi gagner du temps de calcul.

2. Visualisation des courbes de niveaux : observer les lignes de profondeur sur le domaine d'exploration et en choisir une sur laquelle baser la recherche de cycles stables. Tant que le programme n'aura pas de notion de population de cycles stables en fonction de la forme de l'isobathe et des caps du l'AUV, ce choix restera à l'initiative de l'ingénieur.
3. Représentation des cycles stables : observer la répartition des cycles stables trouvés sur le terrain à explorer et choisir lesquels garder ou filtrer afin d'avoir une répartition de cycles couvrant toute la zone d'intérêt.

3.3.1.3 Conclusion

L'approche que nous avons choisi ici est une approche d'un point de vue "existence et nombre de cycles stables". Elle implique le choix préalable des différents caps que prendront l'AUV lorsqu'il croise une isobathe donnée. Afin de se rapprocher de la réalité du besoin, une amélioration future de ce programme serait d'adopter une approche "recherche de cycles stables recouvrant une zone donnée".
Cette approche est compatible avec la précédente car elle devra faire appel à la fonction de recherche de cycles stables, mais en faisant varier les caps choisis et les isobathes de travail. L'objectif final étant de pouvoir proposer une machine à états permettant d'explorer l'ensemble de la zone sous-marine, avec la meilleure répartition de cycles possibles, en garantissant un certain niveau de stabilité.

3.3.2 Intelligence

3.3.2.1 Position de la tâche dans le projet

La tâche 7 a été réalisée en étroite collaboration avec les membres de la tâche 5 ainsi que ceux de la tâche 1.

En effet les sorties de la tâche 5 sont nécessaires à la tâche 7 car ce sont les entrées de l'algorithme.

Celles-ci sont :

— Une liste de cycles, avec dans chaque cycle la liste des zones avec leur position géographique.
— Une liste de transitions autorisées entre les différents cycles.

De plus les sorties de l'algorithme d'intelligence sont :

— Un fichier prolog consultable
— Les points extrêmes pour passer d'un cycle à un autre ainsi que le cap qu'il faut prendre pour naviguer du point de départ au point d'arriver

Cette sortie va être utilisée par la tâche 1 qui a besoin de cette information pour réaliser l'automate, ayant besoin de savoir quel est le trajet optimal pour se rendre d'un cycle à un autre.

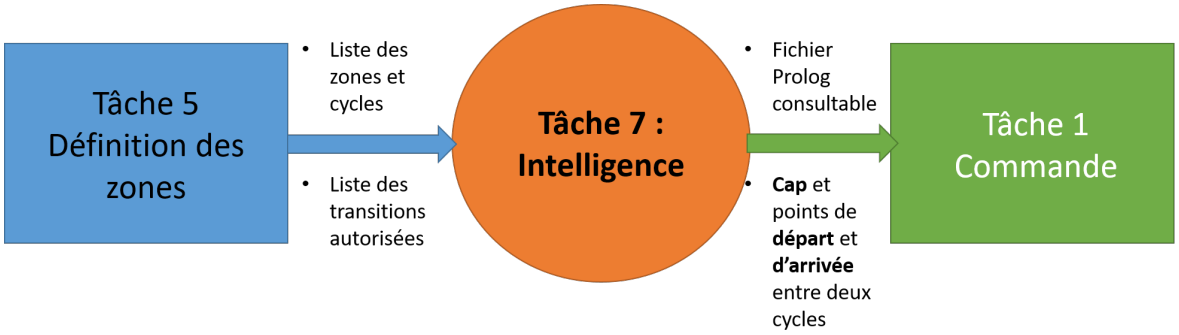

FIGURE 3.19 – Position de la tâche 7 dans le projet Go-zone

3.3.2.2 Travail réalisé

Le travail a été réalisé en trois étapes, ce qui a permis de commencer simplement et continuer sur un cas plus général et plus complexe à coder.

Etape 1 : Création d'un fichier Prolog sur un exemple particulier

Dans un premier temps nous avons commencé par réalisé un programme Prolog qui réalisait les tâches demandées dans le cas où nous connaissions déjà les caractéristiques de la carte.

Nous avons donc coder en dur la liste des cycles ainsi que la liste des transitions autorisées entre chaque cycle. Cette étape nous a permis de tester et améliorer notre code Prolog.

Etape 2 : Génération automatique du fichier Prolog

Dans un second temps nous avons réalisé un programme python qui génère automatiquement le fichier Prolog. Ce programme prend en entrée la liste des cycles ainsi que la liste des transitions autorisées et génère automatiquement le Prolog associé à ces informations. La génération automatique est réalisée de la manière suivante :

1. Déclaration des zones et de leur appartenance à un cycle
2. Déclaration de la liaison possible entre chaque zone consécutive dans un cycle.
3. Ajout des liaisons possibles grâce aux transitions autorisées entre les cycles
4. Ecriture des fonctions de calcul du plus court chemin en Prolog.

Ici, on voit que l'étape 4 sera toujours similaire quelle que soit la zone mais les étapes 1,2,3 seront différentes si la carte est différente. C'est pour cette raison qu'il a fallu générer automatiquement du code Prolog.

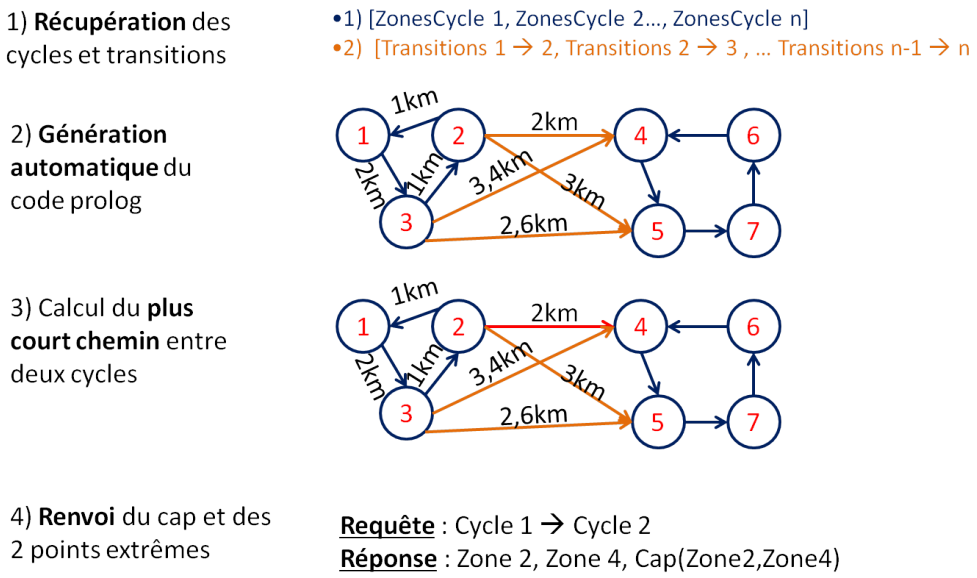

FIGURE 3.20 – Schéma explicatif tâche 7

L'étape 1 correspond donc à la création de **prédicats** (déclarer les zones) et de **règles unaires** (appartenance à un cycle donné). Le même type de règle unaire est alors instancié dans les étapes 2 et 3.

Dans l'étape 4, on définit les règles suivantes :

— Pour vérifier si un chemin existe (est autorisé) dans le graphe généré automatiquement, la clause **travel(A,B,_,_,L)** sera vraie si et seulement si le chemin de A à B est possible, avec une longueur de parcours L

— **path(A,B,Path,Len)**, qui est vraie lorsque Path est une liste de points reliant A à B et Len désigne la longueur de Path

— **shortest(A,B,Path,Length)**, qui est vraie lorsque Path désigne le chemin de longueur minimale Length.

Etape 3 : Détection des changements de cycle

Finalement, lorsqu'il sera interrogé, le code Prolog va chercher tous les chemins possibles entre deux points et retourner le plus court d'entre eux. Cette sortie sera examinée sous Python : si il y a un pont reliant deux cycles différents sur le parcours proposé, alors l'algorithme le signalera et renverra le cap et les zones impliquées dans ce changement de cycle.

3.3.2.3 Evolutions possibles

Pour la suite de cette partie, il y a plusieurs pistes à explorer :

— Ecrire un code Prolog consultable capable de répondre à des requêtes inattendues : remplir une mission particulière, éviter de passer par une certaine zone, etc. Cette piste impliquerait l'utilisation de **Réseaux de Petri** qui sont particulièrement efficaces dans ce type de situation. On aura eu l'occasion de les découvrir dans le cours disponible sur

la page web du projet.

— Rendre ce même réseau de Petri **temporisé et interprété**, afin d'avoir une véritable machine à états qui fournirait directement la commande et les transitions d'états à l'AUV. Il faudrait peut-être, pour cela, passer à un langage de programmation capable d'envoyer une commande.

— Optimiser l'algorithme de recherche de chemin pour pouvoir l'appliquer à des situations mettant en jeu un très grand nombre de zones tout en conservant une complexité raisonnable. Une idée serait d'utiliser les **colonies de fourmis** qui convergent naturellement vers une solution, par exemple.

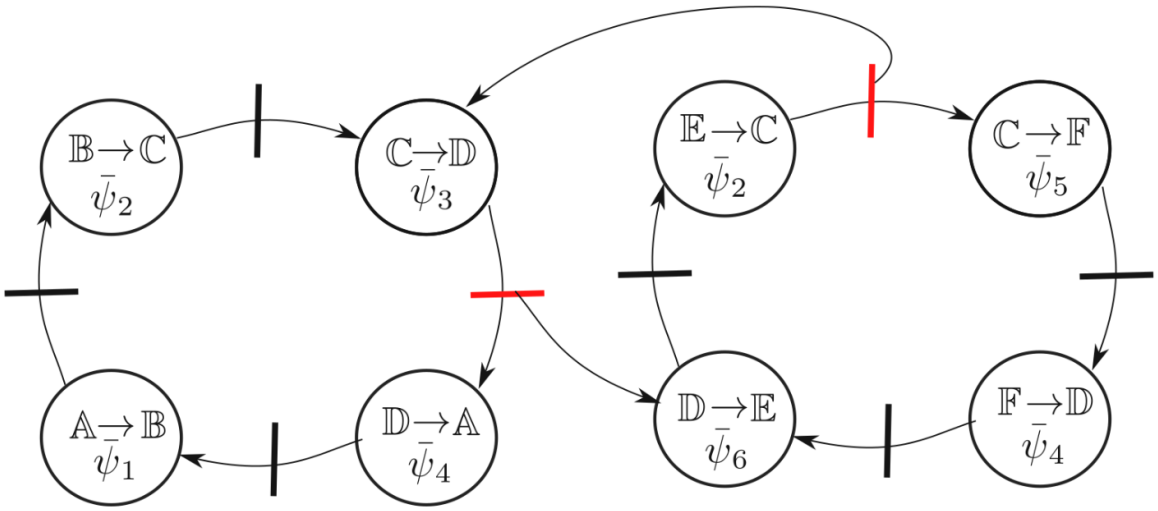

FIGURE 3.21 – Réseau de Petri représentant une mission possible de l'AUV

3.3.3 Analyse de stabilité par Tubex

3.3.3.1 TUBEX

TUBEX est une librairie développée en C++ par Simon Rohou [4] qui hérite pour une grande partie de la librairie IBEX qui propose des algorithmes permettant d'effectuer des opérations sur des ensembles. Les ensembles utilisés dans la librairie IBEX sont des intervalles de une ou plusieurs dimensions. La librairie TUBEX vise à exploiter la librairie IBEX pour étudier la dynamique de systèmes. Il est donc possible en précisant un ensemble de départ et une équation d'évolution ou une trajectoire de calculer l'évolution temporelle de notre ensemble.

Avec un vecteur d'état

$$X = \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.13}$$

On peut prendre pour fonction d'évolution :

$$\dot{X} = f(t) \tag{3.14}$$

Avec :

$$f(t) = \begin{bmatrix} 1 \\ sin(t/10) + 1/5 \end{bmatrix} \tag{3.15}$$

L'évolution du tube donnera :

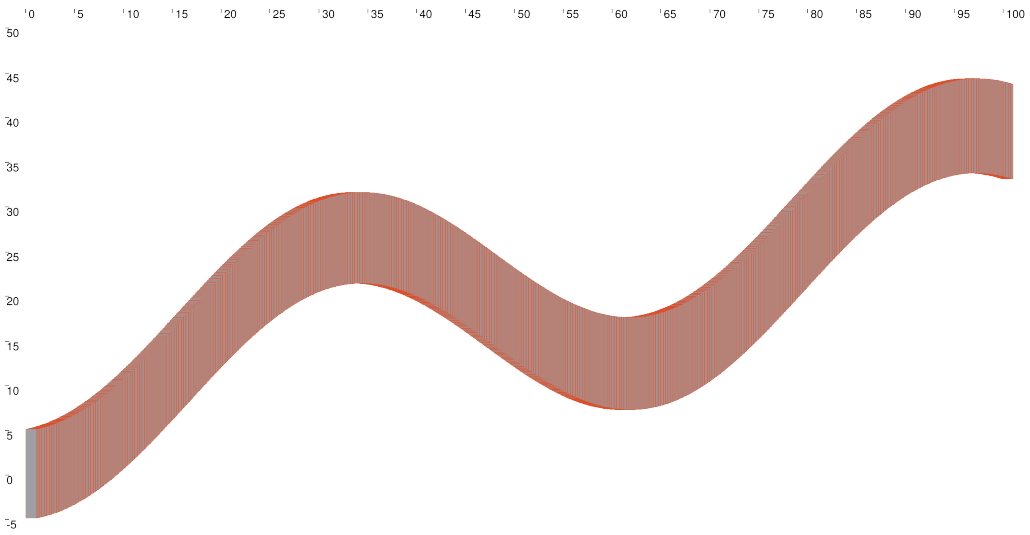

FIGURE 3.22 – Tube2D

Sur la figure 3.22, on peut voir l'évolution d'une boxe (intervalle de deux dimensions) au cours du temps.

3.3.3.2 TUBEX pour Go Zone

Dans le cadre du projet Go Zone, nous utilisons TUBEX pour valider la stabilité de cycle de déplacement de nos AUV. Pour l'implémenter, nous avons d'abord légèrement simplifié le problème.

1. Le vecteur d'état de l'AUV est :

$$X = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{3.16}$$

En utilisant un modèle de Dubins, on choisit une équation d'évolution de type :

$$\dot{X} = \begin{bmatrix} v.cos(\theta) \\ v.sin(\theta) \\ u \end{bmatrix} \tag{3.17}$$

v étant la vitesse de l'AUV et u la commande en cap. L'approximation du modèle est compensée par les incertitudes que l'on peut définir pour v et θ.

2. L'ensemble de départ est un portion d'isobathe, on ne part pas d'une zone aléatoire.

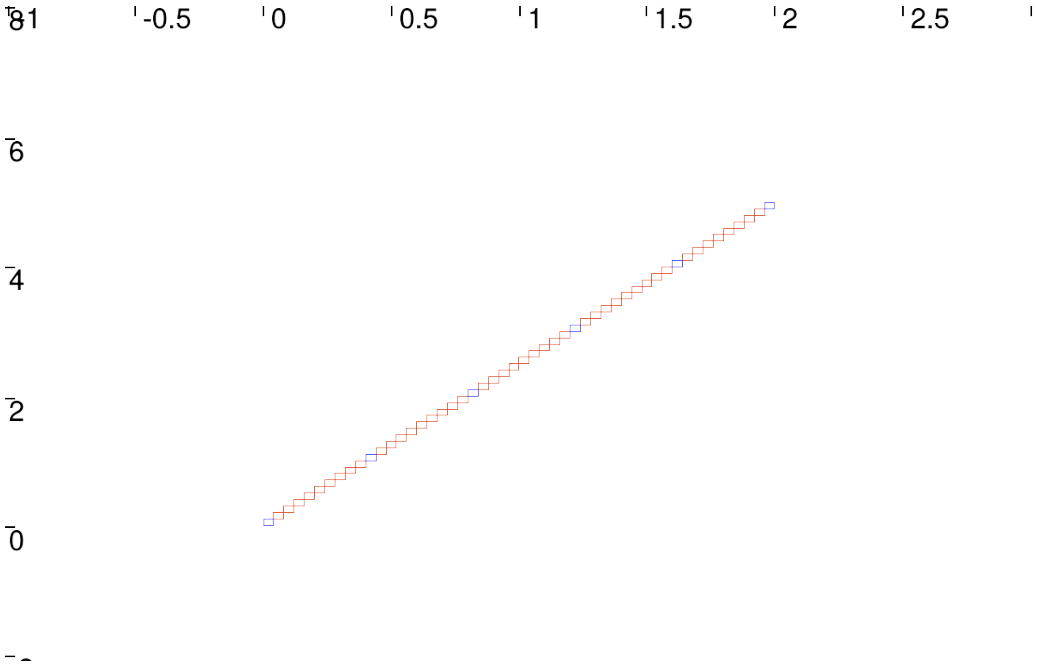

FIGURE 3.23 – Isobathe

On découpe l'isobathe en petite boîte 2D qui contiennent des portions de l'isobathe. La taille de ces boîtes peut être modifiée pour modéliser une plus grande incertitude sur l'ensemble de départ.

Avec ces approximations, il suffit de définir des incertitudes assez grandes dans l'équation d'évolution pour encapsuler l'ensemble des positions de l'AUV au cours du temps.

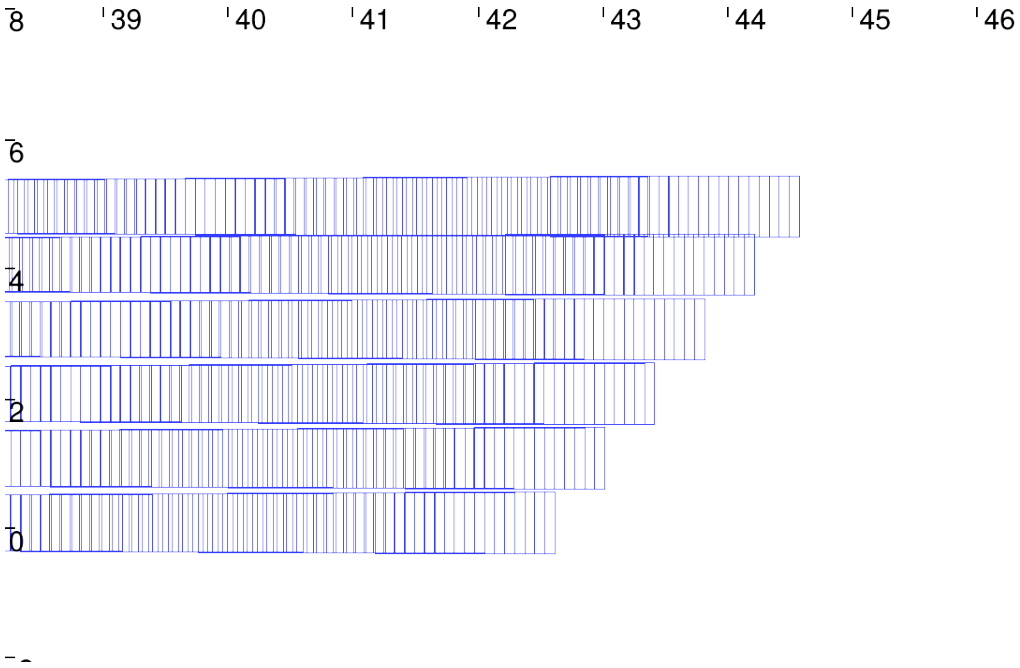

FIGURE 3.24 – Evolution de la position de l'AUV

Dans la figure 3.24, nos boîtes caractérisant l'ensemble des positions de l'AUV augmentent

au cours du temps à cause de l'incertitude en vitesse et en cap.

Stratégie cycle en L Le cycle d'étude choisie est un cycle de déplacement en L. Ce cycle est décrit par quatre déplacements, partant d'un isobathe A, l'AUV se déplace plein EST pendant un temps donné t1, puis va plein NORD jusqu'à rencontrer l'isobathe B pour faire demi-tour. Le retour ce fait donc plein SUD pendant un temps t2 puis plein OUEST jusqu'à retrouver l'isobathe initiale.

Le schéma présentant ce cycle est ci-contre :

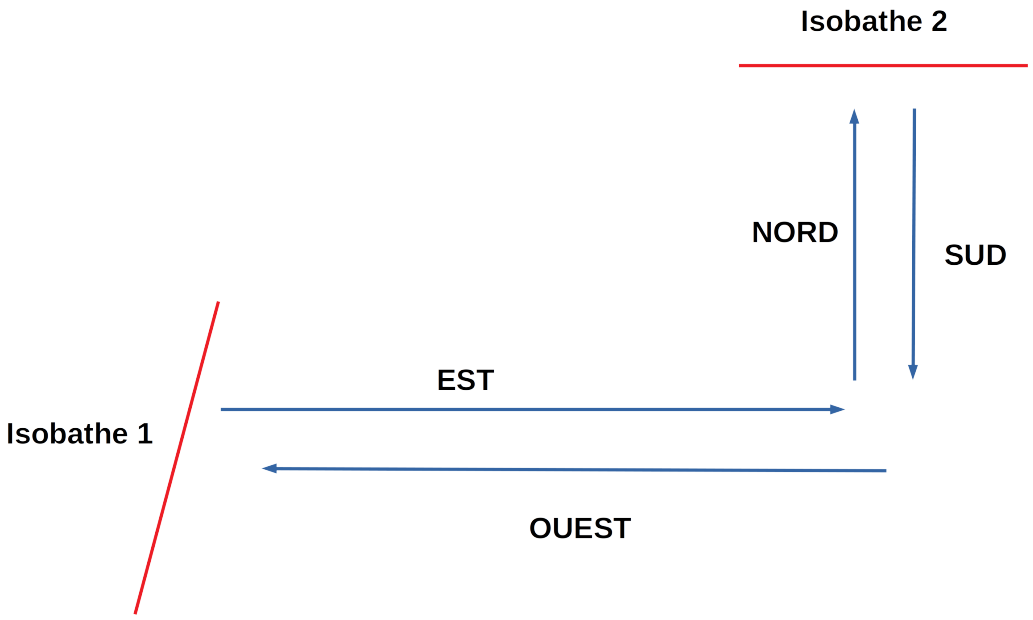

FIGURE 3.25 – Cycle L

Voilà donc les résultats de l'évolution de nos ensembles de départ :

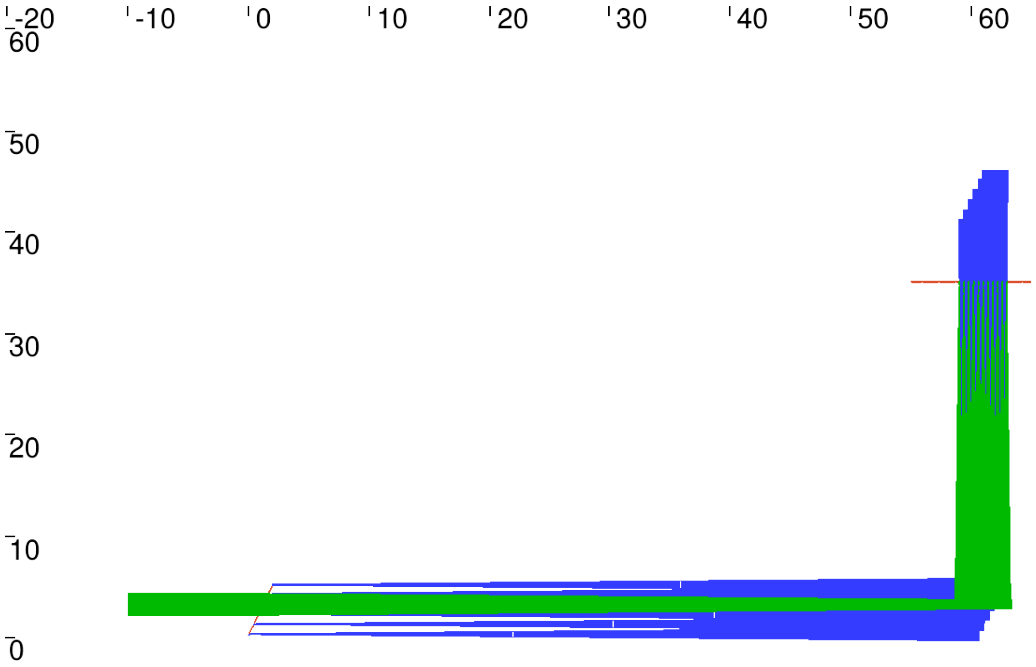

FIGURE 3.26 – Aller/retour en cycle L

Sur la figure 3.26, il n'est affiché qu'un certain nombre d'évolution de nos ensembles de départ pour faciliter la visibilité. En bleu, l'évolution aller du cycle en L, et en vert le retour.

Durant ces déplacements la position de l'AUV, modélisée par nos boîtes, va gagner en incertitudes. Lorsque nos boîtes arrivent sur les isobathes elles sont projetées sur cette dernière et réduisent leur taille. L'incertitude est donc limitée. On peut observer que la portion de segment de l'isobathe 1 qui est coupée par le tube retour est plus petit que l'isobathe 1 elle-même qui était notre ensemble de départ. Notre ensemble d'arrivée étant contenu dans notre ensemble de départ, on obtient donc un cycle stable mais dépendant des incertitudes en cap et en vitesse que l'on a défini.

3.4 Simulation

3.4.1 Simulation de la commande

3.4.1.1 Machine à états finis

Une machine à états finis a ensuite été implémentée dans le contrôleur des AUVs. Avant le déroulement de la mission, on leur rentre pour chacun des différents cycles les isobathes de transition et les caps $[\Psi_1, \Psi_2, ..., \Psi_n]$ entre ces isobathes. Cette machine à états prend en paramètre également les transitions et les caps entre les différents cycles ainsi que le nombre de tours que l'on veut réaliser pour chaque cycle.

Cette partie du code pendant les missions prend en entrée la profondeur mesurée par le sondeur et renvoie en sortie la consigne de cap à suivre en fonction de l'état dans lequel se trouve l'automate.

L'automate est défini par une succession de transitions. Une transition est définit par un état de départ, un état d'arrivée, une fonction booléenne de transition et une fonction appelée.

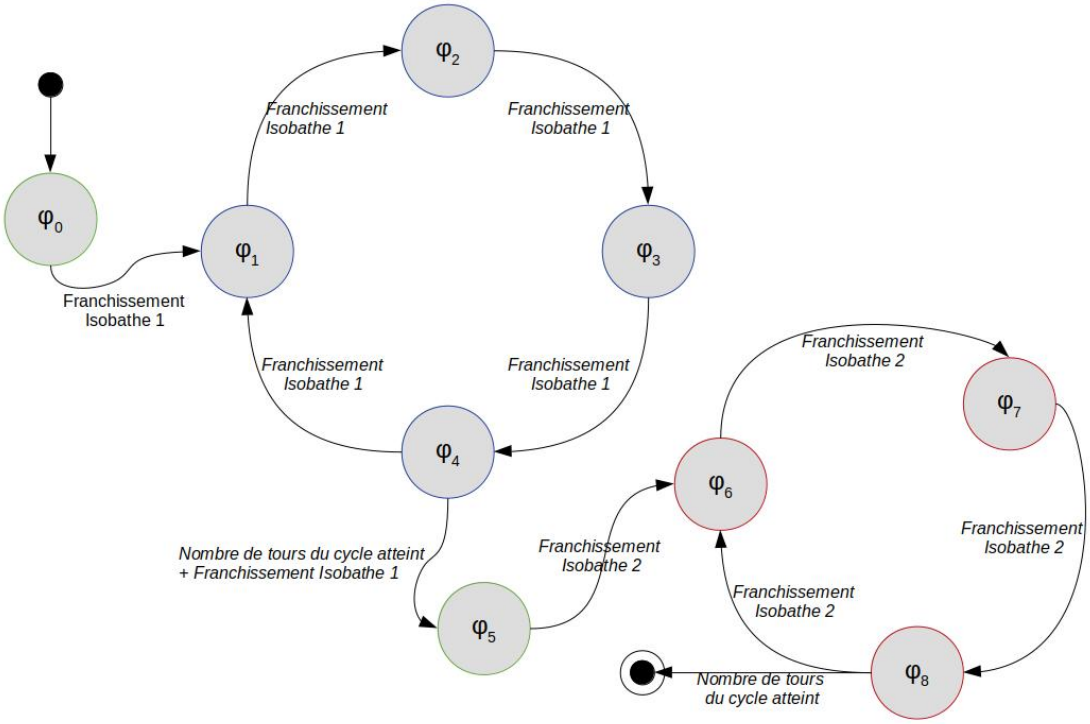

FIGURE 3.27 – Diagramme d'états-transitions de deux cycles

Un état entre deux isobathes correspond donc à une consigne de cap. Pour s'adapter à l'inertie des AUVs, il est nécessaire de rajouter des états *secondaires*, en effet à chaque transition l'AUV franchit un isobathe, sort de la zone délimitée par les isobathes avant d'y rentrer sur le cap du nouvel état. Il faut prendre en compte un double franchissement de chaque isobathe.

Un travail a ensuite été fait sur l'automatisation de la génération de cet automate en le rendant le plus robuste et flexible possible.

3.4.1.2 Contrôle de l'AUV

Afin de tester les automates développés nous avons donc implémenté un contrôleur pour les AUV.

On note R_0 le référentiel fixe supposé cartésien qui servira de base pour la régulation en vitesse et attitude. On note R_v le système de coordonnées lié à l'AUV. Ces référentiels sont représentés dans la figure 3.28 et seront utilisés pour le contrôle des AUVs Riptide et Folaga.

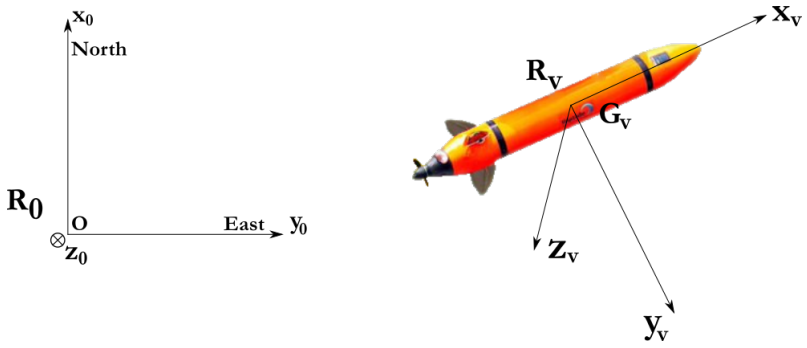

FIGURE 3.28 – Repères fixe et mobile utilisés pour décrire l'état des AUVs

Riptide À partir du vecteur d'état de l'AUV et des actionneurs disponibles, nous développerons d'abord un contrôleur en vitesse puis un contrôleur pour les angles d'Euler qui caractérisent l'attitude.

$$X = \begin{pmatrix} x \\ y \\ z \\ \varphi \\ \theta \\ \psi \\ v \end{pmatrix} \quad ; \quad U = \begin{pmatrix} thrust \\ topFin \\ latFin1 \\ latFin2 \end{pmatrix}$$

Note

Les sous-marins Riptide embarquent nativement des contrôleurs en vitesse et attitude. Du fait de la complexité du changement de *middleware*, l'écriture de commandes directement aux actionneurs est à ce jour impossible. Cela rend la partie qui suit inutilisable jusqu'à nouvel ordre.

Contrôleur en vitesse Considérons un contrôleur qui permet d'obtenir

$$\dot{v} = k_0.(v_{target} - \hat{v})$$

Cela est possible en prenant $k_0 = p_2$, avec p_2 tel que défini précédemment dans le modèle dynamique, puis $u_0 = \sqrt{\frac{p_2}{p_1}}.v_{target}$.
On peut remarquer ici que, à cause du fait que nous n'avons pas accès à la mesure de vitesse, nous travaillons en boucle ouverte. Cela peut engendrer des différences entre la vitesse cible et la vitesse réelle, selon la précision de l'estimation des paramètres physique de l'AUV.

Remarque

A partir de l'erreur d'estimation sur les paramètres physiques p_1 et p_2 de l'AUV, le contrôleur va en réalité générer une commande $u_0 = \sqrt{\frac{\hat{p_2}}{\hat{p_1}}}.v_{target} = \sqrt{\frac{(1+\beta).p_2}{(1+\alpha).p_1}}.v_{target}$.
Cela conduit à une accélération du système $\dot{v} = p_2.(\frac{1+\beta}{1+\alpha}.v_{target} - v)$.

Il s'agit d'une équation différentielle non linéaire dont on peut trouver la solution en utilisant par exemple le solveur en ligne *Wolfram Alpha*, lequel nous donne l'expression temporelle de v :

$$\begin{cases} v(t) = \frac{\sqrt{1+\beta}.v_{target}.\tanh(K+\frac{\sqrt{1+\beta}}{\sqrt{1+\alpha}}.v_{target}.p_2.t)}{\sqrt{1+\alpha}} \\ K = \tanh^{-1}(\frac{\sqrt{1+\alpha}.v_{origin}}{\sqrt{1+\beta}.v_{target}}) \end{cases}$$

Nous voyons en particulier que : $v(t) \xrightarrow[t\to\infty]{} \sqrt{\frac{1+\beta}{1+\alpha}}.v_{target}$.

En négligeant les périodes de transition (on ne devrait pas avoir trop de changements pour v_{target}, donc pas trop d'ennuis même si le temps de convergence n'est pas particulièrement faible), nous prendrons $\hat{v} = v_{target}$ en tant que mesure de vitesse qui sera utile pour le contrôleur en attitude.

Si nous pouvions construire un bon estimateur pour la vitesse de l'AUV (à partir d'une intégration fiable des accélérations mesurées par une éventuelle IMU embarquée par exemple), nous pourrions envisager de travailler en boucle fermée ce qui conduirait à une meilleure précision pour le contrôle de vitesse.

Contrôleur en attitude Ce contrôleur est divisé en deux parties : le guidage (les changements d'attitude attendus) et le pilotage (le calcul des commandes pour les actionneurs).

On recherche tout d'abord la vitesse de rotation que l'AUV devrait avoir autour de chacun de ses axes pour réaliser son objectif courant.

L'objectif pour la position angulaire de l'AUV dans le repère fixe est défini comme suit :

$$\begin{cases} \varphi_{target} = 0 \\ \theta_{target} = max(-0.1, min((z_{target} - \hat{z}), 0.1) \\ \psi_{target} = currentHeading \end{cases}$$

où $currentHeading$ est donné par la machine à état fini à partir des données du cycle stable trouvé à plus haut niveau.

Considérons un contrôleur qui permet d'obtenir :

$$\begin{bmatrix} \dot{\varphi} \\ \theta \\ \psi \end{bmatrix} = \begin{pmatrix} k_1 \\ k_2 \\ k_3 \end{pmatrix} . \begin{pmatrix} \varphi_{target} - \hat{\varphi} \\ \theta_{target} - \hat{\theta} \\ \psi_{target} - \hat{\psi} \end{pmatrix}$$

A partir de cet objectif nous pouvons déduire la vitesse de rotation que nous voulons autour de chacun des axes de l'AUV :

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = E'(\hat{\varphi}, \hat{\theta}, \hat{\psi})^{-1}.K. \begin{pmatrix} \varphi_{target} - \hat{\varphi} \\ \theta_{target} - \hat{\theta} \\ \psi_{target} - \hat{\psi} \end{pmatrix}$$

où

$$E'(\varphi,\theta,\psi) = \begin{pmatrix} 1 & \tan\theta.\sin\varphi & \tan\theta.\cos\varphi \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \frac{\sin\varphi}{\cos\theta} & \frac{\cos\varphi}{\cos\theta} \end{pmatrix}$$

Maintenant que nous avons les objectifs de rotation du sous-marin, nous voulons générer les commandes adéquates pour les atteindre.

D'après l'équation d'état, nous avons :

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = v.B. \begin{bmatrix} \sin(u_1) \\ \sin(u_2) \\ \sin(u_3) \end{bmatrix}$$

$$\text{avec } B = \begin{pmatrix} p_3 & 0 & 0 \\ 0 & p_4 & 0 \\ 0 & 0 & p_4 \end{pmatrix} . \begin{pmatrix} -1 & -1 & -1 \\ 0 & \sin(2\pi/3) & -\sin(2\pi/3) \\ 1 & \cos(2\pi/3) & \cos(2\pi/3) \end{pmatrix}.$$

Ainsi nous pouvons générer le vecteur de commande U tel que $\sin(U) = \frac{1}{\hat{v}}.B^{-1}. \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$.

Performances Afin d'évaluer de manière plus précise les performances des contrôleurs du Riptide, nous avons construit l'ensemble *système physique + contrôleurs* sous forme de schéma-bloc avec le logiciel *Scilab*. Une vue d'ensemble vous en est proposée figure 3.29. A partir de cette modélisation, on peut facilement vérifier différentes hypothèses pour l'implémentation de différents contrôleurs et plusieurs simulations peuvent être effectuées. Celles-ci permettent en particulier de mesurer les temps de réponse pour les différentes consignes ou la qualité des réponses pour des entrées sinusoïdales.

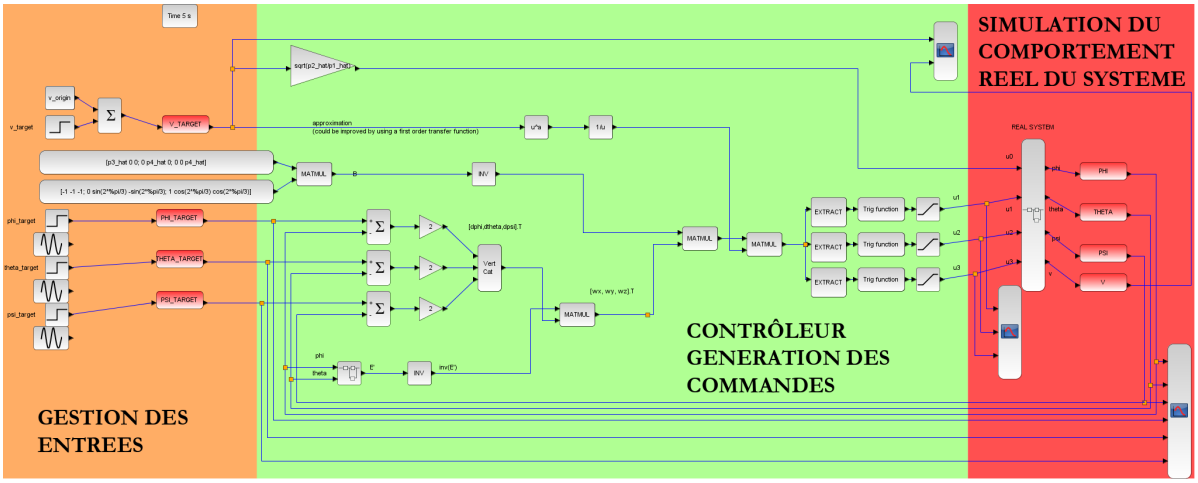

FIGURE 3.29 – Modélisation sous Scilab

Au moment où ces lignes sont écrites, les quatre paramètres qui décrivent le comportement dynamique des AUVs Riptide ne sont pas encore identifiés. On ne peut donc tirer de

conclusions quantitatives sur les simulations actuellement. Elles permettent toutefois de valider l'approximation faite dans le contrôleur en attitude de négliger les périodes transitoires en vitesse puisqu'elles ne sont pas source d'instabilité et causent simplement une augmentation légère du temps de convergence, comme le montre la figure 3.30. Ces simulations montrent également l'impossibilité de gérer séparément les trois axes de rotations à travers la visualisation des commandes générées pour atteindre les différents objectifs.

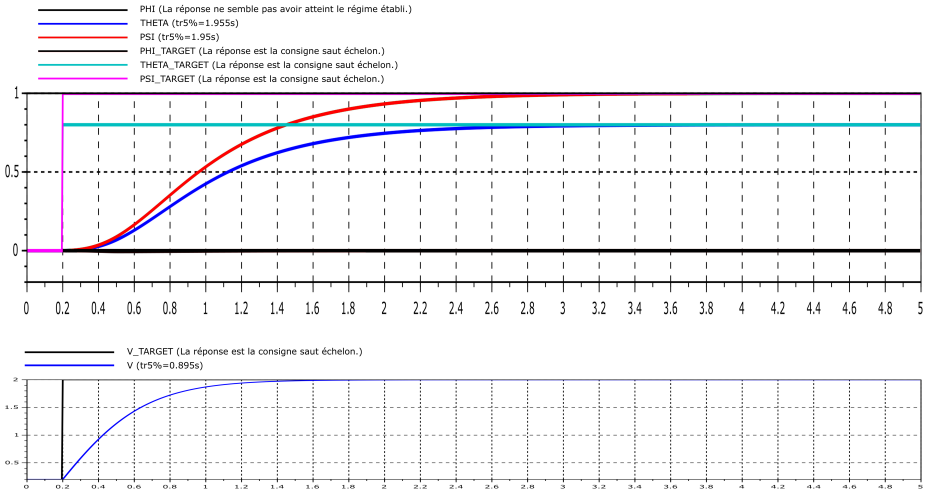

FIGURE 3.30 – Réponse impulsionnelle du système à une consigne d'attitude et de vitesse

Folaga Du fait de la disposition et de la nature de ses actionneurs, l'AUV de la start-up Kopadia présente une manoeuvrabilité bien plus importante que les Riptides. Ses pompes lui permettent par exemple de tourner sur lui-même y compris à vitesse nulle. Ces choix permettent notamment de contrôler chacun des axes de rotation indépendamment (contrairement à ce que l'on a pu observer précédemement dans le modèle *torpille* des Riptides), donc un simple contrôleur proportionnel sur chacun des axes de rotation permet de manoeuvrer convenablement ce sous-marin. Par conséquent, nous ne détaillerons pas plus le contrôle de cet AUV ici.

Validation Pour la validation du comportement du contrôleur, du comportement de la machine à états finis et de la cohérence des paramètres dynamiques mesurés, nous avons développé une première simulation 3D sous *Python* en attendant la version plus complète basée sur *Gazebo*.

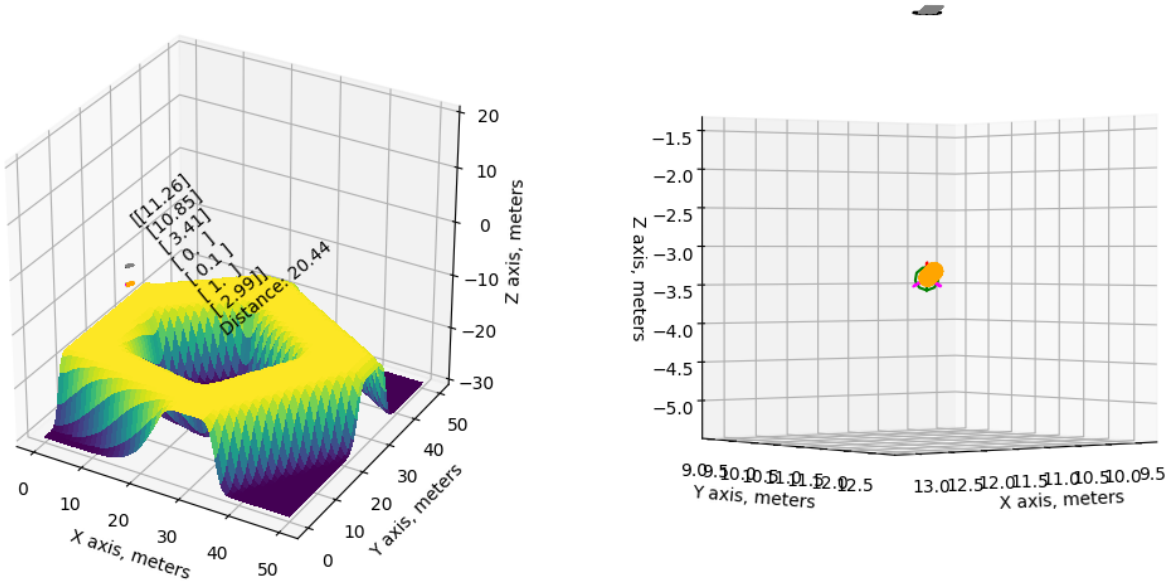

FIGURE 3.31 – Simulation sous Python. Le fond est choisi par l'utilisateur.

3.4.2 Simulateur 3D

3.4.2.1 Présentation du simulateur

Un des buts du projet "GoZone" était la capacité de pouvoir simuler une mission. Pour ce faire nous avons fait le choix d'utiliser le simulateur Gazebo. Gazebo est un simulateur 3D nous permettant une simulation réaliste de la physique. Le fait qu'il soit compatible avec ROS est un réel avantage pour nous, il nous permet ainsi de simuler entièrement une mission ou simplement d'en rejouer une à partir de "logs".
Lors de ce projet nous avions deux AUV, nous n'avons eu le temps que d'implémenter entièrement le comportement du Folaga. Nous avons, cependant, un modèle "visuel" du Riptide.
Sous Gazebo, lors de l'implémentation de n'importe quel robot il y a différentes étapes à respecter. Il faut commencer par définir le monde dans lequel ce robot va apparaître, nous avons fait le choix logique d'un monde type "océan". Nous avons ensuite commencer la création de notre robot, pour le Folaga nous avons juste récupérer un fichier tiré d'un logiciel de CAO, nous avons du faire le Riptide depuis un logiciel.
Afin de simuler des capteurs et des actionneurs ainsi que la physique inhérentes à notre AUV nous avons fait le choix d'utiliser le plugin **uuv simulator**.

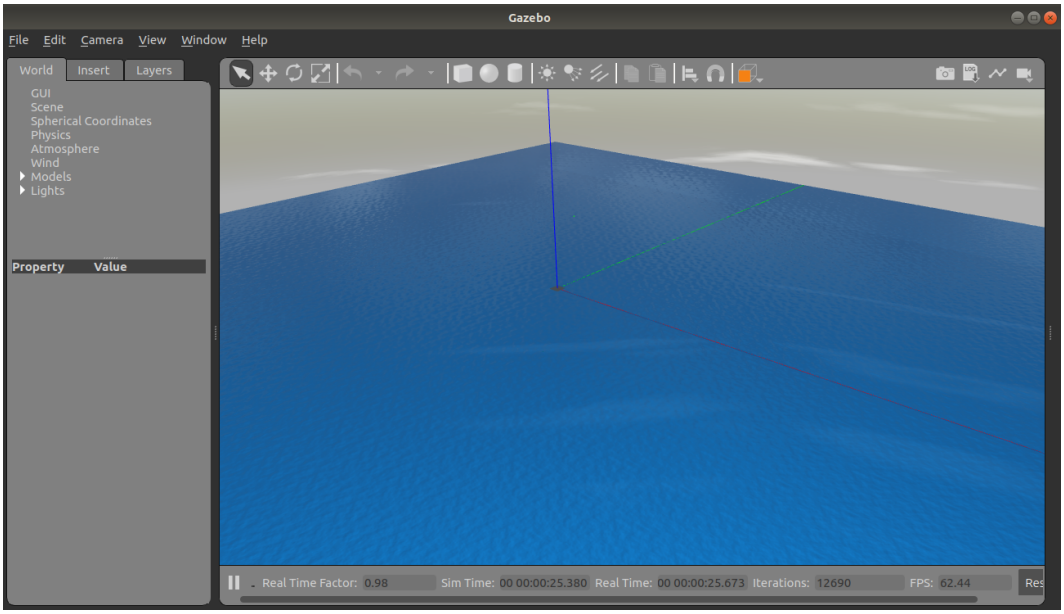

FIGURE 3.32 – Monde Ocean sous Gazebo

3.4.2.2 Implémentation du Folaga

Comme dit plus haut nous avons donc utilisé un fichier .stl afin d'implémenter facilement un modèle du Folaga.

3.4.2.3 Aspect visuel de l'AUV

Afin d'avoir une simulation réaliste nous avons rajouter une hélice pour la propulsion du Folaga. Nous avons aussi fait le choix de simuler les pompes de l'AUV avec des moteurs car leur fonctionnement reste très similaire.

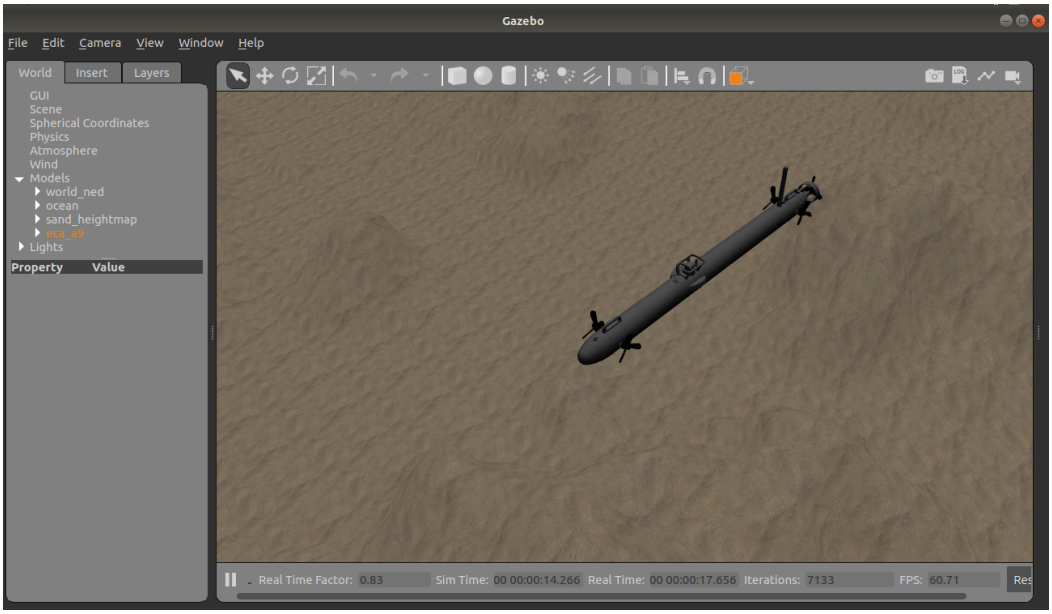

FIGURE 3.33 – Folaga sous Gazebo

3.4.2.4 Physique du Folaga

UUV Simulator Le plugin UUV Simulator est utilisé pour obtenir une dynamique cohérente du robot dans un milieu aquatique.

Actionneurs Le plugin permet tout d'abord de générer puis paramétrer les différents actionneurs. Nous avons modéliser les 4 pompes latérales du Folaga par des propulseurs. Ainsi, pour tourner à gauche (resp. à droite) nous actionnons simultanément les propulseurs avant-droit et arrière-gauche (resp. avant-gauche et arrière-droit).
Il est également possible de générer des ailettes (utile pour la modélisation du Riptide uniquement).

Modélisation des efforts UUV Simulator utilise les équations de mouvement de Fossen. Pour encore plus de réalisme, nous avons rajouté la matrice de masse ajoutée ainsi qu'une matrice Linear Damping pour simuler les frottements fluides. La poussée d'Archimède est également représentée.

Inertie du Folaga Nous avons décrit le Folaga par un cylindre plein. Nous obtenons alors une matrice d'inertie diagonale dont les valeurs dépendent de la masse, du rayon et de la longueur du sous-marins (valeurs connues).

3.4.2.5 Implémentation du monde depuis un MNT

Le monde que nous avons choisi pour la simulation est *ocean_ waves.world*. Ce monde nous permet de simuler simplement un océan. Malgré cela il y a la possibilité d'avoir un monde plus

fidèle à la réalité rencontré sur le terrain, nous pouvons créer un monde à partir d'un Modèle Numérique de Terrain.
Créer un monde à partir d'un MNT à beaucoup de sens quand le but du projet Go Zone est de "rebondir" sur des isobathes. Si nous désirons rejouer une mission nous pourrions alors nous apercevoir avec précision si l'AUV à bien rejoint l'isobathe désiré.

Chapitre 4

Expérimentation

4.1 Préparation de la mission

L'ensemble des travaux étant fonctionnel nous avons pu les mettre à l'épreuve lors d'une expérimentation.

4.1.1 Choix d'un lieu

Afin d'effectuer une expérimentation, deux principaux lieux étaient envisageables. Le premier étant la Penfeld à Brest, proche de l'ENSTA, le second le lac de Ty Colo à Saint Renan. N'ayant pas pu récupérer de modèle numérique de terrain (MNT) de la Penfeld, nécessaire aux algorithmes de la tâche 5 pour trouver des cycles stables, le choix s'est porté sur le lac de Ty Colo dont nous disposions d'un MNT. L'expérimentation a donc été programmée pour le Lundi 24 février 2020.

4.1.2 Préparation logicielle

La préparation logicielle consiste à trouver des cycles stables sur le lieu de l'expérimentation et de les implémenter dans les AUVs. Trois tâches sont principalement concernées dans cette phase. La tâche 5 qui est en charge de trouver les cycles stables, la tâche 7 devant gérer les requêtes telles que la navigation entre deux cycles stables et la tâche 1 devant programmer l'automate correspondant au parcours des cycles stables. Ce sont bien ces trois seules tâches qui doivent adapter leur résultat en fonction de la mission, les travaux fournis par les autres tâches sont dits statiques, une fois opérationnels ils non plus besoin d'être modifiés.

Ainsi si nous nous référons au schéma d'interdépendance des tâches [Figure : 2.1], nous obtenons le diagramme fonctionnel suivant :

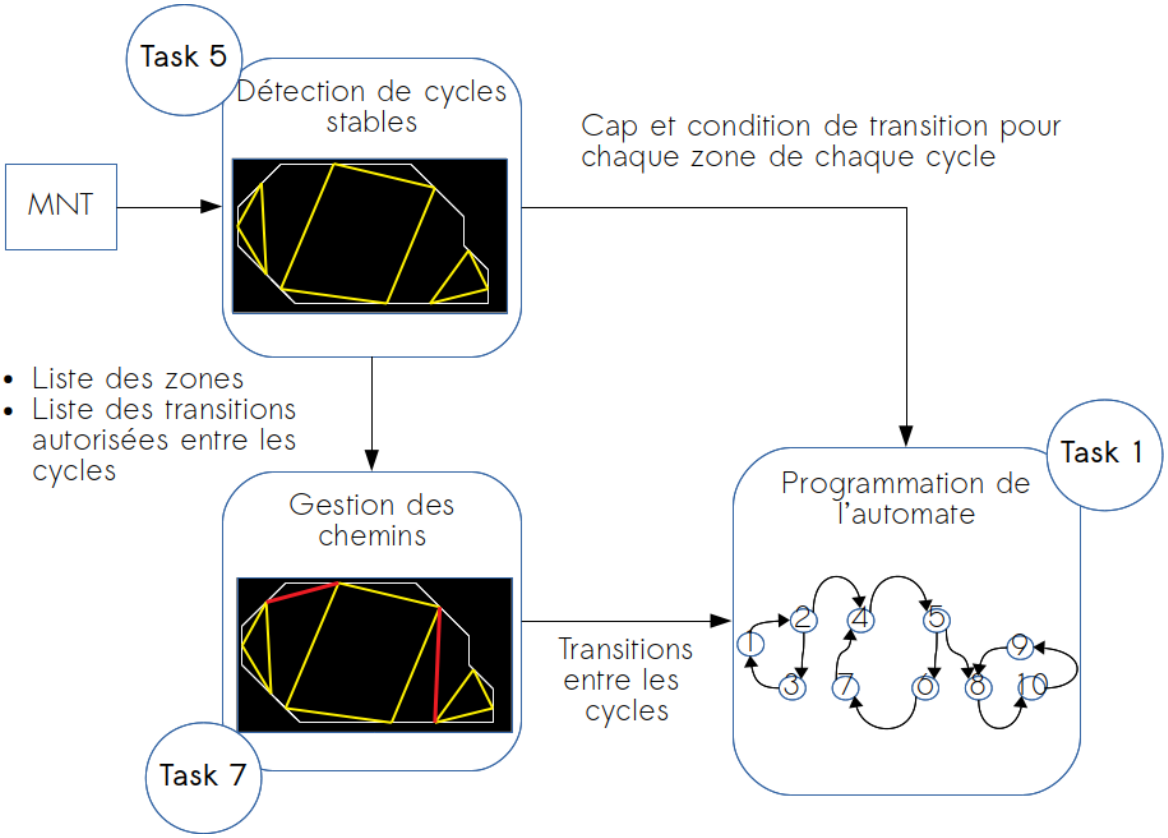

FIGURE 4.1 – Interdépendance initiale entre les tâches 5, 7 et 1

Ce qui est important de noter ici c'est que la Tâche 5 fournit des informations différentes aux tâches 7 et 1. De plus la programmation des différents cycles nécessite du temps. Afin d'optimiser ces échanges d'informations entre les tâches ainsi que le temps de programmation, deux programmes on été développés : **Cycle FSM descriptor** et **FSM Generator**.

Cycle FSM descriptor

Ce programme prend en entrée toutes les informations issues de la tâches 5 sous la forme d'un fichier YAML. Il va ensuite en extraire les informations nécessaires pour la tâche 7 et questionner le fichier Prolog généré par cette dernière (voir 3.3.2) afin de calculer pour tous les cycles trouvés, quel est le plus cours chemin pour passer de l'un à l'autre.
Supposons que pour que la transition trouvée entre une Cycle A et un Cycle B soit celle entre la Zone A du Cycle A et la Zone B du Cycle B. Le programme va alors calculer le cap nécessaire pour passer de cette Zone A vers la Zone B et repérer dans le fichier YAML quelle était la condition permettant de valider le fait que l'on soit arriver dans la Zone B.
Ainsi une fois toutes ces informations mise sous cette forme, le programme les combine avec celles d'entrée décrivant, quant à elles, la navigation dans un même cycle. Le tout permet de décrire la navigation entre n'importe que zone de n'importe quel cycle. Ce résultat est mis sous la forme d'un fichier dans lequel chaque ligne est composée de 4 éléments :

1. Zone de départ
2. Zone d'arrivée

3. Condition à vérifier pour considérer la transition entre la Zone de départ et la Zone d'arrivée
4. Cap à prendre une fois la transition effectuée

FSM Generator

Ce programme prend en entrée un fichier semblable à celui généré par le programme **Cycle FSM descriptor**. En lisant ce fichier il va générer un squelette Python définissant la FSM décrite dans le fichier d'entrée. Il faudra alors par la suite compléter les fonctions générées symbolisant les conditions à tester pour les transitions et celle à appeler une fois la transition effectuée.

Ainsi ayant développé ces deux programmes, le nouveau diagramme fonctionnel est :

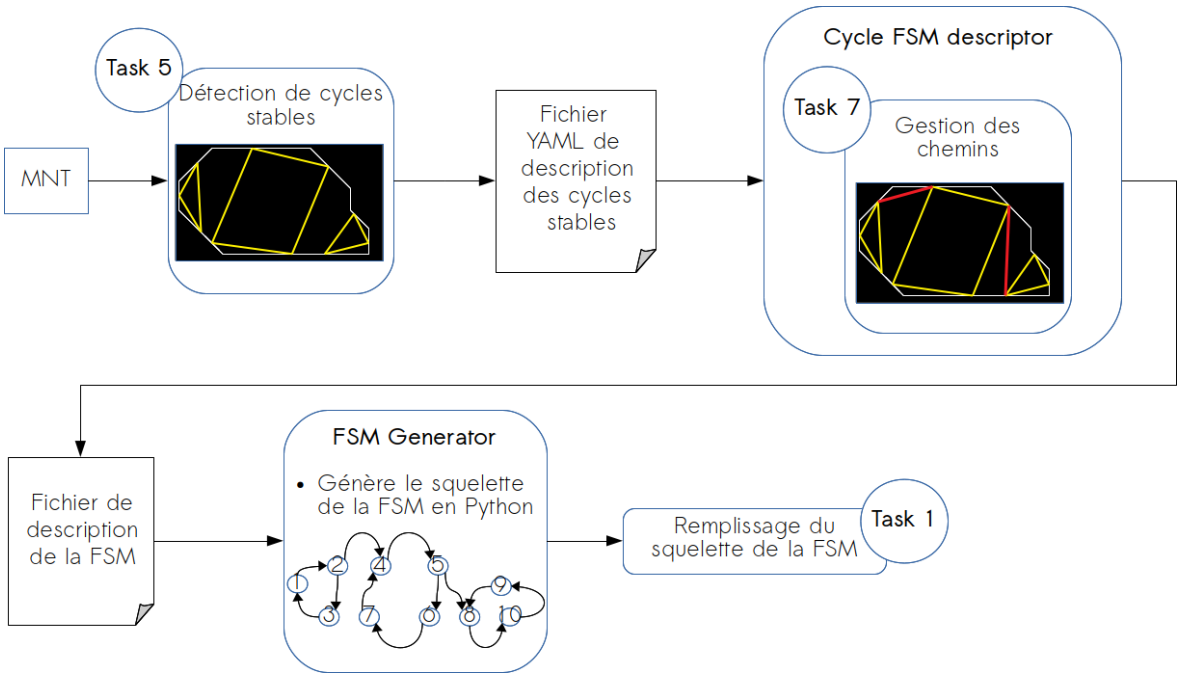

FIGURE 4.2 – Interdépendance finale entre les tâches 5, 7 et 1

On a pu ainsi automatiser une grande partie de la programmation des automates dès la description des cycles par la tâche 5. Ceci permet donc un gain de temps pour l'implémentation de la FSM dans les AUV ainsi qu'une meilleure répartition de l'information.

4.1.3 Cycles envisagés

Le MNT fourni, voir figure 4.3, représentait en réalité un fond "plat" d'environ 1.90m +/- 20cm de profondeur.

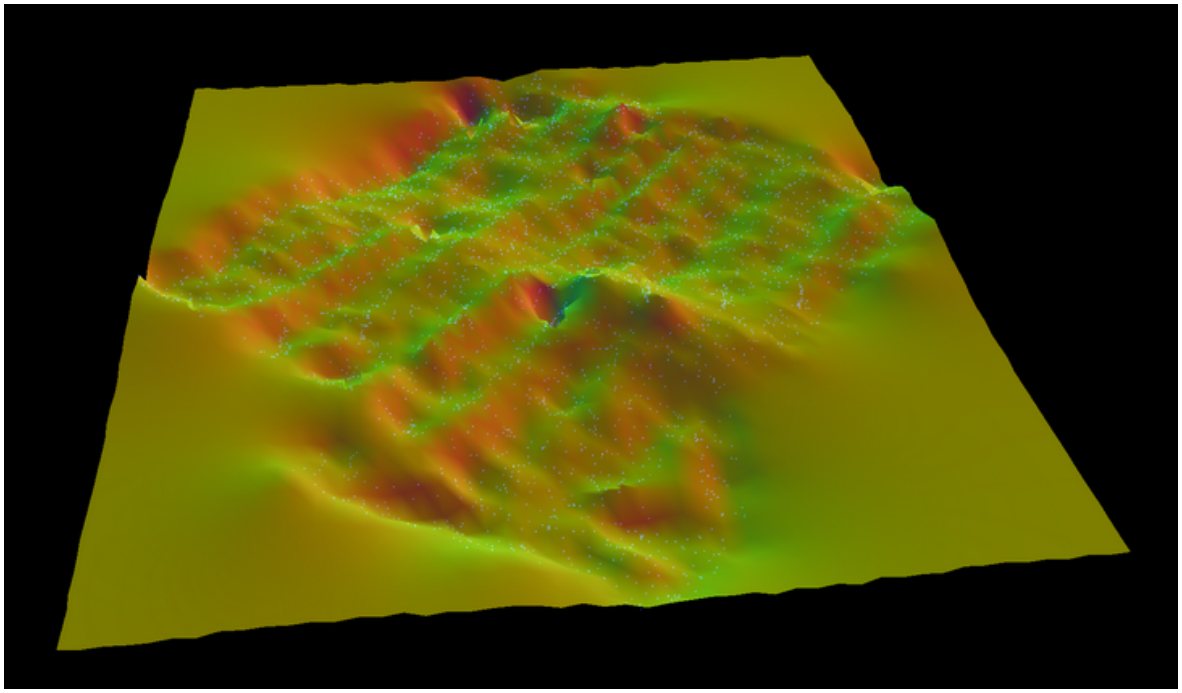

FIGURE 4.3 – MNT du lac de Ty Colo

Ceci était donc non exploitable par les AUV au vu de l'incertitude sur la valeur de mesure par échosondeur de ces derniers.

Nous avons donc choisi de tester des cycles en "L". Voici sous forme imagée un cycle en L :

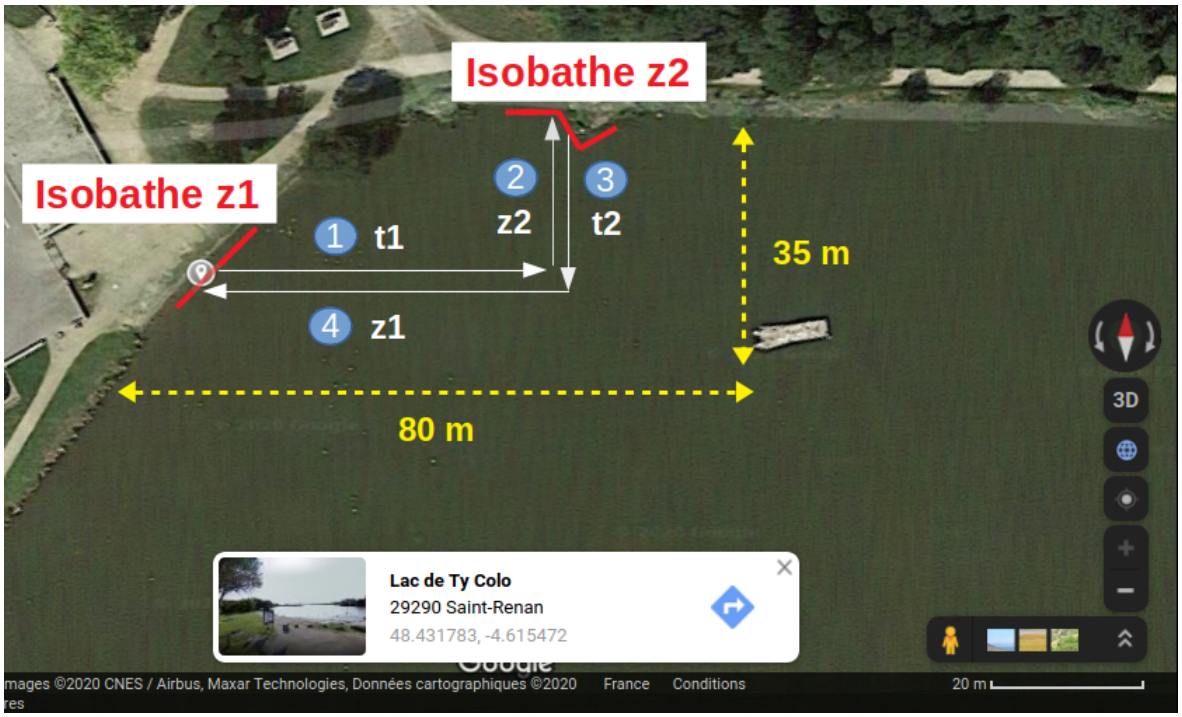

FIGURE 4.4 – 1er cycle stable en "L" sur le Lac de Ty Colo

Ainsi un cycle en "L" se décrit par quatre étapes :

1. Navigation vers l'Est pendant un temps **t1** préalablement définit.
2. Navigation vers le Nord jusqu'à l'isobathe **z2**, préalablement mesuré.
3. Navigation vers le Sud pendant un temps **t2**, correspondant au temps de parcours de l'étape précédente.
4. Navigation vers l'Ouest jusqu'à l'isobathe **z1**, préalablement mesuré.

Ces quatre étapes se répètent en boucle. De plus il est important de noter que le temps **t2** n'est mesuré qu'une seule fois et servira de référence pour les prochains parcours du cycle.

Une fois ce cycle envisagé nous avons pu en établir un second ainsi qu'une transition entre ces deux cycles comme décrit dans la figure ci-dessous :

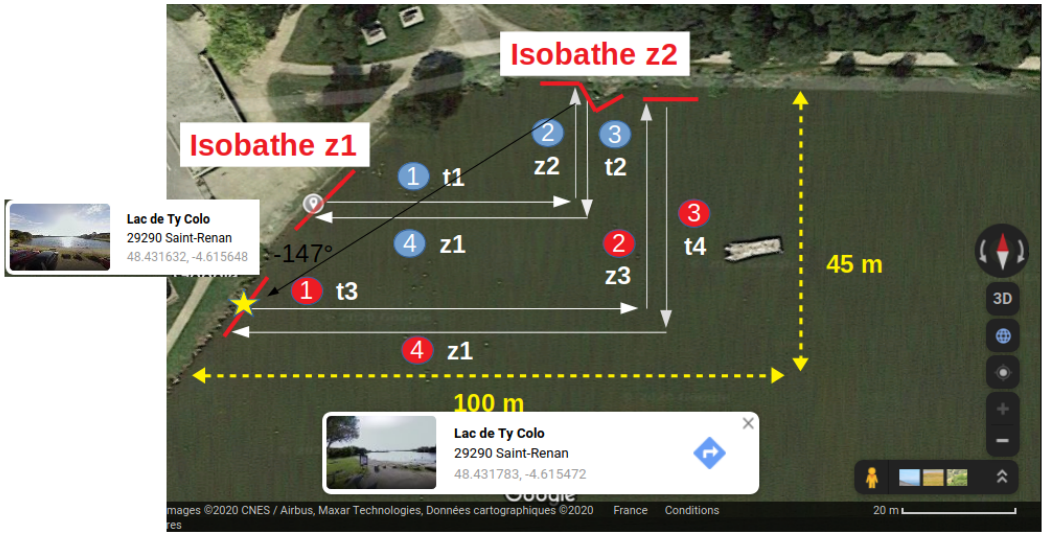

FIGURE 4.5 – 1er et 2nd cycle stable en "L" sur le Lac de Ty Colo

La transition entre les cycles s'effectue au niveau de la seconde étapes du premier cycle vers la première étape du second cycle, après avoir effectué N premiers cycle. Lors de cette expérimentation ce N a été choisi égal à deux. La programmation de ces cycles au sein des AUV a pu se faire en utilisant la préparation logicielle [4.1.2] précédemment décrite. Voici la machine à état résultante de cette préparation d'expérimentation :

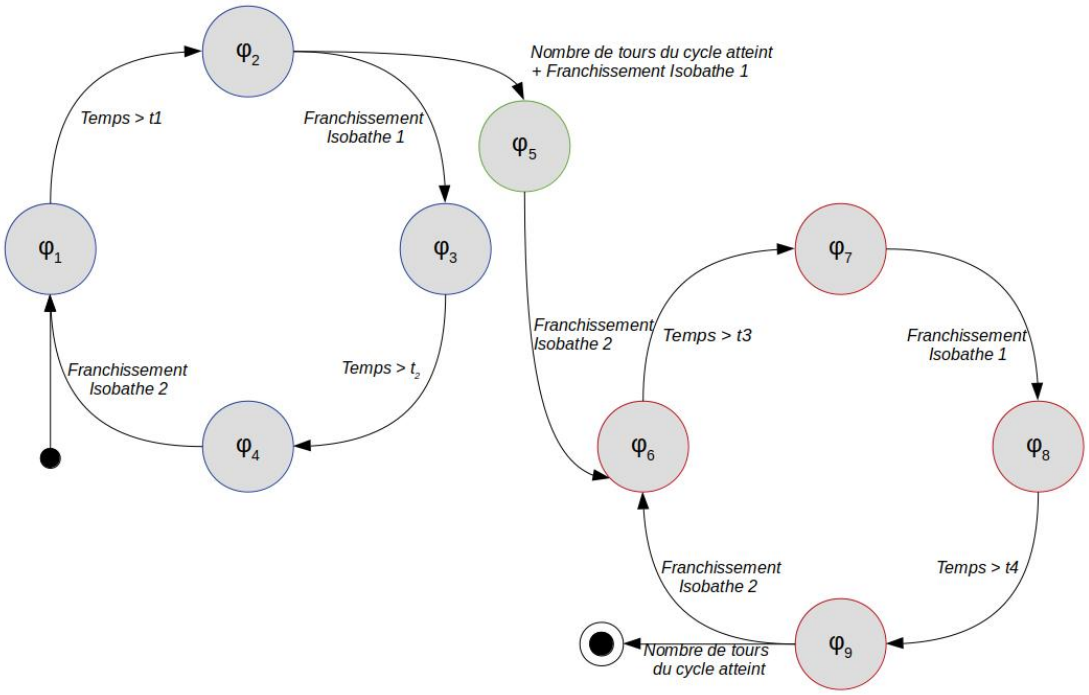

FIGURE 4.6 – Diagramme d'états-transitions des deux cycles en "L"

4.2 Exécution de la mission

L'expérimentation s'est bien déroulée le Lundi 24 Février 2020 sur le Lac de Ty Colo. Elle a commencé à 9h30 et s'est terminée à 15h30. Le but était ainsi de réaliser les cycles stables en "L" préalablement définis par le Folaga et le Riptide. Des étapes préliminaires ont été réalisées.

4.2.1 Manipulations préliminaires

4.2.1.1 Folaga

Les manipulations préliminaires nécessitent de tester les codes bas niveaux du Folaga dès sa mise à l'eau : bon fonctionnement du GPS, de l'échosondeur ...
Ensuite, les caps à suivre définis en simulation sont orientés par rapport à l'Est. Dans le code implémenté dans l'AUV, l'Est correspond à zéro degré, ce qui n'est pas forcément le cap que retournera la boussole. Un offset à donc était prévu, et a été complété au début des manipulations afin de coller au mieux à ce qui avait été prévu.

4.2.1.2 Riptide

Pour sa première mise à l'eau, les manipulations préliminaires du Riptide consistent à tester les programmes réalisés par le constructeur. Pour cela nous utilisons l'interface *web* du Riptide pour mettre au point des missions simples à réaliser, telles que des suivis de caps avec plusieurs vitesses de moteurs en surface et en profondeur. Cependant les tests en profondeur n'ayant pas fonctionné (extinction du robot au moment de plonger) nous n'avons pas réalisé plus de tests avec cet AUV.

4.2.2 Réalisation des cycles stables

Les manipulations préliminaires n'ayant pas été concluantes pour le Riptide, nous n'avons pas pu tester de cycles stables avec cet AUV. Ainsi seul le Folaga à pu se prêter à cette méthode de navigation.

Nous avons donc commencé par réaliser le premier cycle en L prévu. Le test étant concluant, le cycle a été effectué 6 fois entièrement dans un premier temps. Ensuite une seconde session de 6 cycles a été effectuée. Enfin nous avons testé la transition entre le "petit L" et le "grand L". Cette transition s'est bien effectuée mais le cap considéré ainsi que la bathymétrie réelle a amené l'AUV à effectuer un grand "L" dont la première partie est commune avec celle du petit "L". En se référant au schéma 4.5, nous pouvons noter que lors de l'expérimentation, les deux "L" se sont donc vus superposer.
La figure 4.7 est une photo prise lors de la réalisation d'un petit L. La forme du L ainsi que les zones d'arrivée et de départ y ont été ajoutées :

FIGURE 4.7 – Réalisation du "petit L" sur la lac de Ty colo

Enfin, l'ensemble des données ROS a été enregistré sous la forme de *rosbag*. Ceci nous permet donc de rejouer la mission via le simulateur développé.

4.3 Résultats

Pour le premier cycle en L, l'AUV a réalisé 6 fois un L. On remarque simplement deux amas de points verticaux sur l'image suivante. Le premier pic vertical correspond au premier L. Ensuite, le Folaga repart de l'isobathe 1.5m et non de la berge, ce qui lui fait aller plus loin qu'au L d'avant. Cela montre que même si le point de départ ne se trouve pas sur l'isobathe correspondant au demi tour, l'AUV converge bien vers le cycle stable.

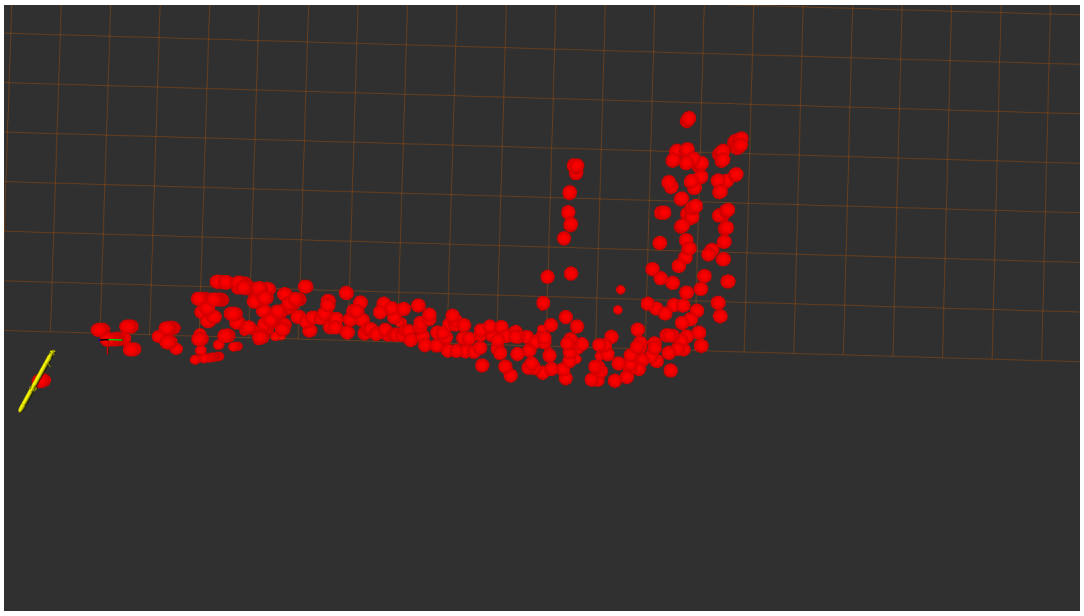

FIGURE 4.8 – Premier cycle en L

Pour le second cycle en L, l'AUV a réalisé 6 L. Cette fois, principalement dû au vent, on remarque que les trajectoires ne sont plus superposées. Néanmoins, on remarque un comportement de convergence, notamment vu à l'aide de la partie intervalle, qui se traduit par une trajectoire du Folaga qui est incluse dans un couloir englobant le cycle stable.

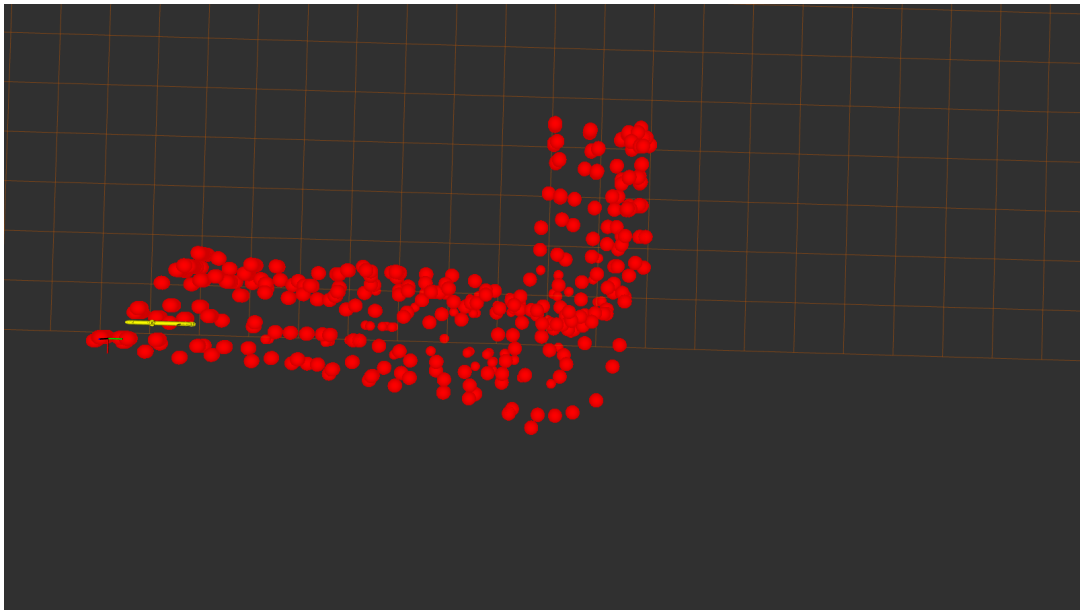

FIGURE 4.9 – Second cycle en L

Enfin la transition entre le petit L et le grand L s'est bien effectuée comme décrit précédemment. On observe bien dans le figure 4.10 la superposition des L, due au cap de transition. Il est important de noter que lors de ce test seulement 2 cycles de chaque L ont été réalisés. Ceci explique le manque de tracé franc pour le cycle, visible sur les deux cycles ci dessus.

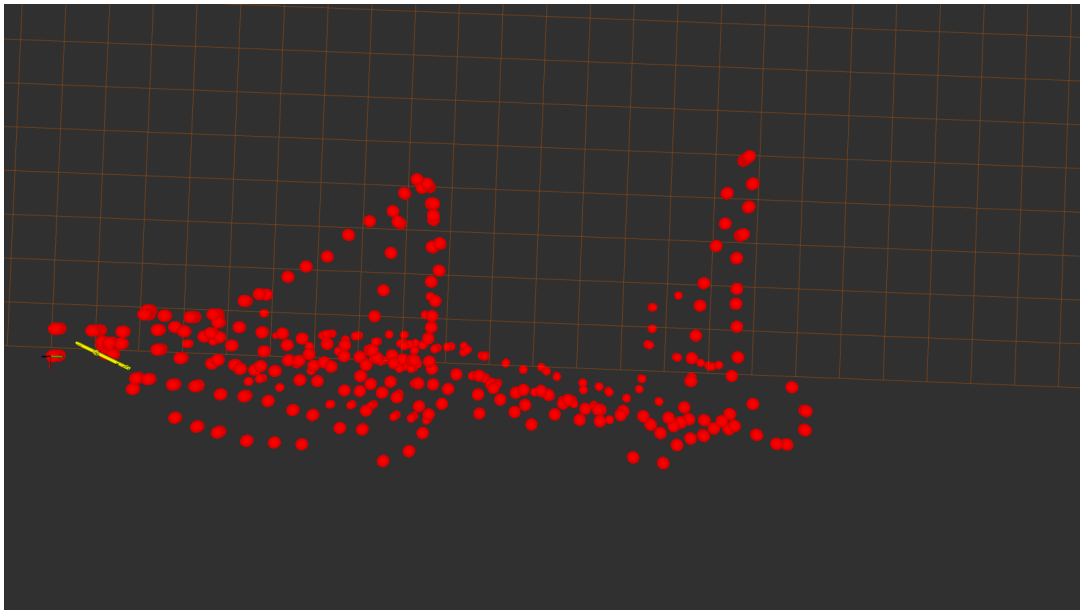

FIGURE 4.10 – Transition entre le petit L et le grand L

Les données de l'expérimentation ont également pu être utilisées pour l'estimation des paramètres des AUV ainsi que pour le paramétrage et réglage de la modélisation 3D effectuée sous Gazebo.

Chapitre 5

Conclusion

Au vu des résultats lors de l'expérimentation finale, nous pouvons considérer que le projet, bien que ambitieux, a été réussi. En effet, d'une part nous avons réussis à organiser le travail de façon coordonnée et cohérente. L'ensemble du travail fourni a permis d'aboutir à l'implémentation et à la réalisation de deux cycles stables par le Folaga. Nous nous sommes adaptés aux imprévus tels que l'impossibilité de réaliser un cycle à partir du MNT du lac de Ty Colo, ainsi que les difficultés survenues et subsistantes pour la prise en main des AUV de l'entreprise Riptide. D'autre part, même si les cycles stables réalisés sont de type en "L", le concept de navigation sur zone grâce à cette méthode a pu être validé.
Cependant, de nombreuses améliorations dans les travaux fournis sont possibles et restent à étudier. Notamment en ce qui concerne la modélisation des AUV ainsi que le comportement des actionneurs sur le simulateur 3D Gazebo, pouvons-nous trouver des modèles plus fidèles ? De même l'analyse de la stabilité par Tubex pourrait se baser sur les équations d'états des AUV et non pas sur le modèle de Dubins comme nous l'avons effectué par manque de temps. Enfin, il aurait été intéressant d'automatiser l'ensemble de la chaîne aboutissant à la programmation d'un automate parcourant les cycles stables en partant d'un MNT. Ce travail n'est à ce jour que partiellement automatisé, comme nous vous l'avons décrit dans ce rapport.
Au final nous avons tout de même réussi à gérer le temps qui nous a été imparti et fournis des résultats plus qu'encourageants pour la navigation sur zone d'un AUV par méthode des cycles stables.

Annexes

Diagramme de Gantt

La tâche 4 concernant les capteurs est toujours en attente de réponses de la société Riptide que nous avons sollicité concernant l'accès aux codes sources des AUV ainsi que les références des capteurs et actionneurs utilisés.

Bibliographie

[1] JB Baillon. Quelques aspects de la theorie des points fixes dans les espaces de banach-ii. *Séminaire Analyse fonctionnelle (dit" Maurey-Schwartz")*, pages 1–32, 1979.

[2] Thor I Fossen. *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.

[3] Luc Jaulin. *Mobile robotics*. John Wiley & Sons, 2019.

[4] Simon Rohou, Luc Jaulin, Lyudmila Mihaylova, Fabrice Le Bars, and Sandor M Veres. Guaranteed computation of robot trajectories. *Robotics and Autonomous Systems*, 93 :76–84, 2017.