Rapport d'Architecture Robotique

Projet Cordelière
U.V. 5.7

Promotion Robotique 2018

Sous l'encadrement de :
Luc JAULIN, Enseignant-chercheur, ENSTA Bretagne

Avant-Propos

L'essentiel du travail réalisé est disponible sur GitHub aux adresses suivantes :

— Le projet global :
https://github.com/EnstaBretagneClubRobo/Cordeliere
— Le projet ROS :
https://github.com/EnstaBretagneClubRobo/Cordeliere_ROSws
— Le projet PyUnityVibes :
https://github.com/EnstaBretagneClubRobo/PyUnityVibes

Remerciements

Nous tenons à remercier les personnes suivantes pour leur implication dans ce projet :

— Michel L'HOUR, directeur du DRASSM : Pour l'initiative du projet.
— Max GUÉROUT : Pour la ré-utilisabilité de son travail.
— Luc JAULIN : Pour son aide sur les calculs par intervalles.
— Benoît DESROCHERS : Pour son aide sur les calculs par intervalles.
— Thomas LE MÉZO : Pour son aide sur l'architecture en arbre de l'analyse de la carte.
— Olivia HULOT, archéologue ayant travaillé en collaboration avec la robotique.
— Vincent CREUSSE, chercheur au LIRMM en robotique archéologique : Pour s'être tourné vers l'ENSTA Bretagne.
— Denis DEGEZ, responsable technique du DRASMM : Pour son aide concernant les magnétomètres.
— Samuel YOUNG, responsable modélisation chez ILM pour Pirates des Caraïbes : Pour son aide dans la modélisation de la *Cordelière*.
— Éric RIETH, conservateur au musée de la marine de paris : Pour nous avoir donné accès à certains documents.
— Sabrina MARLIER, Archéologie chargée de mission au Musée départemental Arles antique : nous a aidée à comprendre l'Archéologie sous-marine.
— Romain SCHWAB, thésard à l'ENSTA Bretagne : Pour son travail sur la recherche des objets métalliques en fond marin à l'aide de magnétomètres.
— Fabrice LE BARS : Pour son aide concernant les questions matérielles.
— Benoît ZERR
— Gilles LE CHENADEC
— Hervé DE FORGES
— Arnaud COATANHAY

Nous tenons également à remercier les organismes suivants ayant pris part au projet :

— Le DRASSM : Pour avoir relancé le projet et s'être tourné vers l'ENSTA Bretagne.
— Le SHOM : Pour nous avoir permis de faire des tests magnéto-métriques et nous avoir donné accès à leurs cartes bathymétriques.
— Le GRAN : Pour leur travail en lien avec Max GUÉROUT.
— Le LIRMM : Pour s'être tourné vers l'ENSTA Bretagne.
— La région Bretagne : Pour les financements liés au projet.
— L'ENSTA Bretagne

Résumé

Dans ce rapport, la promotion Robotique 2018 de l'ENSTA Bretagne résume son travail de l'U.V. 5.7 (Architecture Robotique), dont l'objectif est la définition d'une architecture robotique dans le but de retrouver la légendaire *Cordelière*, navire français ayant coulé dans le goulet de Brest au XVI^{ème} siècle. Elle présente une solution de localisation par intervalles d'un AUV [1] dans un environnement connu, une étude sur une solution magnétique permettant la détection de cette épave ainsi qu'une librairie graphique pour permettre une simulation vidéo de qualité et une aide au développement.

Mots clefs : Cordelière, AUV, localisation, calculs par intervalles, ROS, Unity [2]

Abstract

This report concerns the 2018 ENSTA Bretagne Robotics promotion's research of solutions and achievements on the work made for the U.V. 5.7 - Robotics Architecture -, in which the goal is to define a Robotics Architecture in order to find the legendary *Cordelière*, a French battleship which sank in the Brest bottleneck during the XVIth century. In this document they introduce an interval analysis way to localize an AUV in a known environment, they make a study on a magnetic solution to localize the shipwreck and they bring a graphical library that generates a simulation of the mission and that is a great help for the development.

Key-Words : Cordelière, AUV, localization, interval analysis, ROS, Unity

1. Autonomous Underwater Vehicle : sous-marin autonome
2. Moteur graphique et physique pour jeux-vidéo que nous utilisons pour le rendu graphique de la simulation

Sommaire

Introduction

Présentation du projet

Partie réalisée par Sophie TUTON et Maha ABOUZAID

La Région Bretagne [1] a mis à l'honneur la thématique de l'archéologie subaquatique et sous-marine dans le but d'encourager la découverte et l'innovation liées à ce domaine, et le DRASSM va réaliser une campagne archéologique pour retrouver la *Cordelière* échouée dans la rade de Brest depuis le 10 août 1512.

C'est dans ce contexte que le travail réalisé dans le cadre de l'U.V 5.7 de la spécialité Robotique de l'ENSTA Bretagne s'inscrit. Les étudiants de 3ème année sont amenés à travailler jusqu'au 9 Mars sur un projet qui a pour objectif de retrouver la *Cordelière* à l'aide de robots sous-marins. Il s'agit principalement d'une pré-étude sur simulation qui devrait mener à la réalisation de premières mesures. Ce projet a également pour objectif de promouvoir le concept d'Archéologie robotique qui vise à automatiser l'exploration de zones très vastes et peu hospitalières à l'aide de robots afin d'y trouver des sites archéologiques.

Présentation du travail réalisé

Ce projet sera mené à bien à travers différentes tâches telles que :

— L'étude de l'architecture navale et du déroulement de la bataille pendant laquelle la *Cordelière* s'est échouée.
— La prise de contact avec des interlocuteurs de proximité (historiens, archéologues, géologues de la rade, roboticiens, chasseurs d'épave,...).
— Le contrôle de suivi d'une isobathe.
— La planification de trajectoires.
— La localisation.
— La méthode de détection par anomalies magnétiques.
— La simulation du système.
— La gestion de projet et de versions de logiciels développés.

1 Partie historique

1.1 L'architecture navale à travers l'Histoire

Partie réalisée par Raphaël ABELLAN–ROMITA

1.1.1 Des Cogues aux Caraques

Au début du XV^{ème} siècle se passe un changement de paradigme dans l'architecture navale qui révolutionnera à jamais l'histoire de la navigation et lancera l'époque des grandes découvertes. Des Cogues, bateaux à font plat et à voiles carrées et incapables de remonter au vent, on crée, en incorporant des éléments de bateaux présents en Méditerranée et en changeant le mode de fabrication, les Nefs et les Caraques. [2]

Pour ces bateaux, le premier élément à être jeté [3] est la quille, grande nervure centrale qui court de la figure de proue au haut du château de poupe. Cet immense morceau de bois solidifie la structure, et permet de contrecarrer la dérive naturellement provoquée par un vent venant de travers, ou même partiellement de face. Ainsi, les allures rapprochées sont rendues accessibles, et les marins deviennent capables alors de dépasser le cap Boujdour, alors appelé le cap de la Peur.

Le deuxième grand changement fut l'augmentation du nombre de mâts. D'un ou deux simples mâts chargés d'une, éventuellement deux voiles carrées comme ce fut le cas lors de la 2ème croisade au XII^{ème} siècle, le nombre de mât monte à trois, puis quatre, tandis que les mâts se chargent de plus en plus, avec deux, puis trois voiles par mâts.[3]

Ces changements ont permis l'apparition des Nefs et Caraques, dont la différence est plus géographique que technique, Nef venant du portugais "Nao", tandis que Caraque vient de l'anglais "Carrack". C'est grâce à ces bateaux que les grands explorateurs comme Christophe Colomb ou Vasco de Gama ont pu faire leurs grandes traversées.[4]

1.1.2 L'architecture générale du navire

Au vu des différentes informations recueillies tantôt auprès des prisonniers de la bataille [5], tantôt en s'inspirant de bateaux contemporains [6], on peut reconstituer une architecture générale du bateau.

Comme on peut le voir dans l'image en annexe A, figure 36, le gréement de la *Cordelière* fait partie des cas les plus avancés pour l'époque, avec deux mâts de misaine, portant un total de trois voiles latines. Même si tous les écrits ne s'accordent pas quant au tonnage du bateau, la présence de ces nombreuses voiles nous font supposer que le bâtiment faisait au moins 600 tonneaux.

1.1.3 La sainte-barbe et le bordage

Parmi les découvertes qui ont fortement influencé la forme des bateaux de l'époque, un autre changement dans la façon de construire un bateau est le passage du bordage à clin au bordage à franc-bords. Jusqu'au XV^{ème} siècle, les planches (ou virures de bordé) latérales des coques de bateau se recouvrent l'une l'autre pour assurer l'étanchéité de la coque. [7] On appelle cela le bordage à clin.

3. Dans le cas de la fabrication d'un bateau, on ne pose pas les premiers éléments, on les "jette".

Avec l'amélioration des techniques de charpenterie et d'étanchéification des navires à l'aide de goudron et de bourre de chanvre, on commence à pouvoir assembler les bordés de telle sorte qu'elles ne se recouvrent plus. Cela permet une plus grande cargaison interne pour le même volume externe, entre autres.[8] Mais le changement n'est pas qu'à propos des bordés. La nouvelle méthode de construction, associée à la présence d'une quille et de membrures, grandes pièces de bois suivant la forme du bateau selon un plan transversal, rigidifient très fortement la structure.

De cette rigidité découle deux choses. Tout d'abord, la résistance nécessaire pour lever des mâts plus hauts, et surtout prenant le vent de travers, comme c'est le cas pour la *Cordelière* avec ses deux mâts d'artimon. Sur ces mâts sont placés des voiles latines, c'est à dire triangulaires et orientées dans le sens de la marche. Ce sont les voiles d'artimon et de contre artimon, qui permettent au bateau de remonter au vent. De plus, les ponts, c'est à dire les étages du bateau, sont plus résistants et peuvent porter des charges plus lourdes. C'est ainsi qu'apparaissent les sabords. Les sabords dans un bateau sont ces ouvertures latérales qui permettent de sortir les canons, apportant ainsi la possibilité d'emporter plus de canons, en les mettant sur plusieurs ponts. La construction en franc bord permettant de faire des ouvertures dans la coque, et renforcent la charge que peut porter chaque pont.

De ces nouvelles techniques de charpenterie navale, on obtient donc des bâtiments capables de remonter au vent, d'affronter des mers plus violentes, d'emporter plus de cargaison et d'emporter de l'artillerie lourde.

Face à ces possibilités d'emport d'artillerie, un nouveau compartiment est créé dans les vaisseaux dédiés à la guerre ou à naviguer dans des eaux troubles : la sainte-barbe. Cette pièce est située, selon le capitaine Antoine De Conflans dans son traité *'les faits de la marine et navigaige'*, référence sur la marine des XV^{ème} et XVI^{ème} siècles,[9] sous la cabine du capitaine, et à coté de celle du quartier maître.

Les murs sont normalement recouverts de fer ou de bronze, selon la richesse de l'armateur, et la quantité de poudre emportée. La poudre est stockée dans des boites de fer appelées chambres, tandis que les matières premières servant à fabriquer la poudre noire, comme le charbon, le soufre et le salpêtre sont stockées séparément, dans des tonneaux. C'est celle-ci qui explosa lors du combat qui entraîna *Cordelière* et *Régent* par le fond.

1.1.4 Les inventaires des bateaux

Après une rapide analyse des objets traditionnellement emportés sur ces navires et suite au fait que le magnétomètre qui serait utilisé ne peut que repérer des objets métalliques et de forte taille, l'analyse des inventaires des bateaux s'est concentrée sur les plus grosses pièces de métal, c'est-à-dire les ancres et les canons.

Le premier inventaire, qu'on peut espérer le plus précis et fiable, est celui du *Henry Grace à Dieu*, vaisseau anglais lancé en 1514, dont un inventaire fut fait au moment de son lancement, et dont il est retranscrit ici les informations principales. [10]

Cela dit, ce bateau fut lancé deux ans après la bataille, mais ces informations nous donnent des équivalences de poids quant aux noms employés par les artilleurs marins anglais et français, les termes marins étant très proches dans nos pays où les évolutions se sont faites en même temps.[11]

Le second inventaire, cette fois-ci d'un bateau engagé dans la bataille, est celui du *Régent*. Datant de 1492 et rapporté par M. Oppenheim [12], cet inventaire très

Nom de la pièce	Nombre	Poids du projectile (lbs.)	Poids de la pièce (tonnes)
CANNON	4	60	5
DEMI-CANNON	3	32	3.5
CULVERIN	4	18	1.8
DEMI-CULVERIN	2	8	0.7
SERPENTYNE	4	6	0.3
CANNON PERER	2	26	1.8
FALCON	2	2	1

Table 1 – Tableau des pièces d'artillerie de l'*Henry Grace à Dieu*.

complet, de plusieurs centaines de pages et décrivant autant les vivres que les salaires des matelots, nous intéresse pour ces deux passages :

Serpentynes of Brasse xxxj & yron cxciiij
xv with theyr miches Boltes & forelokkes. ccxxv
Chambres to the same of Brasse Ij & Iron ccccxxxj . dmjij
[12], p 274

Dans ce passage, nous voyons que l'armement du *Régent* se limitait à un grand nombre (225) de serpentynes, qui, si l'on se réfère au tableau précédent 1 étaient de petits canons de 300 kg chacun.

Ankers of diuerse sortes that is to say :
Crete ankers of Hampton j
Shute ankers j
Sterborde Bowers ij
Lateborde Bowers ij
Sterborde Destrelles j
Caggyng Ankers j
Warpyng Ankers j
Destrelles broken j
Ankers very lytle for the seid Ship v
Crete Bote Ankers j
in all xvj
[12], p 290

Pour ce qui est des ancres, on peut retenir de cet inventaire :

1. La *Great anchor of Hampton*, ancre principale pouvant peser jusqu'à 5 tonnes.
2. La *Sheet Anchor*, très grosse ancre d'urgence, toujours prête à être larguée pour arrêter le vaisseau dans sa course au besoin.
3. Les *Starboard and Port Bowers*, au nombre total de quatre, sont les ancres à peine plus petites (de l'ordre de 2 tonnes chaque) et présentes à raison de deux de chaque coté du vaisseau.

Du coté français, pas d'inventaire aussi précis, ils furent majoritairement perdus suite aux complications politiques qui secouèrent la région durant les XV^{ème} et XVI^{ème} siècles. Cela dit, on peut s'appuyer sur les informations données par le capitaine Antoine De Conflans dans son traité *'les faits de la marine et navigaige'* [9]

Pour armer vne nef de cinq cens tonneaulx est requis l'artillerie qui s'ensuict : Est besoisg au belle de la nef Deux canons serpentins, Deux grandes couleuurin es et deux bastardes qui sont six pièces pour la belle.

Puys au chasteau -gaillard Deux couleuurines moyennes, Deux canons serpentins et six faulcons qui seruiront tant aux chasteaulx que dans les basteaulx, tant aux descentes que a lever ou mectre les ancres .

C'est le tout saize pièces de bronze : Quatre canons , deux grandes couleuurines, quatre bastardes et six faulcons qui sont les saize dessus dicts.

Il mentionne plusieurs très grandes pièces d'artillerie, mais celles-ci seraient en bronze. Ces recommandations semblent respectées par les artilleurs si l'on en croit deux sources qui ont envoyé des lettres présentes dans *Letters and Papers relating to The War in France 1512-1513* [5]

Antonio Bavarin to Francesco Pesaro. (London : 5 Sept. 1512.)

Qui serano notadi i homeni e ordinanza et altro, erano sopra la grande charachia de Brest del re di Franza [...].300 cavalieri zentilhomeni. 800 tra marinari e soldati. 50 bombardieri. 100 balestrieri. 400 pipe de biscoto. 100 pipe di carne salata. 16 bombarde grossissime di bronzo sopra charete, et altre bombarde, schiopeti e archibusi senza numero. 160 l meze bote di polvere di bombarda. 2 . . ., di portata di 40 bote, Y uno di bronzo. Questo se ha dal nochiero e altri presi vivi. Oltra questo, cadene d' oro de cavalieri e danari contadi per valuta grandissima.

[5], p55

Est ici mentionné 16 canons de grande taille en bronze sur affût [4]

Wolsey to the Bishop of Worcester. (Farnham : 26 August [1512].)

And after innumerabyll shotyng of gunnys and long chasyng one another, at the last the Regent most valyently bordyd the gret caryke of Brest, wherin wer 4 lords, 300 gentylmen, 800 solgers and maryners, 400 crossbowemen, 100 guners, 200 tonnes of wyne, 100 pypys of befe, 60 barells of gonepowder and 15 gret brasyn cortawds with so marvelose nombyr of schot and other gunys of every sorte.

[5],p50

De même, dans cette lettre, sont mentionnés 15 grosses pièces de bronze.

On propose donc que l'artillerie présente sur la *Cordelière* était comme suit :

Nom de la pièce	**Nombre**	**Poids du projectile (lbs.)**	**Poids de la pièce (tonnes)**
CANON	4	60	5
COULEUVRINE	2	18	1.8
DEMI-COULEUVRINE (Batardes)	4	8	0.7
FAUCONS	6	2	1

Table 2 – Tableau des pièces d'artillerie supposées de la *Cordelière*.

Cela dit, le fait que ces canons soient faits en bronze est important car certains magnétomètres ne sont capables que de repérer les métaux ferreux, ce qui exclut donc des objets d'intérêt l'ensemble des pièces de bronze, soit l'ensemble de l'artillerie de la *Cordelière*.

4. Bâti en bois ou métallique qui supporte le plus souvent un canon ou une pièce lourde.

Pour ce qui est des ancres, on suppose que la *Cordelière* possédait les mêmes grandes ancres que le *Régent*, soit une grande ancre de proue, une ancre de miséricorde et quatre ancres de mouillage. cela dit, entre l'explosion de la sainte-barbe et le fait que plusieurs de ces ancres furent utilisées lors de la bataille, certaines ancres ne sont probablement plus avec le reste de l'épave, dont notamment la grande ancre et certaines des ancres de mouillage, situées plus à la poupe du navire.

Enfin, il faut prendre en compte l'érosion et la rouille sur ces pièces de fer. les navires datant de la même époque qui furent retrouvés font majoritairement partie des grandes expéditions de la marine espagnole et portugaise. [4] En moyenne, il restait de l'ordre de 30% des grandes pièces de fer telles que les ancres ou les canons de ces épaves qui furent retrouvées durant le XX^{ème} siècle.

Les objets d'intérêt qui nous permettraient de repérer l'épave de la *Cordelière* et celle du *Régent*, sont donc :

Nom de la pièce	Nombre	Poids estimé (tonnes)	Poids restant (tonnes)
Ancre de proue	1	4	1.3
Ancre de miséricorde	2	3	1
Ancre de mouillage	6	2	0.7

Table 3 – Tableau des pièces d'intérêt potentiellement présentes sur le lieu du naufrage

1.2 Le combat de la *Cordelière*

Partie réalisée par Yoann GUGUEN

1.2.1 Les origines de la bataille

En 1512, année de la bataille de Saint-Mathieu, la France est empêtrée dans la quatrième guerre d'Italie, et est menacée par la Sainte Ligue tout juste formée par le pape. L'Angleterre a rejoint cette coalition qui a pour objectif de calmer les ardeurs françaises en Italie. De plus, les nombreuses captures de navires anglais par Hervé de Portzmoguer (le capitaine de la *Cordelière*) ont entraîné une volonté de représailles de la part de l'amiral Edward Howard.

Ce seront les attaques de celui-ci dans le Nord de la France puis en Bretagne (notamment dans le château de Hervé de Portzmoguer, lui permettant d'obtenir une première revanche) qui forceront le roi de France à réagir et à réunir sa flotte pour préparer la guerre contre les Anglais. La France et la Bretagne (encore dissociées à cette époque) réunissent leurs navires dans la rade de Brest et travaillent leur coordination en vue de la guerre qui s'annonce.

1.2.2 Le début du combat

Le 9 Août, les guetteurs bretons aperçoivent la flotte anglaise de l'amiral Howard se diriger vers le Sud. D'après les informations, les forces en présence seront équilibrées, ce qui amène les Français à préparer le combat plutôt que de se terrer pour

l'éviter. Les navires français et bretons sortent donc de la rade de Brest et s'amarrent vraisemblablement dans l'anse de Camaret pour être préparés en cas d'attaque anglaise. Les anciennes hypothèses supposaient que la flotte était stationnée dans l'anse de Bertheaume, mais la nouvelle étude des marées et des vents nous laissent maintenant supposer que la flotte était belle et bien dans l'anse de Camaret (voir annexe page 58).

Le 10 août vers 11h, les navires anglais sont repérés, la *Mary James* et la *Mary Rose* en amont de la flotte. La flotte anglaise semble deux fois plus importante, mais sur la cinquantaine de navires anglais (contre 22 navires français), il y a 26 hourques flamandes prises par Howard auparavant qui ne transportent que des vivres et des munitions. Les français n'ont pas cette information et décident donc de revenir au plus vite dans le goulet de Brest pour se mettre à l'abri. 19 navires y parviennent, mais la nef de Dieppe, la *Louise* et la *Cordelière* se retrouvent à combattre les Anglais. Est-ce parce que leurs manœuvres d'appareillage ont été plus longues ou bien parce qu'elles voulaient gagner du temps pour le reste de la flotte, nous ne le saurons pas. En tout cas, aux alentours de 12h30, le combat commence et la *Louise* se retrouve assaillie par les canons de la *Mary Rose*. Démâtée, elle parvient quand même à se mettre à l'abri en rejoignant le goulet grâce notamment à la marée montante. Il ne reste alors pour la flotte franco-bretonne que la nef de Dieppe et la *Cordelière* à combattre. La nef de Dieppe, plus petite que la *Cordelière*, passera son temps à harceler les navires attaquant celle-ci.

1.2.3 Le combat de la Cordelière

Figure 1 – Peinture du dernier combat de la *Cordelière*.

La *Cordelière* affronte dans un premier temps la *Mary James* dans un combat de canons à distance, le navire anglais étant bien inférieur au navire breton, il ne peut

s'approcher sans risquer péril. Il gagne ainsi du temps, attendant l'arrivée des plus lourds navires que sont le *Sovereign* et le *Régent*, de tonnages équivalents à la *Cordelière*. Le *Sovereign* est le premier à attaquer la *Cordelière*, et disposant d'une artillerie moins importante, il décide de l'aborder pour profiter de ses troupes. Mais en approchant de la *Cordelière*, celle-ci parvient à le démâter, ce qui lui fait louper sa manœuvre et le met hors d'état de nuire. Le *Régent* abandonne alors son abordage de la nef de Dieppe pour attaquer la *Cordelière*, celle-ci étant la plus dangereuse.

Les deux navires se canonnent un certain temps, puis se rapprochent et s'abordent, le *Régent* à tribord de la *Cordelière*. Le combat fait rage, les carreaux d'arbalètes pleuvent sur ceux qui tentent de rejoindre l'autre navire et des incendies se déclarent sur la *Cordelière*. Hervé de Portzmoguer, sentant le péril qu'encourt la *Cordelière*, décide de faire mouiller une ancre. Cette action fait pivoter les deux navires grâce au courant, l'avant tourné vers le large. Le *Régent* se retrouve maintenant sous le vent, les flammes de la *Cordelière* s'emparent alors de lui, s'ajoutant au chaos ambiant. L'un des mâts de celui-ci s'effondre, créant d'autres brasiers sur les deux navires. Des combattants se jettent à l'eau pour échapper aux flammes, le combat est chaotique, quand soudainement une explosion envoie les deux navires à l'eau.

Figure 2 – Représentation du dernier combat de la Cordelière.

La violence de cette explosion stupéfie les navires alentours, qui mettent à l'eau des embarcations pour sauver ceux qui peuvent l'être. La nef de Dieppe ayant toujours à ses trousses des navires anglais, des barques Guérandaises viendront la soutenir le soir pour lui permettre d'échapper au combat et rentrer en sûreté. Concernant l'explosion, on ne sait toujours pas si elle était intentionnelle, mais on est sûr que c'est la sainte-barbe de la *Cordelière* qui a explosé. La légende raconte qu'Hervé de Portzmoguer lui-même aurait mis le feu à la sainte-barbe, le combat étant désespéré. Mais il est plus probable que la chaleur environnante et les nombreux foyers de flamme aient enflammé la sainte-barbe. Ce qui est sûr, c'est que cette explosion marquera notablement les esprits pour des générations, faisant entrer la *Cordelière* et le *Régent* dans la légende, amenant les recherches actuelles pour retrouver les épaves.

2 Architecture logicielle

Partie réalisée par Yacine BENHNINI, Joris TILLET et François CÉBRON

2.1 Les intérêts et objectifs de l'architecture logicielle

La recherche de l'épave de la *Cordelière* est un projet conséquent qui requiert une grande partie logicielle. De nombreuses lignes de codes devant être écrites, et plusieurs personnes collaborant sur le projet, il était nécessaire de structurer et répartir le travail pour que tout le monde puisse travailler sur une partie bien définie, et surtout en lien avec d'autres parties du projet.

Les différents objectifs de notre architecture logicielle sont résumés ci-dessous.

— Utiliser un Middleware [5] adapté à la robotique (ROS [6]). Ainsi, l'intégration des différentes parties du logiciel codées par différentes personnes, dans différents langages, était simplifiée : le Middleware permet justement cette communication entre les différents composants du logiciel, et fait fonctionner tous les blocs ensemble. L'utilisation de ce Middleware contraint l'utilisation des langages de programmation Python 2.7 et C++. Nous avons choisi de coder majoritairement en Python, pour des questions de simplicité et de temps, cependant nous avions prévu de migrer les portions de codes validées en C++, afin de réduire la consommation en ressources du code.

— Permettre la subdivision du travail au sein d'une équipe aux compétences variées. Pour cela nous avons choisi d'utiliser des paquets ROS [7] afin de segmenter les différentes parties du code selon une organisation hiérarchique. De plus nous avons choisi de définir différents messages types [8] (utilisés dans les topics ROS [9]), afin de simplifier l'interfaçage du travail des différentes équipes travaillant sur le projet.

— Avoir une approche modulaire entre la simulation et le système réel, afin d'éviter d'avoir deux implémentations différentes entre le simulateur et le logiciel définitif.

— Permettre la réutilisation du travail effectué, c'est aussi pour cela nous avons imposé de coder en anglais.

— Utiliser des méthodes de développement agiles, telles que le versionning de code (Git/Github), qui sera décrit plus tard, dans la section 2.4.

5. Logiciel qui crée un réseau d'échange d'informations entre différentes applications informatiques.

6. Robot Operating System : Middleware développé pour la robotique. Ce dernier intègre une architecture client-serveur pouvant être répartie sur plusieurs machines travaillant en réseau. Il permet la créations de programmes indépendants (les nœuds) interagissant entre eux via des canaux de communication (les topics) ou des requêtes (les services). Consultez [13] pour en savoir plus (en anglais).

7. Regroupement de différents nœuds ROS, afin de créer des distinctions entre différentes catégories de nœuds et de les installer de façon simplifiée.

8. Sous ROS, nous pouvons définir des structures de données afin de simplifier les échanges entre différents nœuds.

9. Canaux de communication entre différents nœuds ROS, permettant de contraindre le type de données échangées.

2.2 L'architecture logicielle

Le schéma suivant (voir Figure 3, page 16 - également disponible en Annexe page 62), fait la synthèse de toutes les exigences précédemment exprimées et permet l'interfaçage de façon robuste du travail des différentes équipes dans ce projet.

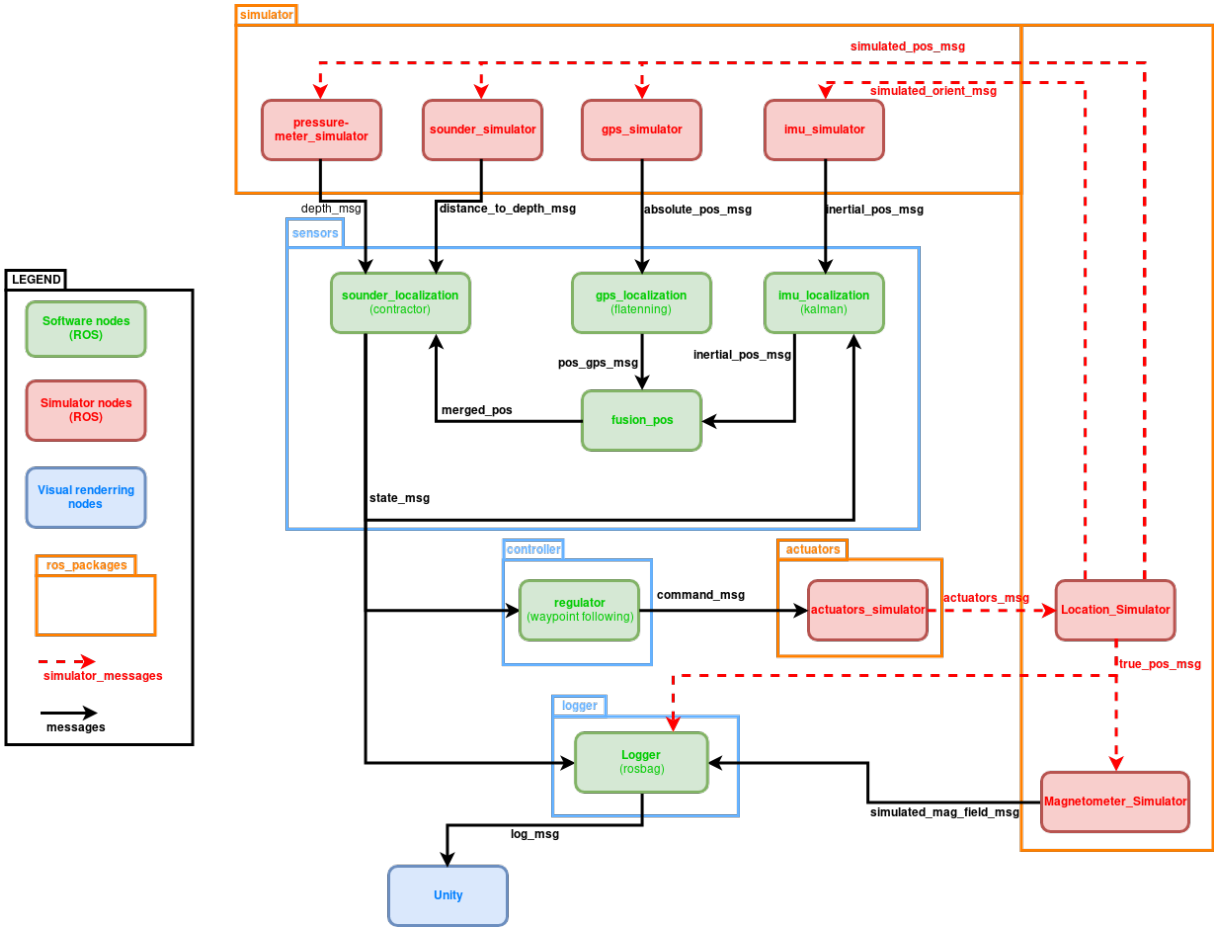


FIGURE 3 – Architecture logicielle retenue (disponible également en annexe page 62)

Afin de traduire le fait que le code doit servir à la fois au simulateur et au système définitif (dépendant des choix matériels), nous avons choisi de créer un packages relatifs à la base de code commune au simulateur et au système final (les paquets "Sensors", "Controller" et "Logger") et des paquets correspondant aux entrées/sorties simulées de cette base de code commune (les paquets "Simulator" et "Actuators"). Ces différents paquets seront décrits plus en détails dans les parties suivantes.

2.2.1 Le paquet "Simulator"

Cette section logicielle a pour but de fournir des données capteurs simulées pour alimenter les algorithmes du contrôleur. L'intérêt principal de cette approche est que l'on garde le même contrôleur dans la simulation et dans le système réel sans avoir à re-coder deux fois la même chose. Pour que ceci se déroule comme prévu, il faut cependant définir des normes quant à l'échange des données entre chaque bloc afin d'assurer leur interfaçage. C'est ce à quoi sert la définition de messages via ROS, on impose un format aux données puis il suffit de les remplir soit avec un nœud simulé, soit avec un nœud connecté à du matériel réel. Pour être réaliste, la simulation prend

en compte le bruit qui pourrait survenir dans un cas réel : bruit intrinsèque au capteur lors de la mesure, ou bruit lié à un phénomène physique. Pour des raisons de tests, les valeurs bruitées ne sont cependant pas transmises à la simulation de la détection d'objets via magnétomètre.

2.2.2 Le paquet "Sensors"

Cette partie sert à l'interfaçage entre les différents capteurs afin de se localiser. Elle contient les nœuds d'interface avec les capteurs suivants :

— Le GPS : sert à se positionner en surface de manière très précise, avant de perdre son signal (le GPS ne passe pas sous l'eau). Ce dernier peut également servir à repositionner le sous-marin dans un cas où il serait complètement perdu, il referait donc surface afin de récupérer une position GPS et pouvoir continuer sa mission.

— La centrale inertielle : elle est un organe essentiel à la localisation d'un sous-marin sous l'eau, car les ondes électromagnétiques étant bloquées par l'eau, on se trouve obligé de faire du *dead-reckoning* [10]. Dans notre cas, nous avons estimé pouvoir utiliser un filtrage de type *Kalman* sur ce capteur, via les informations du vecteur d'état de l'AUV.

— Un baromètre : le capteur de pression permet de connaître la position selon l'axe vertical du sous-marin, il permet d'exploiter le sondeur afin de localiser de façon fiable l'AUV.

— Un sondeur (mono-faisceau) permettant de mesurer la profondeur sous l'AUV. Seul, il peut permettre un asservissement bathymétrique (via un suivi d'isobathes), couplé au baromètre il permet à l'AUV de se repérer verticalement de manière assez précise.

Ces différentes données sont associées de la façon suivante :

1. Les données GPS sont lues lorsque l'AUV est en surface.
2. Ces données sont fusionnées avec la centrale inertielle via un filtrage de type *Kalman* et servent dans l'algorithme de *dead-reckoning*.
3. La position estimée ainsi que son erreur sont envoyées au nœud de localisation par intervalles qui va coupler ces information avec celles issues des deux capteurs donnant une information verticale et à la carte des fonds marins connue. On en tirera un vecteur d'état, qui sera renvoyé dans le nœud de *dead-reckoning*, au régulateur et au logger.

2.2.3 Le paquet "Controller"

Cette partie contient le nœud "Controller", qui est le contrôleur s'occupant de la trajectoire en 3 dimensions de l'AUV, suivant une régulation en cap ainsi qu'en profondeur. Ce nœud et son algorithme de régulation sont décrits plus loin dans la partie Régulateur. Ce nœud accepte en entrée l'équation d'état du robot (l'état actuel : x, y, z, roll (roulis), pitch (tangage), yaw (lacet) et leurs dérivées) et en ressort les consignes

10. Navigation à l'estime où l'on se base sur des capteurs donnant une information de localisation relative et donc cumulant des erreurs de position. En fonction du capteur, et de la robustesse de l'algorithme, la précision de cette méthode de localisation diminue plus ou moins vite.

moteurs selon quatre paramètres (throttle [11], roll [12], pitch [13] et yaw [14]).

2.2.4 Le paquet "Actuators"

Ce paquet est le moins abouti car il dépendra du type d'AUV qui sera utilisé et de la façon dont il est propulsé, s'il possède un seul propulseur et des gouvernes, comment ces dernières sont positionnées, si au contraire, il possède plusieurs propulseurs et pas d'ailettes... Ce paquet contient donc actuellement uniquement un nœud nommé "actuators_simulators". Ce dernier se contente de récupérer les commandes moteurs (throttle, roll, pitch, yaw) et de les faire suivre au reste du simulateur. À l'avenir, il faudra prévoir une transformation pour rendre ces consignes compatibles avec la configuration de l'AUV.

2.2.5 Le paquet "Logger"

Le logger nous sert à centraliser toutes les données en transit dans le système : positions estimées, temps de mission, données magnétométriques, etc... Cela peut servir à la fois pour avoir un retour temps réel du système pendant la mission, mais aussi de pouvoir re-visualiser la mission après le retour du robot. Dans le cas d'un sous-marin, il s'agira donc plutôt de revoir le comportement du robot après sa mission afin de déceler d'éventuels problèmes ou incohérences et d'effectuer les corrections avant la mise à l'eau suivante. C'est aussi la source principale d'informations servant au rendu 3D via le nœud Unity.

2.3 Limites de l'architecture retenue

L'architecture proposée donnait beaucoup d'avantages, mais impliquait également de résoudre quelques problèmes liés à nos choix. En effet, certaines interactions entre les librairies utilisées et notamment ROS nous ont posé deux complications.

1. Dans ce rapport vous sera présenté une partie sur la localisation de notre système, qui utilise une technique particulière appelée *analyse par intervalles*. Seulement, cette technique utilise une librairie, Ibex, qui n'est pas compatible avec ROS. Nous avons donc dû résoudre ce problème en implémentant nous-même la communication entre le travail sur la localisation, et le reste du projet (habituellement automatique avec ROS). Nous avons donc instauré une communication utilisant le protocole TCP pour que cette partie concernant la localisation ne soit pas isolée et puisse donc être utilisée par les autres parties du projet.
2. Le deuxième soucis sur lequel nous avons dû nous pencher est l'intégration du moteur de jeu Unity utilisé pour la représentation graphique. ROS n'est pas directement compatible avec ce moteur, et il a fallu également trouver une solution pour pouvoir avoir les jolis rendus souhaités. Ladite solution vous sera décrite plus loin dans ce rapport, dans la section qui concerne Unity.

11. La vitesse commandée en valeur algébrique.
12. Le roulis commandé, selon les angles d'Euler.
13. Le tangage commandé selon les angles d'Euler.
14. Le lacet commandé selon les angles d'Euler.

2.4 Le système de contrôle de version - Git/Github

Comme nous étions relativement nombreux à travailler sur ce projet, et que la partie logicielle est assez importante, il n'était pas pensable de travailler sans utiliser de système de contrôle de version (VCS : Version Control System). Nous avons utilisé le plus répandu aujourd'hui : *Git*. Ainsi, nous pouvions tous travailler en même temps, sur différentes parties du code ou non, nous pouvions facilement fusionner nos parties, et savoir ce qui avait été modifié, quand, par qui, et pourquoi.

De plus, la totalité du projet est hébergée sur un serveur distant : *GitHub*. De cette façon, tout le monde pouvait aller chercher les dernières mises à jour du projet, quelque soit la personne qui les ait faites et où qu'elles soient. La compatibilité des différentes mises à jour était également sans cesse vérifiée, et ainsi l'intégration de tous les morceaux de code s'est faite progressivement tout au long du projet, et sans problème majeur.

Finalement, pour encore plus de souplesse dans la gestion de ces différentes versions qui ont existé tout au long du projet, nous avons utilisé des *submodules*. Ainsi, certaines parties du projet - les plus lourdes notamment - sont hébergées en dehors du reste du projet, ce qui permet d'éviter que tout le monde télécharge des grosses mises à jour pas forcément souhaitées par tous. Par exemple, la partie sur Unity est hébergée ailleurs, via deux submodules différents [15]. Cependant, un lien est laissé dans le répertoire du projet pour rediriger automatiquement vers ces submodules, en rendant transparent le fait que tout le code ne soit pas au même endroit.

15. Un pour la partie Unity à proprement parler, et l'autre pour la librairie Python qui permet d'envoyer les commandes au moteur graphique.

3 Localisation

Partie réalisée par Clément ROLINAT et Nicolas VEYLON

3.1 Arbre représentant la carte bathymétrique

3.1.1 Problématique

Nous disposons d'une carte bathymétrique, c'est-à-dire d'une carte de profondeur du fond marin. Pour nous localiser convenablement, nous avons besoin d'une carte la plus précise possible. Par exemple, une résolution de quelques dizaines de centimètres. Compte tenu de la zone à couvrir qui s'étend sur plusieurs dizaines de kilomètres, une telle carte serait très volumineuse, et complexe à manipuler.

Pour faciliter l'utilisation de cette carte, nous avons dû la représenter sous une forme différente. Nous avons choisi une représentation sous forme d'arbre binaire. Cette structure permet d'accéder très rapidement à une zone donnée de la carte, par un parcours en profondeur.

3.1.2 Structure de l'arbre

Chaque nœud de l'arbre correspond à une zone donnée sur la carte, jusqu'aux feuilles qui correspondent chacune à un pixel. On créé l'arbre par bissections successives de la zone associée à chaque nœud. Ainsi les zones associées aux fils droit et gauche d'un nœud sont respectivement la moitié droite et la moitié gauche de la zone associée à ce nœud.

Chaque nœud stocke uniquement l'intervalle de profondeur associée, l'axe de bissection utilisé pour la création de ses enfants, et leurs adresses. La zone correspondant à un nœud n'est donc pas stockée directement, mais est retrouvée grâce à l'axe de bissection. Pour les feuilles, l'intervalle de profondeur associé correspond à un intervalle assurant la continuité de la profondeur entre pixels voisins. Pour les nœuds supérieurs, l'intervalle associé correspond à l'union des intervalles associés à leurs nœuds fils.

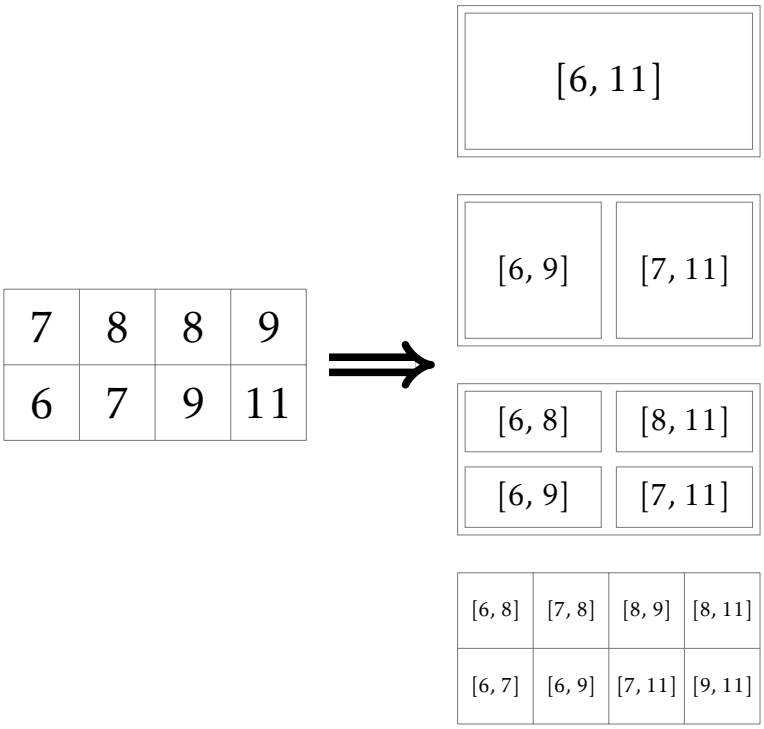


FIGURE 4 – Remplissage de l'arbre avec la carte bathymétrique. Ici la carte donnée à titre d'exemple est de résolution 2×4.

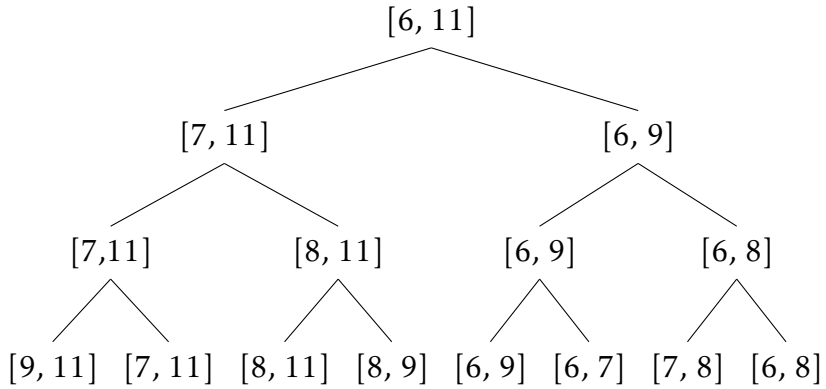


FIGURE 5 – Représentation de l'arbre.

3.2 Contracteur

3.2.1 Algorithme de contraction

Voici l'algorithme de contraction développé [16].

Algorithme 1 : contraction

Entrée : $[\mathbf{b_{in}}] = [\mathbf{b_{pos}}] \times [z_{mes}]$: boite à contracter $[estimation] \times [mesure]$
Sortie : $[\mathbf{b_{out}}]$: boite contracté $C([\mathbf{b_{in}}])$

1 **début**
2 $S \longleftarrow \{\}$
3 $root \longleftarrow$ racine de l'arbre
4 $[\mathbf{b_{root}}] \longleftarrow$ boite liée à $root$
5 `parcoursEnProfondeur`$([\mathbf{b_{in}}], [\mathbf{b_{root}}], root, S)$
6 $[\mathbf{b_{out}}] \longleftarrow \bigsqcup_{[\mathbf{b}] \in S} [\mathbf{b}]$
7 $[\mathbf{b_{out}}] \longleftarrow [\mathbf{b_{out}}] \sqcap [\mathbf{b_{in}}]$

Algorithme 2 : parcoursEnProfondeur

Entrées : $[\mathbf{b_{in}}] = [\mathbf{b_{pos}}] \times [z_{mes}]$: boite à contracter
$[\mathbf{b_{node}}]$: boite liée à $node$
$node$: noeud de l'arbre
S : ensemble de boites validées

1 **début**
2 **si** $[\mathbf{b_{node}}] \sqcap [\mathbf{b_{pos}}] \neq \emptyset$ **alors**
3 $[z] \longleftarrow$ intervalle de valeur lié au noeud (et donc à $[\mathbf{b_{node}}]$)
4 **si** $[z] \subseteq [z_{mes}]$ **alors**
5 ajout de $[\mathbf{b_{node}}] \times [z]$ dans S
6 **sinon si** $[z] \sqcap [z_{mes}] \neq \emptyset$ **alors**
7 **si** $node$ est une feuille **alors**
8 ajout de $[\mathbf{b_{node}}] \times [z]$ dans S
9 **sinon**
10 $[\mathbf{b_{left}}], [\mathbf{b_{right}}] \longleftarrow$ résultat de la bissection de $[\mathbf{b_{node}}]$ suivant $node.axis$
11 `parcoursEnProfondeur`$([\mathbf{b_{in}}], [\mathbf{b_{left}}], node.left, S)$
12 `parcoursEnProfondeur`$([\mathbf{b_{in}}], [\mathbf{b_{right}}], node.right, S)$

Le principe général est de descendre l'arbre en suivant seulement les branches dont leur domaine en valeur $[z]$ intersecte la mesure $[z_{mes}]$, c'est-à-dire dans les cas a, b et c de la figure 6, page 23. Il faut également que leur domaine de position $[\mathbf{b_{node}}]$ intersecte la position estimée $[\mathbf{b_{pos}}]$.

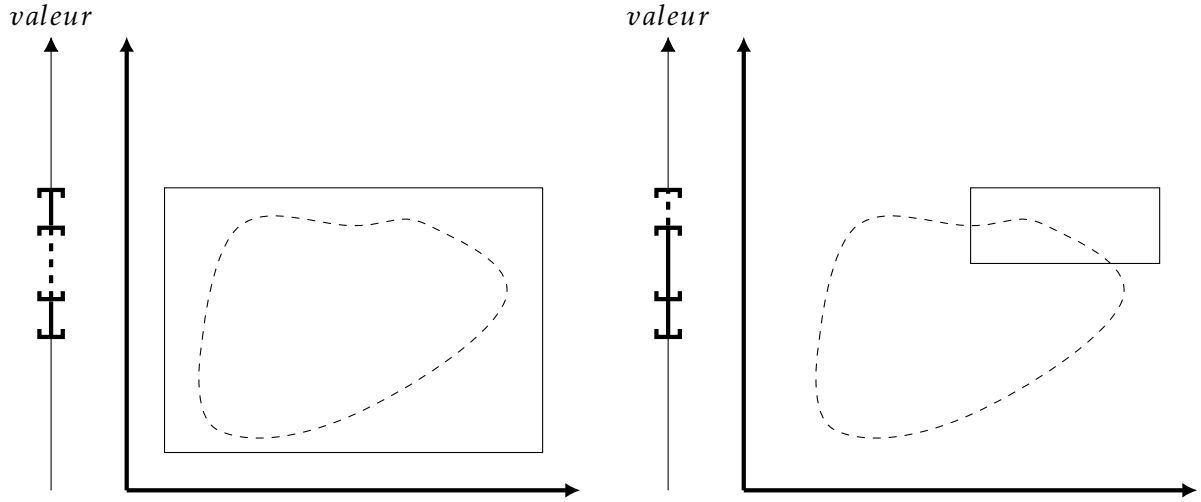
— Si le domaine de valeur du nœud est contenu dans l'intervalle de mesure (axe *valeur* de la figure c), l'intégralité du domaine de position du nœud est compatible avec la mesure, donc on l'ajoute dans la liste S.
— Si le domaine de valeur contient l'intervalle de mesure, ou qu'il l'intersecte

16. Il est important de noter qu'un contracteur est un objet purement mathématique, il ne présente donc aucun trait de notre problématique de localisation sous-marine.

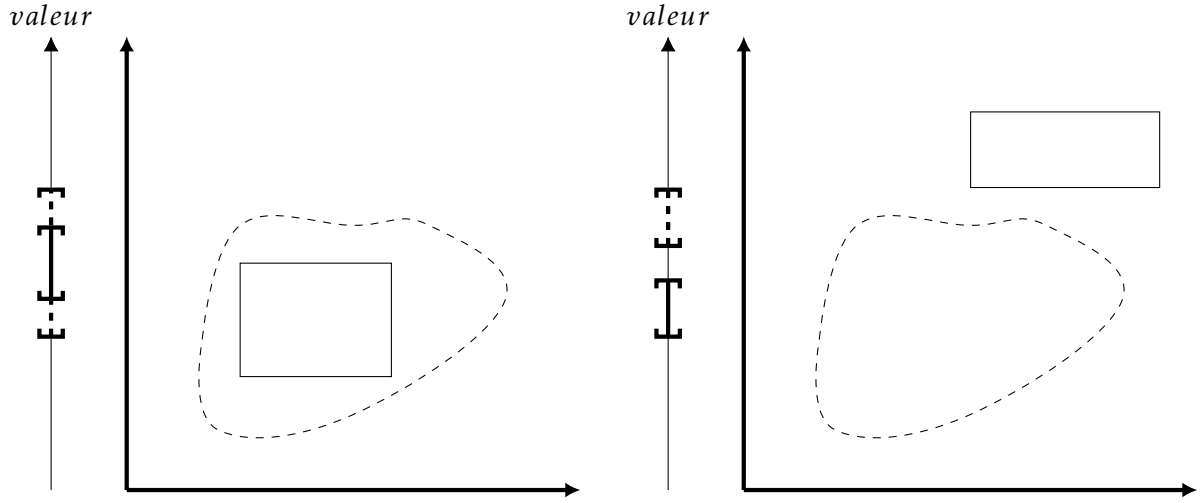
seulement, (axe *valeur* des figures a et b), cela veut dire que la boite contient des positions compatibles avec la mesure, il faut donc parcourir la branche. Si le nœud est une feuille, on ajoute la position lié à la feuille dans la liste S car cette position est compatible avec la mesure.

Après cette descente de l'arbre, la liste S contient des boites de position de la carte telles que leur union donne une boite $[\mathbf{b_{out}}]$ compatible avec la mesure. L'intersection de l'estimation de position $[\mathbf{b_{pos}}]$ et de la mesure de position $[\mathbf{b_{out}}]$ donne la boite contractée, meilleure estimation de la position (voir figure 7, page 24).

Il est possible grâce à la structure en arbre de s'arrêter à une certaine précision, correspondant donc directement à une profondeur maximale dans l'arbre. Cela peut être utile pour obtenir rapidement une estimation de position.



(a) boite contenant l'ensemble solution (b) boite intersectant l'ensemble solution



(c) boite incluse dans l'ensemble solution (d) boite n'intersectant pas l'ensemble solution

FIGURE 6 – Les quatre cas rencontrés lors du parcours de l'arbre. La boite représentée est celle lié au nœud courant. Cet exemple est en 2D mais il généralisable à un problème à plus de dimension.

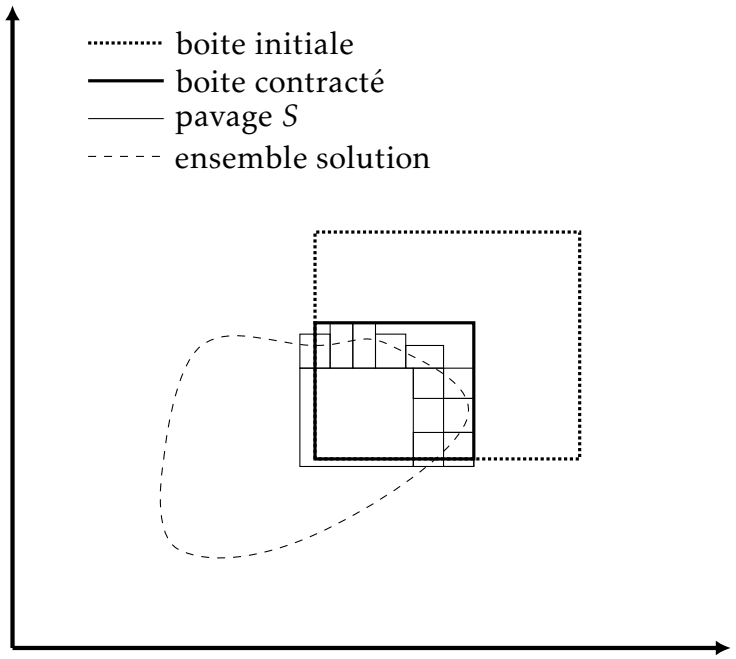


FIGURE 7 – Résultat de la contraction d'une boite

3.2.2 Application du contracteur au problème de localisation sous-marine

Dans notre problème de localisation sous-marine, l'arbre binaire de recherche contient les informations de la carte bathymétrique. Les données sont donc 2D (une valeur de profondeur z pour chaque pixel x-y de la carte).

Ce contracteur sera utilisé dans l'estimateur d'état. Soit $[\mathbf{x_{k|k}}]$ l'estimation de la position x-y-z du robot à l'instant k. A l'instant $k+1$, le robot estime d'abord sa position $[\mathbf{x_{k+1|k}}]$ grâce aux mesures de la boussole et de la centrale inertielle. Cette estimation est faite grâce à l'équation d'état du robot :

$$[\mathbf{x_{k+1|k}}] = [\mathbf{x_{k|k}}] + dt \cdot f([\mathbf{x_{k|k}}]) \tag{1}$$

Ensuite vient la mesure de la profondeur de la mer $[z_{mes}]$, obtenu à partir de la mesure du baromètre et du sondeur. Cette mesure permet d'obtenir l'estimation $[\mathbf{x_{k+1|k+1}}]$ en contractant $[\mathbf{x_{k+1|k}}]$:

$$[\mathbf{x_{k+1|k+1}}] = C_{[z_{mes}]}([\mathbf{x_{k+1|k}}]) \tag{2}$$

avec $C_{[z_{mes}]}$ le contracteur associé à la mesure $[z_{mes}]$.

4 Régulation

Partie réalisée par Fabrice LALLEMENT

De manière générale, la régulation en robotique se définit comme la minimisation d'une erreur. À partir de cette erreur, plusieurs méthodes de minimisation existent ensuite suivant le type du problème (linéaire/non-linéaire, gaussien, etc).

4.1 Régulation en profondeur

4.1.1 Intérêt de la régulation en profondeur

Les AUV évoluant, par définition, en milieu sous-marin, il est important de s'intéresser à la problématique de la régulation en profondeur principalement pour deux objectifs :

— la sauvegarde du robot,
— le succès de la mission en cours.

La sauvegarde du robot peut en effet être compromise par une profondeur trop importante du robot ou encore une collision avec le fond marin. Chaque AUV est en effet défini par des caractéristiques d'utilisation et notamment une profondeur maximale d'utilisation.

Pour ce qui est du respect de la mission, le respect d'une distance au fond est importante. En effet, il faut tenter de minimiser cette distance pour maximiser la précision de détection du magnétomètre. La mesure de magnétisme évoluant suivant l'inverse du cube de la distance. L'atténuation des mesures peut vite devenir énorme.

4.1.2 Calcul de l'erreur

L'erreur dans ce problème correspond à la distance verticale entre la profondeur actuelle du robot et la profondeur désirée. Il est important de noter ici que l'on considère que l'on dispose déjà en entrée de ces données et le calcul de l'erreur ne traitera donc pas de cette détermination.

Le schéma suivant (Figure 8, page 25) permet de comprendre simplement la méthode de détermination de cette erreur.

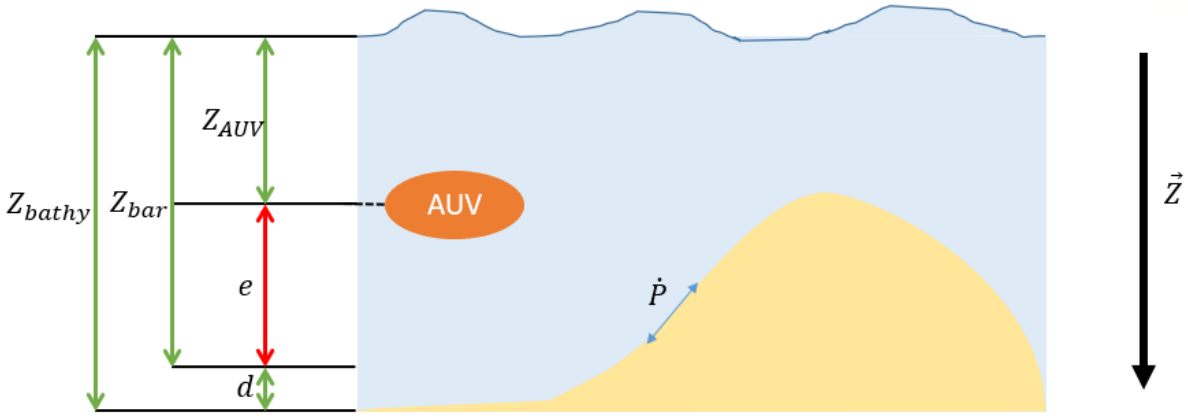


FIGURE 8 – Schéma explicatif du calcul de l'erreur

Les notations sont les suivantes :

— Z_{bathy} correspond à la profondeur du fond marin
— $\dot{P}$ correspond à la dérivée directionnelle du fond marin suivant le cap du robot
— Z_{AUV} correspond à la profondeur actuelle de l'AUV
— Z_{bar} correspond à la profondeur désirée
— d correspond à la distance souhaitée au fond
— e correspond finalement à l'erreur que l'on cherche à minimiser

À partir de ces notations, on peut donc définir l'erreur de la façon suivante :

$$Z_{bar} = Z_{bathy} - d$$
$$e = Z_{bar} - Z_{AUV}$$

On dispose donc maintenant d'une erreur qu'il s'agit de minimiser. Partant de cette erreur, il faut maintenant déterminer quel est notre moyen d'agir dessus. Dans les AUVs, deux types d'action sur cette donnée sont principalement possibles :

— une poussée vectorielle suivant l'axe $\vec{Z}$
— une variation de l'angle de plongée correspondant à l'angle de tangage θ

Dans le cadre de notre projet, nous avons considéré que notre régulation s'effectuerait par l'action sur l'angle de plongée. Bien que plus compliquée à implémenter, elle a l'avantage de ne pas nécessiter de disposer de moteurs dirigés suivant l'axe $\vec{Z}$. Deux implémentations de régulation de cette erreur ont été réalisées. Ces deux méthodes correspondent à deux calculs différents de θ_{bar}. Cet angle désigne l'angle de plongée désiré. La différence entre ces deux calculs d'angle réside dans la prise en compte de la dérivée du fond marin. Sans cette prise en compte, le calcul est plus simple car il n'y a pas besoin de calculer cette dérivée qui peut être compliquée à obtenir. Cependant, des variations rapides de profondeur pourraient alors mettre en danger la sauvegarde du robot car celui-ci n'anticiperait pas ce changement et risquerait de rentrer en collision avec le fond.

$$\theta_{bar} = \tanh(e) \tag{3}$$
$$\theta_{bar} = \tanh(e) - \arctan(\dot{P}) \tag{4}$$

L'addition du terme correctif en $\arctan(\dot{P})$ permet donc de suivre l'évolution de la pente de la trajectoire. L'arc-tangente est, qui plus est, prédominant sur la tangente ce qui assure que la sauvegarde est plus importante que le respect de la mission. On pourrait également penser à prendre en compte $\ddot{P}$ pour être robuste à des changements rapides de pente.

4.1.3 Fonction minimisante

Connaissant l'angle de plongée désiré, il s'agit maintenant de faire converger le tangage actuel vers la valeur voulue. Le principe est donc maintenant d'appliquer une fonction minimisante à la différence des deux angles. Dans notre cas, nous avons pris la fonction sinus. En effet, notre angle de tangage étant forcément compris dans l'intervalle $[\frac{-\pi}{2}, \frac{\pi}{2}]$ où la fonction sinus est strictement croissante. On obtient donc la commande avec :

$$u_{pitch} = \sin(\theta_{bar} - \theta_{AUV}) \tag{5}$$

4.2 Régulation en cap

De manière évidente, dans l'optique de réaliser des missions avec le robot, celui-ci doit être à même de se déplacer dans le plan. Dans cette partie, on se focalisera sur le suivi de lignes. En effet, à partir de cette régulation, quasiment tous les suivis de trajectoire sont facilement réalisables.

4.2.1 Calcul de l'erreur

On se place dans le contexte présenté dans la Figure 9, page 27.

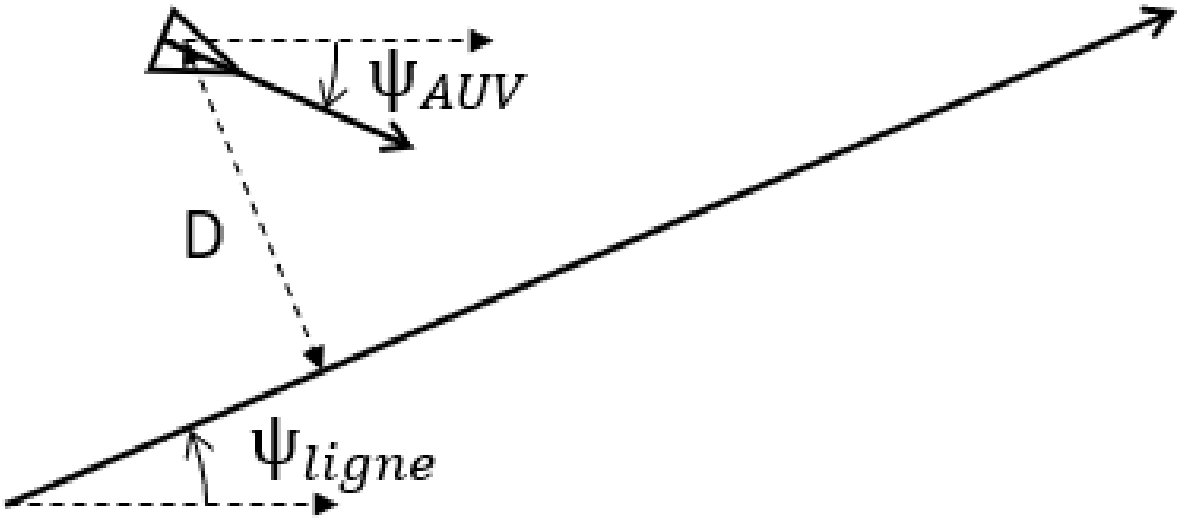


Figure 9 – Contexte du suivi de lignes

Les notations sont les suivantes :
— ψ_{AUV} correspond au cap actuel du robot
— ψ_{ligne} correspond au cap de la ligne
— $\overrightarrow{\psi_{AUV}}$ correspond au vecteur unitaire représentant le cap du robot
— $\overrightarrow{\psi_{ligne}}$ correspond au vecteur unitaire représentant le cap de la ligne
— D correspond à la distance à la ligne

Le principe est donc de faire converger le cap de l'AUV vers celui de la ligne tout en minimisant sa distance à la ligne. Il faut donc veiller à respecter un certain ordre de priorité. Il faut que la distance à la ligne prime lorsque cette distance est grande et que ce soit le cap lorsque cette distance est faible. Le calcul de l'erreur s'effectue donc comme tel :

$$D = \left| \overrightarrow{\psi_{AUV}} \quad \overrightarrow{\psi_{ligne}} \right|$$

$$\psi_{bar} = \psi_{ligne} + k * \tanh(D)$$

$$e = sawtooth(\psi_{bar} - \psi_{AUV})$$

Le terme en tangente hyperbolique de la distance permet de réaliser une correction du cap désiré pour permettre de se rapprocher effectivement de la ligne désirée. Le facteur k permet de régler le degré de correction maximal. Ainsi la valeur de k représente l'angle en radians maximal qui sera soustrait à l'angle désiré. De ce fait, k doit être dans l'intervalle $[0, \frac{\pi}{2}]$.

4.2.2 Régulateur PID

L'un des régulateurs les plus courants est le régulateur PID. Celui-ci se base sur trois gains différents (voir Figure 10, page 28) :

— le terme proportionnel qui diminue le temps de montée
— le terme intégral qui réalise une sommation de l'erreur pour contrer une erreur statique
— le terme dérivée qui vise à améliorer la stabilité

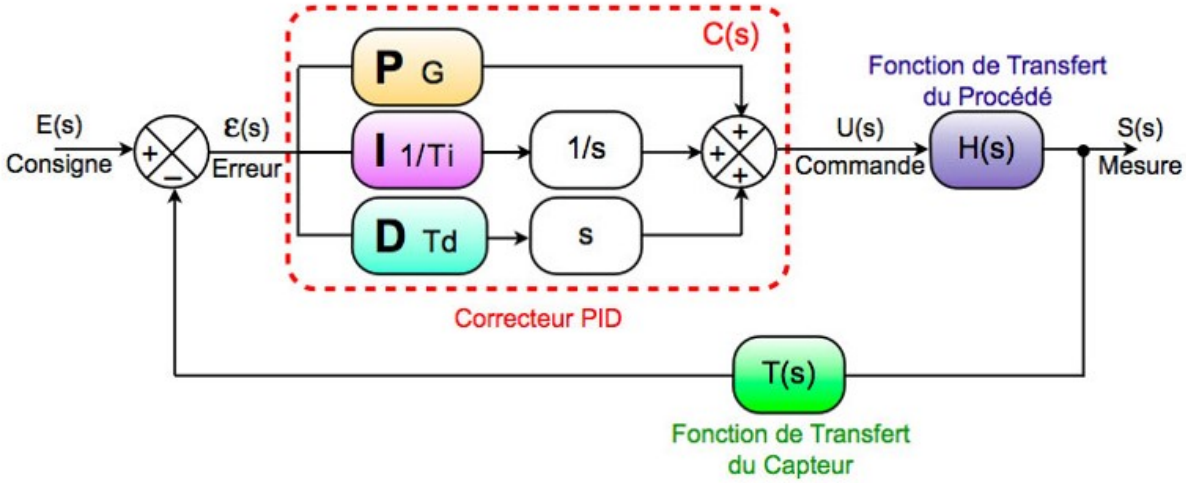


Figure 10 – Schéma d'un régulateur PID

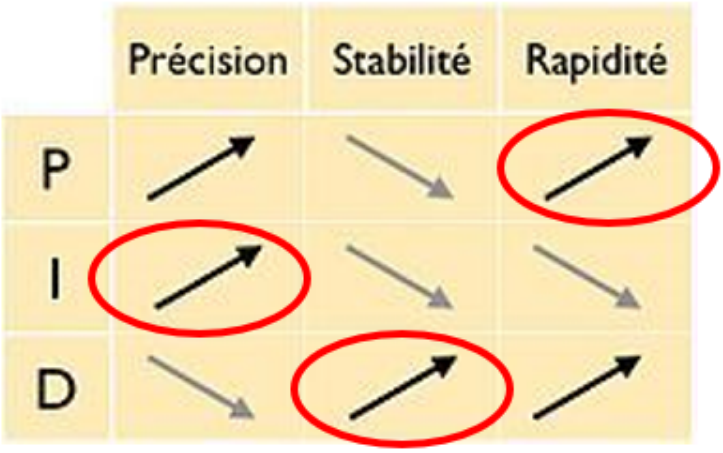


Figure 11 – Influence des coefficients PID

Le régulateur PID prend donc en entrée une erreur et en ressort une commande. Le problème réside dans le réglage des coefficients pour permettre un compromis optimal entre rapidité, stabilité et précision (voir Figure 11, page 28).

5 Contrôleur par Machine Learning

Partie réalisée par Mohamed OUTAHAR.

5.1 Motivations

Les robots utilisent des régulateurs pour bien accomplir leurs missions, et ces régulateurs essayent d'annuler un signal d'erreur en faisant tendre la sortie du système vers une consigne généralement définie par la phase de navigation.

Il existe plusieurs types de régulateurs, les PID sont les plus connus et les plus utilisés. Sauf que les régulateurs PID ne peuvent pas tout régler. C'est pour cela que d'autres méthodes de régulation ont été développées comme la méthode H_{infini} ou le bouclage linéarisant, mais même dans certains cas ces méthodes ne suffisent pas pour avoir de bonnes performances en régulation.

$$M_{ma} + \begin{pmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_i & 0 & 0 \\ 0 & 0 & 0 & 0 & I_j & 0 \\ 0 & 0 & 0 & 0 & 0 & I_k \end{pmatrix} \begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} [T_i \cos(\psi_{t_i}) \cos(\theta_{t_i})] - (M - \rho v) g \sin(\theta) - \frac{1}{2} \rho \frac{lw+wh+lh}{3} C_D \sqrt{u^2+v^2+w^2} \cos(\beta) \cos(\alpha) \\ +m(rv - qw) \\ \sum_{i=1}^{n} [T_i \sin(\psi_{t_i}) \cos(\theta_{t_i})] + (M - \rho v) g \cos(\theta) \sin(\phi) + \frac{1}{2} \rho \frac{lw+wh+lh}{3} C_D \sqrt{u^2+v^2+w^2} \sin(\beta) \\ +m(pw - ru) \\ -\sum_{i=1}^{n} [T_i \sin(\theta_{t_i})] + (M - \rho v) g \cos(\theta) \cos(\phi) - \frac{1}{2} \rho \frac{lw+wh+lh}{3} C_D \sqrt{u^2+v^2+w^2} \cos(\beta) \sin(\alpha) \\ +m(qu - pv) \\ -\sum_{i=1}^{n} \left[T_i \left(y_{t_i} \sin(\theta_{t_i}) + z_{t_i} \sin(\psi_{t_i}) \cos(\theta_{t_i}) \right) \right] - \rho v g \left(y_B \cos(\theta) \cos(\phi) - z_B \cos(\theta) \sin(\phi) \right) \\ -\frac{1}{2} \rho \frac{C_F}{4} (w^2 + h^2)^{\frac{3}{2}} (w + h) l |p| p + (I_j - I_k) qr \\ \sum_{i=1}^{n} \left[T_i \left(z_{t_i} \cos(\psi_{t_i}) \cos(\theta_{t_i}) + x_{t_i} \sin(\theta_{t_i}) \right) \right] + \rho v g \left(z_B \sin(\theta) + x_B \cos(\theta) \sin(\phi) \right) \\ -\frac{1}{2} \rho \frac{C_F}{4} (l^2 + h^2)^{\frac{3}{2}} (l + h) w |q| q + (I_k - I_i) rp \\ \sum_{i=1}^{n} \left[T_i \left(x_{t_i} \sin(\psi_{t_i}) \cos(\theta_{t_i}) - y_{t_i} \cos(\psi_{t_i}) \cos(\theta_{t_i}) \right) \right] - \rho v g \left(x_B \cos(\theta) \sin(\phi) + y_B \sin(\theta) \right) \\ -\frac{1}{2} \rho \frac{C_F}{4} (l^2 + w^2)^{\frac{3}{2}} (l + w) h |r| r + (I_i - I_j) pq \end{pmatrix}$$

FIGURE 12 – Modèle simplifié d'un AUV

Le contrôle des robots sous-marins présente une grande difficulté, à cause des mesures bruitées, des modèles de robots simplifiés et un environnement non maîtrisé. Dans notre projet, les robots sous-marins de recherche ont besoin de contrôleurs performants et robustes en même temps, c'est à dire un contrôleur qui peut garantir une réponse rapide, précise et stable tout en gardant ses performances face aux perturbations.

Nous avons opté, en plus d'un contrôleur PID, de concevoir un contrôleur à base de Machine Learning. Comme montré dans les documents [14], [15] et [16] les contrôleurs à base de Machine Learning ont généralement de meilleures performances que ceux des contrôleurs PID. L'exemple de [16] est le plus convaincant, car les auteurs utilisent un contrôleur "intelligent" afin de contrôler un hélicoptère miniaturisé, à l'envers, ce qui extrêmement complexe.

5.2 Machine Learning

Le Machine Learning est un domaine de l'informatique qui donne aux ordinateurs la capacité d'apprendre des modèles complexes à partir des données en utilisant des lois probabilistes et d'autres concepts mathématiques. C'est dans les années 50 que les premiers algorithmes sont sortis. Mais ce n'est qu'en 2006 que ces algorithmes ont commencé à être implémentés dans des applications commerciales, grâce aux fortes avancées concernant les performances des ordinateurs (suivant la loi de Moore) et la quantité immense de données collectées lors de l'utilisation d'Internet.

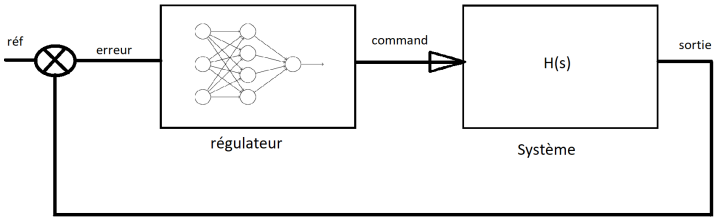


FIGURE 13 – Schéma du contrôleur à base de Machine Learning

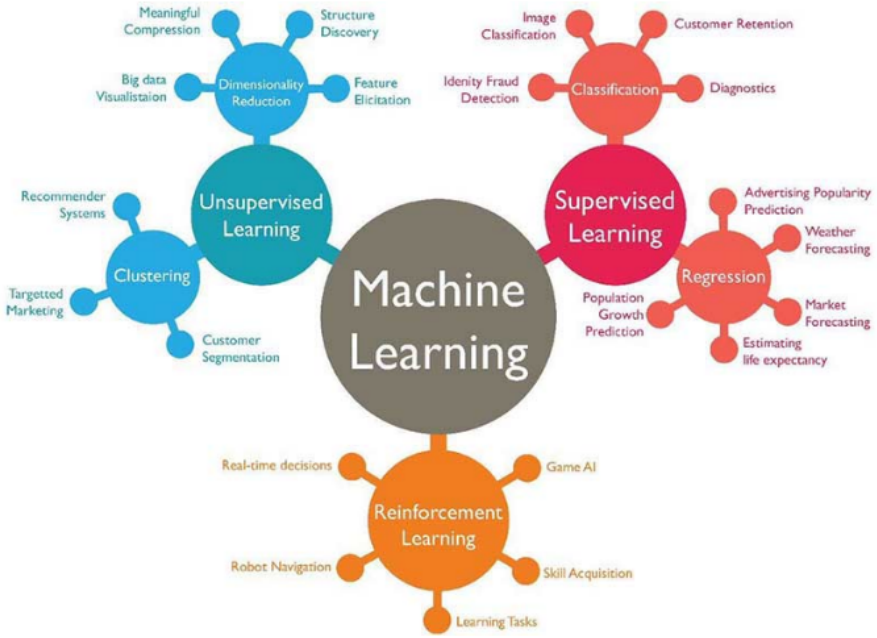


FIGURE 14 – Schéma des différents types d'algorithmes en Machine Learning

Plusieurs types d'algorithmes de Machine Learning existent :

— **L'apprentissage supervisé :** ce type désigne les algorithmes qui utilisent des données pré-traitées de façon à extraire les informations utiles que nous cherchons et apprendre le modèle qui relie ces informations aux données, comme les algorithmes de classification des images utilisé dans les voitures autonomes, le plus grand atout de ces algorithmes étant leurs temps d'exécution.

— **L'apprentissage non supervisé :** ce type désigne les algorithmes que nous appliquons quand les données ne sont pas maîtrisées, ces algorithmes doivent donc extraire le maximum d'informations possibles des données comme dans le cas du Clustering. Il est utilisé dans les entreprises de Marketing afin de subdiviser les clients en groupes homogènes selon leurs habitudes d'achat.

— **L'apprentissage par renforcement :** ce type désigne les algorithmes qui sont un mélange des deux autres types, une partie de ces algorithmes utilisent les données et les informations utiles, une deuxième phase utilise des données provenant de l'environnement. Ce type est le plus utilisé dans le contrôle des robots en milieux complexes, comme l'exemple de hélicoptère mentionné avant, de plus la majorité des grandes avancements dans le domaine du Machine Lear-

ning est lié à l'apprentissage par renforcement.

5.3 Implémentation

5.3.1 Algorithme proposé

Les robots sous-marins doivent utiliser un contrôleur robuste et performant, car dans le cas d'un problème mécanique ou électronique, le robot doit accomplir sa mission. En analysant la problématique de notre projet, la solution la plus optimale que nous avons trouvé est d'appliquer l'apprentissage par renforcement afin de contrôler le robot. Le plus grand atout de ce type d'algorithmes est qu'il continue à apprendre après implémentation, donc même si un problème change le comportement du robot, le contrôleur peut s'adapter et fournir les commandes les plus optimales.

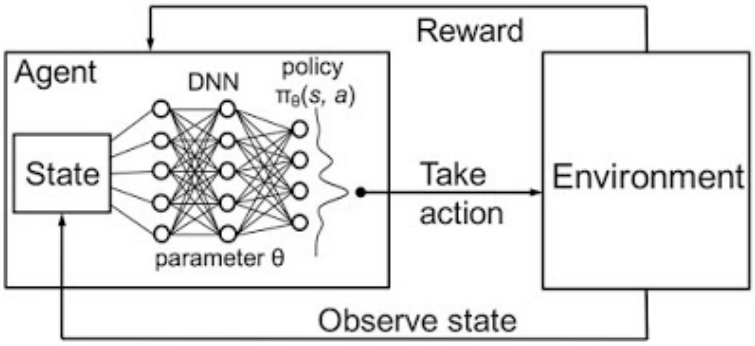


Figure 15 – Schéma des algorithmes d'apprentissage par renforcement

L'algorithme retenu est un réseau de neurones appliqué sur une action agissant sur l'environnement, cette dernière renvoie une équation d'état et une fonction de gain, que le système essaye de maximiser via le réseau de neurones, les poids et les biais du réseau sont mis à jour en fonction de ces deux facteurs. De plus la politique de l'algorithme est aussi changée. La politique est simplement la question entre si l'algorithme doit explorer des nouvelles actions ou utiliser des actions qu'il a déjà appris.

5.3.2 Réseaux de neurones

Les réseaux de neurones sont des systèmes informatiques (architecture informatique) qui sont utilisés pour modéliser des phénomènes fortement non linéaires. Ils contiennent des couches de neurones avec différentes dimensions, le nombre de couches et le nombre de neurones dans chaque couche dépendant du problème et de l'implémentation. Il n'existe pas une relation directe entre le nombre de neurones ou de couches et les performances du réseau. Les arêtes dans le schéma du réseau représentent des poids, devant chaque neurone il y a des biais (sauf les neurones d'entrée), le neurone représente l'application d'une fonction d'activation.

Les réseaux de neurones sont largement basé sur le fonctionnement d'un cerveau humain simplifié car leurs architectures ont étés développées durant les années 50, avec la participation de biologistes, mathématiciens et informaticiens pour définir ces algorithmes et les faire évoluer. Cette architecture est depuis utilisée dans l'apprentissage supervisé et l'apprentissage par renforcement.

Pour bien comprendre un réseau de neurones nous allons présenter une explication en utilisant un réseau de neurones avec 4 couches dont chaque couche est un neurone

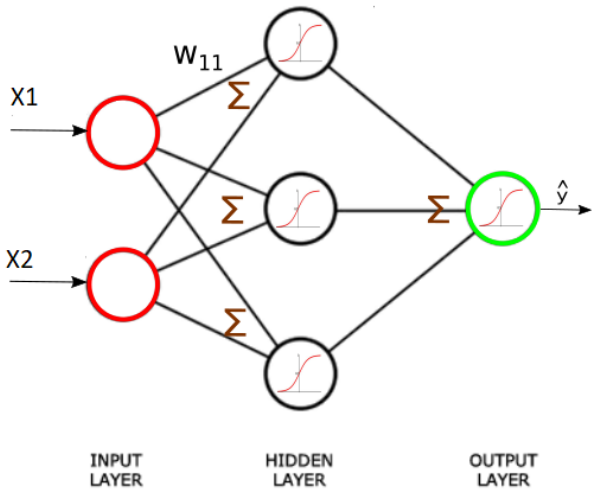


FIGURE 16 – Réseau de neurones

sans biais [17]. La fonction d'activation utilisée est la fonction Sigmoïde [17] parce qu'elle est simple à dériver.

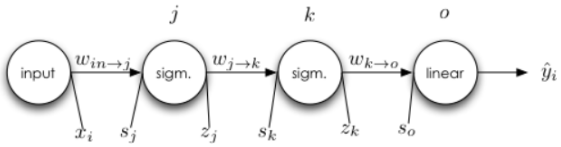


FIGURE 17 – Réseau de neurones simple

L'écriture de la sortie et de son erreur en fonction des entrées, des poids et les fonctions d'activation, est disponible en Figure 18, page 32. Cette formalisation va nous aider à simplifier les calculs de dérivées pour les gradients.

$$
\begin{aligned}
s_j &= w_1 \cdot x_i \\
z_j &= \sigma(in_j) = \sigma(w_1 \cdot x_i) \\
s_k &= w_2 \cdot z_j \\
z_k &= \sigma(in_k) = \sigma(w_2 \cdot \sigma(w_1 \cdot x_i)) \\
s_o &= w_3 \cdot z_k \\
\hat{y}_i &= in_o = w_3 \cdot \sigma(w_2 \cdot \sigma(w_1 \cdot x_i)) \\
E &= \frac{1}{2}(\hat{y}_i - y_i)^2 = \frac{1}{2}(w_3 \cdot \sigma(w_2 \cdot \sigma(w_1 \cdot x_i)) - y_i)^2
\end{aligned}
$$

FIGURE 18 – Modélisation du réseau de neurones

La dérivation de l'erreur dépend des poids de la dernière couche, afin de calculer le gradient et les mettre à jour. Les autres couches et leurs poids sont mis à jour de la même façon, ainsi que les biais, nous dérivons l'erreur quadratique multipliée par $\frac{1}{2}$ pour simplifier les formules et vu que nous multiplions par un facteur d'apprentissage, ce coefficient ($\frac{1}{2}$) n'a pas d'effets sur les résultats. Pour chaque poids dérivé il existe une partie qui se répète dans les couches d'avant. Les dérivés des fonctions d'activation figurent aussi dans les calculs des gradients des poids et des biais.

17. $f(x) = \frac{1}{1+e^{-\lambda x}}$

$$
\begin{aligned}
s_j &= w_1 \cdot x_i \\
z_j &= \sigma(in_j) = \sigma(w_1 \cdot x_i) \\
s_k &= w_2 \cdot z_j \\
z_k &= \sigma(in_k) = \sigma(w_2 \cdot \sigma(w_1 \cdot x_i)) \\
s_o &= w_3 \cdot z_k \\
\hat{y}_i &= in_o = w_3 \cdot \sigma(w_2 \cdot \sigma(w_1 \cdot x_i)) \\
E &= \frac{1}{2}(\hat{y}_i - y_i)^2 = \frac{1}{2}(w_3 \cdot \sigma(w_2 \cdot \sigma(w_1 \cdot x_i)) - y_i)^2
\end{aligned}
$$

FIGURE 19 – Dérivation des poids de la dernière couche

5.3.3 Résultat

Le contrôleur créé pour ce projet est une version simplifiée du contrôleur étudié dans la partie précédente, il comporte un réseau de neurones avec trois couches, la première possède deux neurones d'entrées, la couche intermédiaire à 10 neurones et la sortie un seul.

La première phase consiste à entraîner le réseau de neurones sur les données provenant du contrôleur. La condition d'arrêt sur chaque donnée est d'avoir une erreur inférieure à 0.005 (voir Figure 20, page 33).

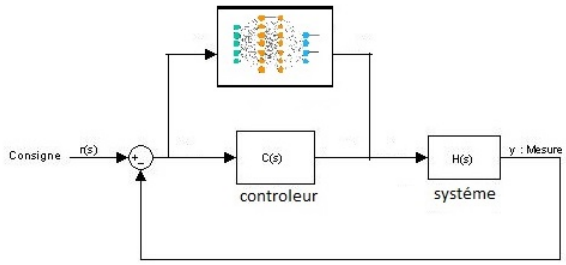


FIGURE 20 – Première phase de l'apprentissage

Après cette phase nous remplaçons le contrôleur PI avec le nouveau réseau entraîné (voir Figure 21, page 33).

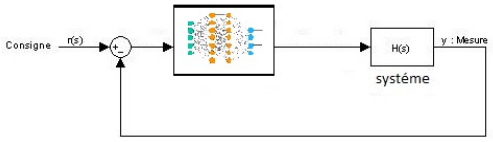


FIGURE 21 – Deuxième phase de l'apprentissage

Les résultats obtenus par la régulation en cap du réseau de neurones sont visibles en Figure 22, page 34.

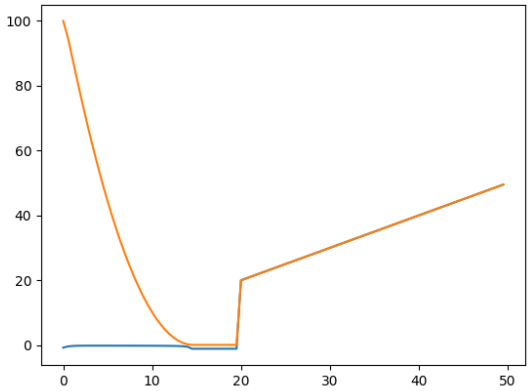


FIGURE 22 – Graphique présentant l'erreur du réseau de neurones

6 Planification d'une trajectoire

Partie réalisée par Sophie TUTON et Maha ABOUZAID

6.1 Objectif

La planification de trajectoire est une solution possible pouvant se substituer à la localisation (notamment dans le cas d'une perte de position), ce qui est non négligeable, car se localiser sous l'eau n'est pas évident.

L'objectif est de trouver une trajectoire fiable, c'est-à-dire une trajectoire pour laquelle on retrouvera le robot toujours au même endroit à partir du moment où il part à peu près de la même position de départ.

6.2 Hypothèses

Le robot sous-marin est supposé à profondeur constante et mesure son altitude par rapport au fond marin. Ainsi, il connaît la carte bathymétrique du fond marin à chaque instant.

Le principe est le suivant : le robot est mis à l'eau à une position donnée (dans le cercle noir de la Figure 23, page 35), il descend "verticalement" jusqu'à une profondeur donnée. Puis il doit avancer dans une certaine direction jusqu'à trouver l'isobathe voulue. Ensuite, il doit suivre l'isobathe (flèche noire de la Figure 23) et il doit remonter lorsqu'il est dans une certaine direction (dans le cercle blanc de la Figure 23 s'il y a fiabilité).

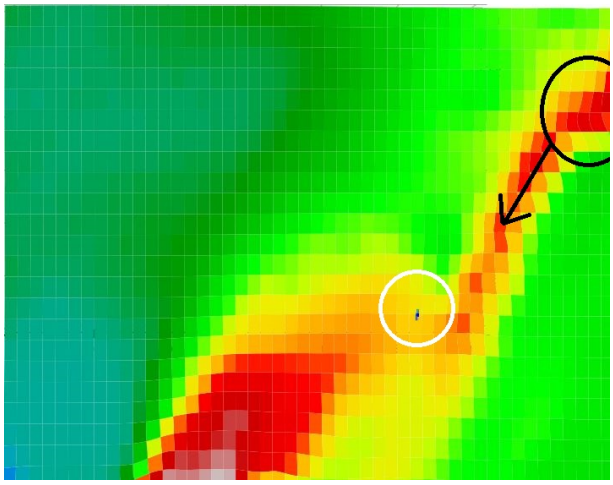


FIGURE 23 – Schéma explicatif de la trajectoire fiable à partir d'une carte bathymétrique

6.3 Méthodes utilisées

6.3.1 La carte bathymétrique

Dans la réalité de ce contexte, l'AUV doit mesurer son altitude, c'est pourquoi il est nécessaire de simuler des données de cartes bathymétriques fictives ou réelles. Pour estimer les mesures de l'AUV qui ne sont pas accessibles avec les données récupérées,

une interpolation "simple" est utilisée (cf. partie 6.3.3). Voici les différentes approches réalisées :

— **Première phase :** utilisation d'une carte fictive simple (cf. Figure 24 page 36).
— **Deuxième phase :** utilisation de la carte du SHOM [18] avec interpolation simple.
— **Troisième phase :** utilisation de la carte du SHOM réduite à la zone de recherche et améliorée par la méthode de krigeage qui sera détaillée plus tard. (cf, référence au groupe magnétisme), toujours avec interpolation simple.

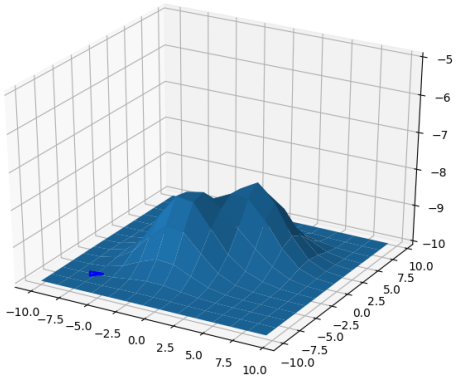


FIGURE 24 – Carte utilisée pour la phase I. à base d'exponentielles

6.3.2 Les équations : vecteur d'état et contrôleur

Le suivi d'isobathe réalisé se fait dans un plan 2D. La profondeur du robot et sa vitesse v sont supposées constantes. Ainsi, les équations d'états sont les suivantes :

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} v \cdot \cos(\psi) \\ v \cdot \sin(\psi) \\ 0 \\ u \end{bmatrix}$$

Avec (x, y, z) la position du robot, ψ son cap et u la commande donnée par la formule (6) qui suit :

$$\begin{cases} \phi &= \tanh(h+w) + frac\pi 2 \\ \text{sawtooth}(x) &= 2\arctan(\tan(\frac{x}{2})) \\ \text{u} &= \text{sawtooth}(\phi + \theta_{grad} - \psi) \end{cases} \tag{6}$$

Dans ces équations, la consigne est notée w et correspond à l'isobathe que l'on désire suivre, h l'altitude mesurée par le robot, θ_{grad} l'angle formé par le gradient du dénivelé qui est donné par la formule (7) suivante.

$$\text{grad}(x,y) = \begin{bmatrix} \frac{h_{(x+\epsilon,y)} - h_{(x,y)}}{\epsilon} \\ \frac{h_{(x,y+\epsilon)} - h_{(x,y)}}{\epsilon} \end{bmatrix} \tag{7}$$

18. Service Hydrographique et Océanographique de la Marine

6.3.3 L'interpolation simple

Soit P, le point de position (x, y, z_P) avec (x, y) la position du robot et z la profondeur d'eau à cet endroit.

L'interpolation simple se base sur l'hypothèse suivante : Si A, B et C sont trois points assez proches de P (dans le plan 2D (x, y)), alors P appartient au plan (ABC). Ainsi, en récupérant trois points connus (i.e. faisant partie des données de la carte) proches de P dans le plan (x,y), il suffit de calculer l'équation du plan (ABC) et de l'appliquer au point P pour obtenir une estimation de z_P

6.4 Remarques

Pour chacune des phases nous avons commencé par faire des tests par un robot seul, ensuite par un nuage de robots. Sur les illustrations de la figure 25 (page 37) on peut observer une vingtaine de robots répartis autour d'une position initiale désirée, puis suivre l'isobathe désirée rencontrée. On remarque qu'ils se rejoignent jusqu'à s'arrêter au même endroit sur la Figure 26 (page 37).

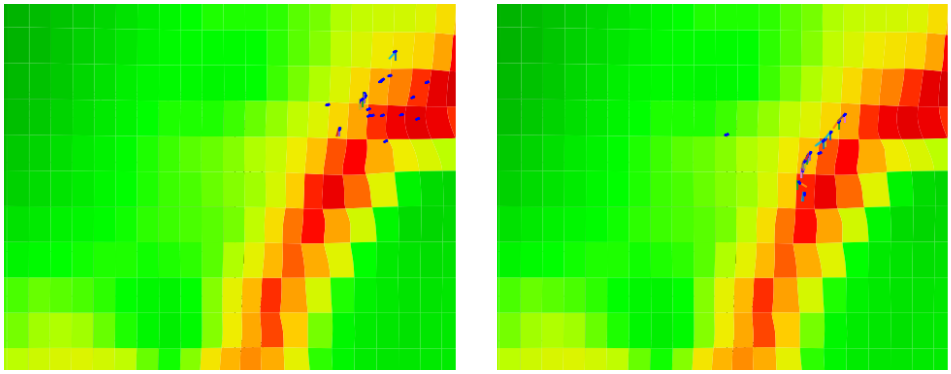


FIGURE 25 – Nuage de robots au début de la mission, puis suivant l'isobathe

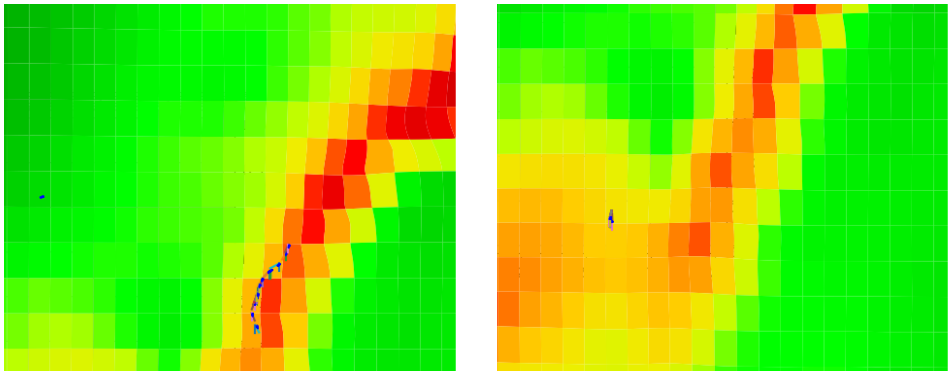


FIGURE 26 – Nuage de robots en suivi d'isobathe, puis à l'arrêt de fin de mission

Cependant, si on regarde attentivement les illustrations de la figure 25, un des AUV ne converge pas. En effet, les incertitudes sur la position de départ de mission peuvent mener à ce cas : le robot cherche l'isobathe mais comme sa direction et sa position de départ sont trop éloignées de celles prévues, il ne peut trouver l'isobathe désirée. C'est pourquoi il est très important de bien modéliser les incertitudes liées au positionnement de départ de la mission induit par le positionnement à la surface et à la phase de coulage de l'AUV.

6.4.1 L'interpolation simple

Dans la réalité l'AUV va mesurer en temps réel, alors que dans la simulation il nous faut interpoler des mesures, ce qui rajoute de l'imprécision par rapport à la réalité. Évidemment, moins la carte de mesure est fine plus l'hypothèse de base sera fausse et donc plus l'interpolation "simple" sera erronée. C'est pourquoi une interpolation par krigeage, plus robuste, a été envisagée.

6.4.2 Le type de mission

L'avantage d'une telle solution est qu'une localisation n'est pas nécessaire. Cependant, l'utilisation de trajectoire "simple" comme celle que l'on décrit ici en quatre parties ne permet pas de recouvrir une grande surface à chaque fois. Mais il est aisément possible de recourir à des trajectoires plus complexes en se servant des lignes bathymétriques et de la connaissance du terrain au préalable. En effet, en prenant en compte les isobathes les plus profondes et le temps que prend l'AUV pour les dépasser, une solution envisageable peut ressembler à la mission suivante :

1. L'AUV est déposé à la position prévue.
2. Il suit la direction prévue jusqu'à rencontrer l'isobathe prévue.
3. Il suit l'isobathe pendant un temps t_1.
4. Il se dirige dans la direction normale pendant un temps t_2.
5. Il se dirige dans la direction normale pendant un temps plus court t_3.
6. Il se dirige dans la direction normale (vers l'isobathe voulue) jusqu'à rencontrer l'isobathe ; ...

7 Magnétisme

Partie réalisée par Mohamad MEZHER et Mohamed Amine OUADRHIRI

7.1 Krigeage

La théorie du modèle de krigeage a été développée par le minier sud-Africain Danie Gerhardus Krige en 1951. C'est une technique géostatistique qui est basée sur des modèles statistiques comprenant l'auto corrélation, c'est-à-dire les relations statistiques entre les points mesurés. Elle permet ainsi d'estimer une mesure en points où on n'a pas pris de mesures.

7.1.1 Définition

Le krigeage est une méthode avec une forte précision. En effet, l'outil krigeage présuppose que la distance ou la direction liant les points d'échantillonnage reflètent une corrélation spatiale pouvant expliquer les variations de la surface. De plus, l'outil krigeage applique une fonction mathématique à tous les points, où certains points déterminés sont situés dans un rayon précis. Il détermine la valeur en sortie de chaque emplacement.

7.1.2 Utilisation

Le krigeage est utilisé dans la partie traitement de données. Dans notre cas, on va l'utiliser soit pour estimer les mesures du magnétomètre ou bien les positions du robot dans des endroits où l'on n'a pas une carte assez précise, soit pour affiner des cartes bathymétriques. On a choisi d'utiliser le modèle de krigeage pour les raisons suivantes :

- Il nous permet d'adopter une perspective probabiliste. On peut ainsi utiliser les résultats donnés par la théorie des probabilités.
- On a une estimation de l'erreur de prédiction en tout point de notre domaine d'étude, et cela en prenant en compte dans son modèle les erreurs de mesures.
- Il nous donne une idée de la précision du modèle aux points non échantillonnés.
- Le krigeage nous permet également d'adopter un point de vue statistique de nos données.

Tout cela nous permet d'avoir une prédiction optimale en chaque point.

7.1.3 Principe de krigeage

Interpolation

Pour bien comprendre le concept de krigeage, il est nécessaire de comprendre le principe de l'interpolation. Pour une interpolation, on cherche à prédire des valeurs inconnues dans une zone donnée.

L'exemple ci-dessous explique le concept d'interpolation :

On cherche à prédire le point de couleur mauve dans la Figure 27, page 40. On distingue les trois points de couleurs rouges qui sont les plus proches et on applique un calcul en se basant sur la distance. Le but est de calculer la distance séparant le point de couleur mauve de chacun des trois points rouges.

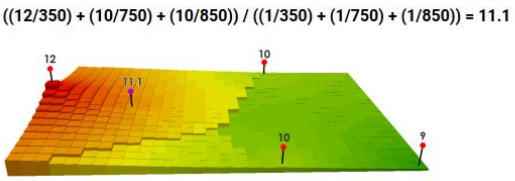


Figure 27 – Représentation d'une Interpolation en 3 dimensions.

En utilisant l'interpolation seule, on n'arrive pas à calculer la certitude de nos prédictions. Dans ce cas, on utilise la méthode de krigeage, qui crée notre surface de prédiction et une autre surface qui reflète la certitude de notre modèle. Dans la Figure 28, page 40, la première surface prédit les valeurs que l'on est en train de kriger et la deuxième, celle de l'erreur, représente l'erreur standard avec un taux d'erreur plus élevé lorsque l'on n'a pas assez de données d'entrées.

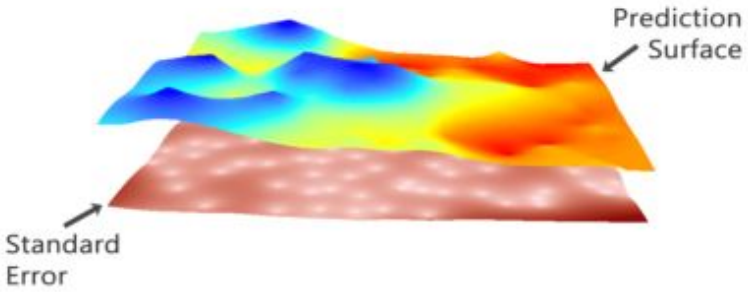


Figure 28 – Exemple de krigeage

Hypothèse sur l'erreur

Le modèle de krigeage suppose que l'erreur de donnée d'un point n'est pas liée aux autres. Ceci est le contraire des autres modèles, qui eux les considèrent indépendants. Grâce à cette hypothèse, si notre modèle est continu, ce qui est notre cas,l'erreur est aussi continue.

7.1.4 Modélisation de krigeage

Notations

Dans les parties qui suivent, on note :
- **Y :** un processus gaussien (Modélisation de notre krigeage),
- **y :** la réalisation de Y,
- **Z :** un processus gaussien (Dans notre cas Z représente la modélisation de nos données),
- **R :** la matrice de corrélation,
- **Cx :** Vecteur des coefficients,
- $\mu = F \times \beta$ (Dans le cas général).

On pose $Y(x) = \mu(x) + Z(x)$.

On considère μ : la moyenne de notre modèle. Ensuite, μ peut être vu comme un écart entre notre modélisation des données et la modélisation de notre krigeage.

$Z(x)$ est à noyau connu.

Noyau

Le noyau permet de mesurer le degré de ressemblance entre deux points dans le domaine de conception. On a deux types de noyaux de covariance :

— Le noyau stationnaire.

— Le noyau non stationnaire.

Le stationnaire est le type le plus utilisé. Le choix de ce noyau est primordial pour modéliser un phénomène. Si on choisit un mauvais noyau, notre modélisation sera faussée. Aucune méthode n'existe permettant de choisir le meilleur noyau adapté à un problème, mais on peut guider notre choix soit à travers la méthode de validation croisée, soit par calcul d'erreur via des points de validation.

Un exemple de noyau classique : noyau exponentiel et noyau gaussien, avec une possibilité de créer un noyau à partir d'opérations algébriques de noyau élémentaire. Comme nous pouvons le voir dans l'expression des deux exemples de noyaux classiques, ces derniers possèdent des hyper-paramètres qu'il faut définir.

Ces hyper-paramètres étant θ, σ et β.

Moyenne du modèle

C'est le terme μ qui définit notre type de krigeage : on en a trois.

— Si μ est connu : notre processus de krigeage est appelé krigeage simple.

— Si μ est une constante inconnue : on parle de krigeage ordinaire.

— Si μ est une fonction polynomiale inconnue : on l'appelle krigeage universel.

Modélisation finale

Pour avoir notre modélisation on doit d'abord estimer nos hyper-paramètres.

Pour estimer ces hyper-paramètres, on peut utiliser deux méthodes : soit la méthode du maximum de vraisemblance, soit la méthode de validation croisée.

Dans le cas d'un krigeage universel, c'est grâce a la fonction III.21 qu'on estime Les hyper-paramètres.

$$L\left(\mathbf{y};\theta,\sigma_z^2,\beta\right)=\frac{1}{\sqrt{2\pi\sigma_z^2}^n\sqrt{\det(\mathbf{R})}}\exp\left(-\frac{1}{2\sigma_z^2}\left(\mathbf{y}-\mathbf{F}\beta\right)^t\mathbf{R}^{-1}\left(\mathbf{y}-\mathbf{F}\beta\right)\right). \tag{III.21}$$

Ceci revient, par la méthode de maximum de vraisemblance à minimiser l'opposé du logarithme de cette fonction.

$$-l(\mathbf{y};\theta,\sigma_z^2,\beta)=\left(\frac{n}{2}\log(\sigma_z^2)+\frac{n}{2}\log(\det(\mathbf{R}))\right)\left(-\frac{1}{2\sigma_z^2}(\mathbf{y}-\mathbf{F}\beta)^t\mathbf{R}^{-1}(\mathbf{y}-\mathbf{F}\beta)\right). \tag{III.22}$$

On obtient les hyper-paramètres suivants :

$$\hat{\beta} = \left(\mathbf{F}^t \mathbf{R}^{-1} \mathbf{F}\right)^{-1} \mathbf{F}^t \mathbf{R}^{-1} \mathbf{y}. \quad \text{(III.23)}$$

$$\hat{\sigma}_z^2 = \frac{(\mathbf{y} - \mathbf{F}\hat{\beta})^t \mathbf{R}^{-1} (\mathbf{y} - \mathbf{F}\hat{\beta})}{n}. \quad \text{(III.24)}$$

Dans le cas d'un krigeage ordinaire et après simplification, cette équation devient, III.27

$$-l(\mathbf{y}; \theta) = n \log\left(\hat{\sigma}_z^2\right) + \log(\det(\mathbf{R})) \quad \text{(III.27)}$$

Ce qui nous donne les hyper-paramètres suivants :
(III 25 et III 26)

Il faut par la suite optimiser cette fonction.

On obtient l'expression du noyau de covariance que l'on a choisi. En revenant à notre première équation, on a maintenant un Z à noyau connu. C'est-à-dire que l'on a modélisé toutes nos mesures qui dans ce cas sont les mesures du magnétomètre.

Lors de l'optimisation de notre fonction on estime également notre β pour le cas général et le μ pour le cas ordinaire.

On introduit le terme de prédiction linéaire qui est égal à $(Cx^T \times y)$ et qui est défini comme combinaison linéaire de notre y par des coefficients que nous noterons Cx . On va ensuite minimiser l'erreur quadratique de cette précision. Notre modèle de krigeage devient donc (III.13) qui est l'équation de prédiction et (III.14) qui est l'équation de variance.

L'équation de variance reste inchangée.

C'est en dessinant les courbes de ces équations qu'on obtient les plans souhaités.

$$\hat{\mu} = (\mathbf{1}^t \mathbf{R}^{-1} \mathbf{1})^{-1} (\mathbf{1}^t \mathbf{R}^{-1} \mathbf{y}), \tag{III.25}$$

$$\hat{\sigma}_z^2 = \frac{(\mathbf{y} - \hat{\mu}\mathbf{1})^t \mathbf{R}^{-1} (\mathbf{y} - \hat{\mu}\mathbf{1})}{n}. \tag{III.26}$$

$$\hat{y}(\mathbf{x}) = \mathbf{f}_\mathbf{x}^t \hat{\beta} + \mathbf{r}_\mathbf{x}^t \mathbf{R}^{-1} (\mathbf{y} - \mathbf{F}\hat{\beta}), \tag{III.13}$$

$$s^2(\mathbf{x}) = \hat{\sigma}_z^2 \left[1 + \mathbf{r}_\mathbf{x}^t \mathbf{R}^{-1} \mathbf{r}_\mathbf{x} \right]. \tag{III.14}$$

7.2 Simulations

Partie réalisée par Philipe MIRANDA DE MOURA.

A partir de mesures d'anomalies magnétiques faites par des robots ou bien à l'aide de bouées plus ou moins bien localisées, il est nécessaire d'utiliser une méthode de krigeage pour reconstituer la carte des anomalies magnétiques du fond marin.

7.2.1 Simulation de bouées

Une alternative à l'utilisation d'un robot avec un magnétomètre est d'utiliser plusieurs bouées. Ces bouées sont dotées d'un système de ballasts qui leur permet de plonger et ainsi de prendre des mesures en profondeur. En remontant à la surface, on peut connaître la position de celles-ci.

Ceci est une simulation de quatre bouées qui suivent les marées.

On considère quatre robots en quatre différentes positions. Ensuite, on associe chaque robot à une cible de mouvement inconnu dont on connaît les positions, $\bar{p1}$, $\bar{p2}$, $\bar{p3}$, $\bar{p4}$ et leurs vitesses à l'instant présent. Le comportement de nos robots sera modélisé par le potentiel suivant (Équation 8, page 44) :

$$V(p) = \widehat{v^T}.p + \|p - \widehat{p}\|^2 + \frac{1}{\|p - \widehat{q}\|} \tag{8}$$

où le potentiel $\widehat{v^T}$ représente la consigne de vitesse, le potentiel $\|p - \widehat{p}\|^2$ rend attractive la position cible $\widehat{p}$. Dans notre cas, les cibles ont la même vitesse et le même

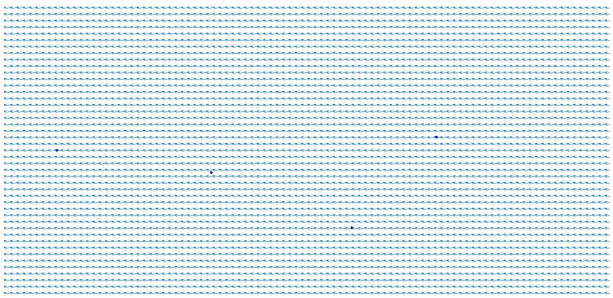


FIGURE 29 – Simulation des boues

comportement, dans le but de simuler quatre robots qui suivent un courant maritime avec la fonction d'état suivante :

$$\begin{cases} \dot{x} &= v.\cos(\theta) \\ \dot{y} &= v.\sin(\theta) \\ \dot{v} &= u1 \\ \dot{\theta} &= u2 \end{cases}$$

7.2.2 Simulation d'un robot qui effectue des rails

Pour mener à bien sa mission et prendre les mesures avec le magnétomètre, le robot doit pouvoir explorer une surface de 1 km² en faisant des rails. Voici un contrôle qui va permettre au robot de faire ses trajectoires.

On considère un robot se déplaçant en suivant des rails tout au long d'un plan et décrit par les équations suivantes :

$$\begin{cases} \dot{x} &= \cos(\theta) \\ \dot{y} &= \sin(\theta) \\ \dot{\theta} &= u \end{cases}$$

où θ est le cap du robot et (x, y) les coordonnées de son centre. Dans notre cas, on veut que le robot suive un schéma en forme des rails. On suppose que le robot est en un point m, et on crée une ligne attractive pour que le $\bar{\theta}$ se dirige vers cette ligne. En effet, plus on est loin plus la ligne est attractive.

7.3 Application de modèle de krigeage

7.3.1 Application sur les données issues du robot

On cherche ici à estimer grâce à la méthode de krigeage les positions de passage du robot. En prenant des échantillons de x et y lors de son mouvement, on cherche à reconstituer toute sa trajectoire.

Les données ont été enregistrées à partir de la simulation du robot qui fait des rails. Lors de cette simulation notre robot est contrôlé en position à une profondeur fixe.

Voici la carte obtenue après application de la méthode de krigeage sur tout l'espace (voir Figure 30, page 45).

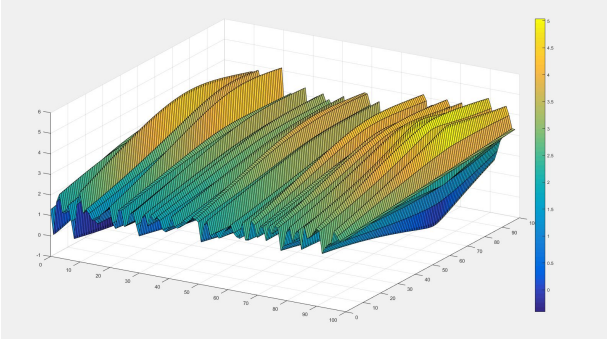


Figure 30 – Krigeage des données robots

On peut remarquer que le krigeage reconstitue bien le fait que le robot fasse des rails et que son mouvement est maintenant continu sur tout notre espace de mesure. Ce mouvement est reconstitué avec une certaine précision. Si notre estimation est en jaune, cela veut dire que la précision est maximale et que le robot est vraiment passé par là. Si elle est en bleu, cela veut dire que la précision est nulle et que l'on n'est pas sûr que le robot soit passé par cet endroit.

Comme notre robot se déplace à une profondeur fixe, on voit sur la carte de krigeage qu'à cette altitude on est sûr du mouvement du robot. Et en changeant de profondeur on perd de la précision.

7.3.2 Application sur les données du magnétomètre

L'autre application où on peut utiliser la technique de krigeage est pour estimer les mesures du magnétomètre dans des endroits où l'on n'a pas réalisé de mesures et ainsi reconstituer une carte magnétique continue de l'endroit d'exploration du robot, ceci

avec une certaine précision. On a appliqué le krigeage sur les données de la simulation de détection d'anomalies magnétiques.

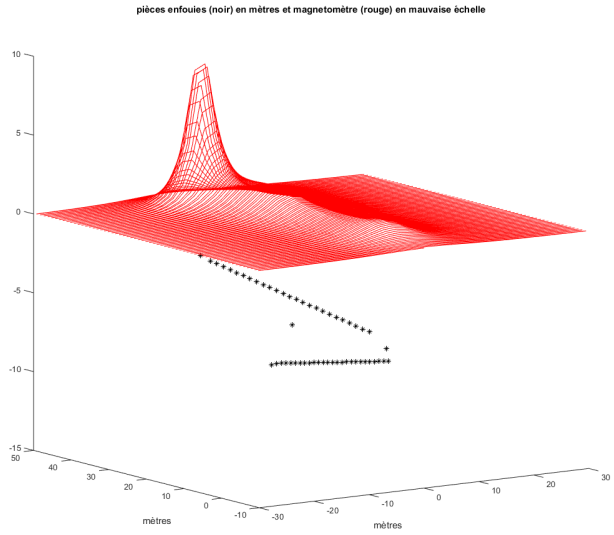


FIGURE 31 – Image représentant une trajectoire sinusoïdale du robot et la mesure du magnétomètre sur une épave enfuie inclinée.

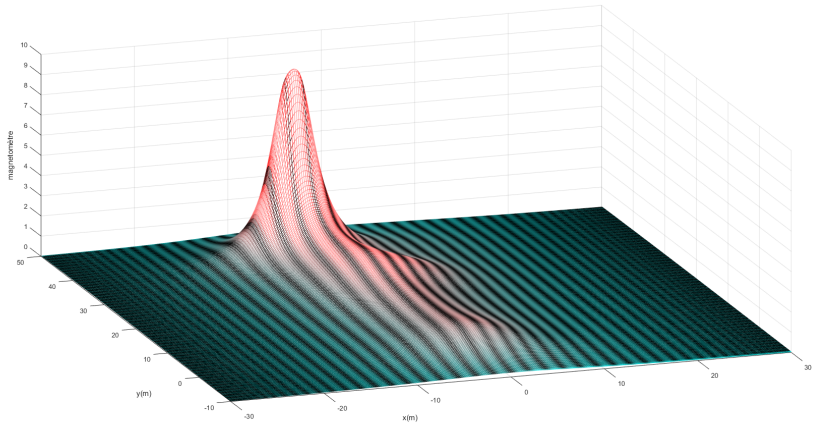


FIGURE 32 – Carte issue du krigeage sur les données de l'image 31

Le krigeage a non seulement transformé la liste de points initiale en un maillage, mais a également calculé l'incertitude associée à chaque point du maillage. Visuellement, cette information est montré sur la figure 32 par les couleurs : le rouge représente une mesure importante et fiable du magnétomètre (c.à.d. forte probabilité de présence de l'épave ou d'un autre objet enfoui, comme des câbles sous-marins), le bleu-turquoise représente une mesure également fiable, mais faible, du magnétomètre (c.à.d. que l'on est sûr de ne pas avoir des objets enfouis) et le noir représente les régions où nous ne sommes pas sûr de l'estimation car les points étaient éloignés des points mesurés. Ce code-couleur permettra d'établir donc les zones où :

— nous sommes sûr de ne pas avoir des épaves (les zones bleu-turquoise)

— nous sommes sûr d'avoir des objets enfouis (les zones rouges)
— il faut refaire des levés de données car nous ne sommes pas sûr des estimations faites

8 Carte bathymétrique

Partie réalisée par Jean WALTER

8.1 Introduction

L'objectif de cette partie est d'obtenir une carte bathymétrique de la zone sous-marine d'étude, à savoir le goulet de Brest. L'obtention de cette carte permettra de faire avancer le projet sur plusieurs points, notamment pour la reconstruction de la bataille navale sur Unity et la planification de trajectoire du robot sous-marin. Le SHOM (Service Hydrographique et Océanographique de la Marine) mettant à disposition sur son site internet des jeux de données bathymétriques de la façade Atlantique, les données utilisées dans la suite de ce rapport en sont directement extraites.

8.2 Traitement des données

8.2.1 Premiers traitements : extraction et représentation

Les données récoltées par le SHOM sont composées d'un fichier de semi de points ascii (.glz) et se présentent sous la forme longitude, latitude, altitude. Le fichier téléchargé contenant tous les relevés de la façade Atlantique (soit plus de 400 millions de points) un premier traitement a été nécessaire pour ne conserver que la zone d'étude, et ainsi diminuer considérablement la taille du fichier de données. Ayant les coordonnées géographique du goulet de Brest, les données intéressantes ont pu être extraites à l'aide de Matlab.

Avant de faire une représentation 3D des points obtenus, il est important de savoir que ces derniers ont déjà été interpolés par le SHOM.

L'objectif de cette pratique est de partir de points mesurés à intervalles de distance non réguliers et d'estimer les valeurs situées entre ces derniers. Cela permet par exemple comme ici, d'obtenir des points disposés suivant un quadrillage régulier selon la longitude et la latitude.

La figure ci-dessous représente une première représentation 3D de la zone.

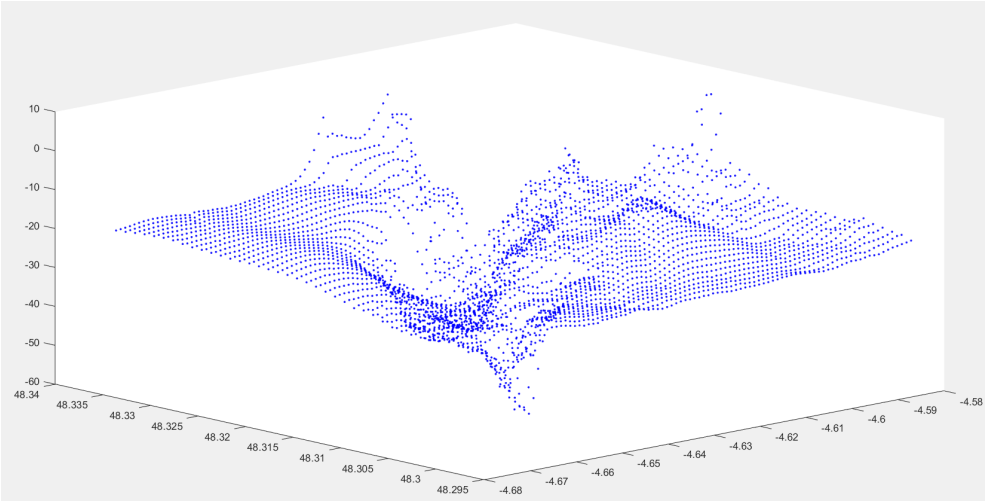


Figure 33 – Carte 3D des fonds marins

A noter que les abscisses et ordonnées sont exprimées en coordonnées géographiques et l'axe vertical en mètre. Chaque point est écarté d'une distance de 90 mètres suivant la longitude et la latitude, ce qui pose un problème pour le projet puisqu'une précision de l'ordre de la dizaine de mètres est souhaitée. Une nouvelle interpolation est nécessaire, qui se fera grâce à une méthode appelée krigeage.

8.2.2 Krigeage et interpolation

La méthode de krigeage expliquée dans la partie intitulée "Krigeage" permet d'obtenir un écart entre les points de 10 mètres. La figure suivante est obtenue et offre un meilleur visuel du fond marin.

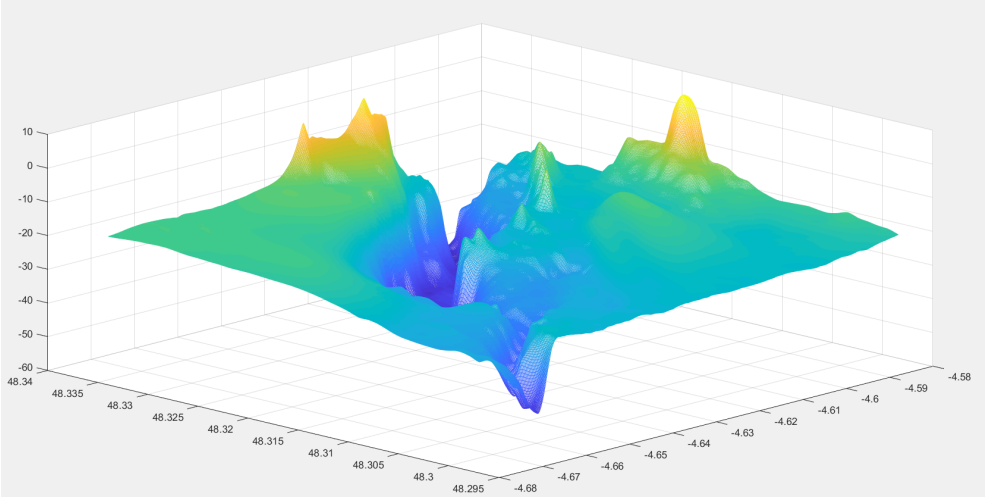


Figure 34 – Carte 3D après krigeage

Cette nouvelle carte remplissant le critère de précision de l'ordre de la dizaine de mètres, les points tracés sont exportés dans un fichier répertoriant leur longitude, latitude et altitude. Ces données peuvent ensuite être utilisées par les autres équipes du projet pour leurs besoins, notamment pour la reconstitution de la bataille navale sur Unity et la planification de trajectoires.

9 Simulation

Partie réalisée par Noëlie RAMUZAT et Rémi RIGAL.

Dans cette partie est présenté l'outil de simulation permettant la reproduction 3D de la mission de localisation de la *Cordelière*.

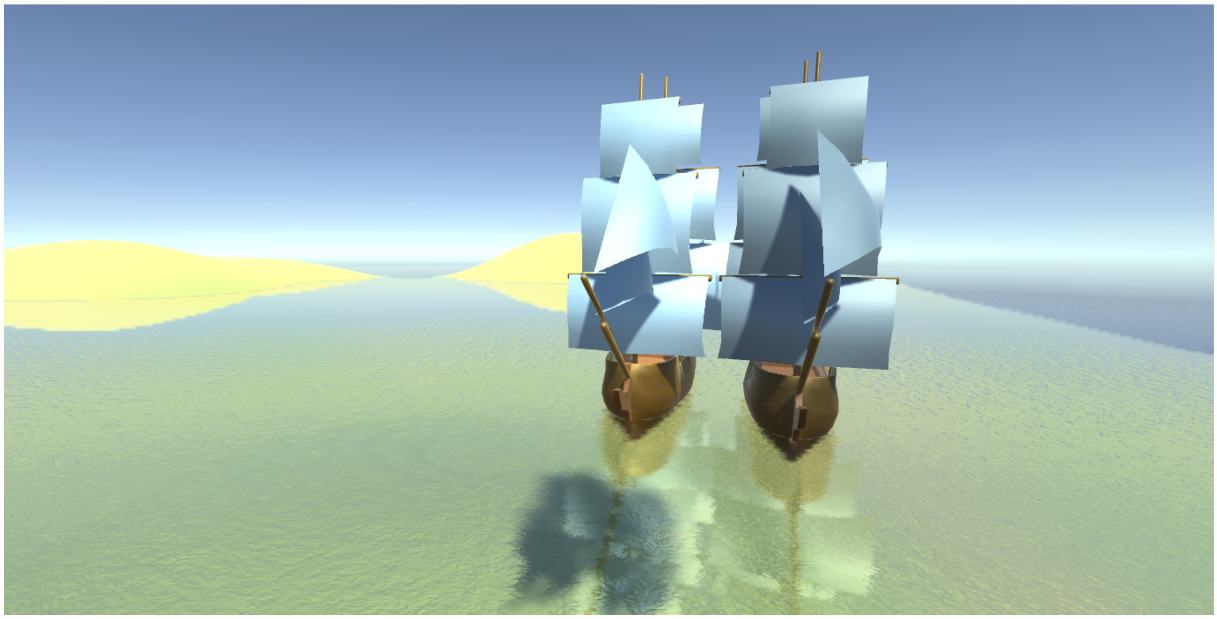


Figure 35 – Exemple de simulation

9.1 Objectifs

Du point de vue de l'architecture du système cette section se situe en fin de chaîne, elle récupère les informations en sortie du logger et les exploite pour les afficher en 3D dans Unity (cf. partie Qu'est-ce que Unity ?).

Ainsi à partir des données de position et du magnétomètre il est possible de représenter le déplacement du sous-marin et l'aspect magnétique du fond marin. Lorsque le logger récupère des informations lors d'une simulation la représentation 3D permet de mieux visualiser les résultats et de les interpréter plus facilement. Lors de la mission réelle, il est possible de suivre en temps réel les avancées du sous-marin et du sondeur ou de la rejouer en post-traitement.

De plus cet outil peut-être utilisé pour recréer la bataille de la *Cordelière* contre le *Régent*. Simple d'utilisation, il est voué à être réutilisé dans un cadre plus étendu d'aide pour la simulation en 3D.

9.2 Choix d'implémentation

Pour pouvoir simuler en 3D la localisation de la *Cordelière* il était important de chercher un logiciel graphique à la fois modulable et plutôt simple d'utilisation. Le choix s'est porté sur Unity, un moteur de jeu possédant un moteur graphique pratique et adaptable.

Le choix de Unity a été fait notamment par sa simplicité d'utilisation et son accessibilité. En effet, c'est un moteur gratuit pour sa version personnelle et libre d'utilisation et de diffusion pour des projets à but éducatif. Ces derniers critères ont joué en faveur de Unity plutôt que d'autres moteurs de jeux vidéo comme l'Unreal Engine ou encore le Cry Engine.

Néanmoins, il était impossible de s'interfacer avec le middleware ROS choisi dans ce projet et ce avec n'importe quel moteur tel que Unity. En effet ROS n'ayant pas pour but premier la représentation graphique, il n'est pas compatible avec de nombreux logiciels.

Ainsi pour palier à ce problème, il a été décidé de créer une librairie Python communiquant avec Unity. Cette librairie donne la possibilité de créer et de modifier des objets 3D ou 2D dans Unity avec un simple script. Python s'interfaçant parfaitement avec ROS, la représentation 3D à partir des informations du logger s'effectue simplement.

9.3 Qu'est-ce que Unity ?

Unity est un moteur de jeux vidéo multi-plateforme réputé pour sa rapidité d'implémentation. Il utilise le framework Mono qui est une initiative permettant de compiler du C#, utilisant initialement la machine virtuelle .Net de Microsoft, pour des plateformes différentes de Windows.

Son point fort est qu'il propose une licence gratuite permettant de développer et diffuser sans restriction lorsqu'il s'agit d'un usage non commercial. De plus, l'API Mono offre une implémentation des structures réseau assez simple, ce qui sied particulièrement à notre utilisation.

9.4 La Librairie Python

9.4.1 Implémentation

La librairie Python créée se décompose en deux parties principales :
- La communication avec Unity par le biais d'un TCP
- Les classes permettant de créer des objets 3D/2D

TCP Client

Le TCP est un protocole de transport fiable, où les applications transmettent des flux de données sur une connexion réseau. La librairie Python tient le rôle du client qui se connecte au serveur TCP, ici représenté par Unity, afin d'envoyer les commandes de l'utilisateur. Lorsqu'un script python utilisant la librairie est lancé, le client TCP est créé et la communication est initialisée. Cette action s'effectue de façon transparente pour l'utilisateur, qui ne doit préciser dans son code que la création d'une nouvelle figure ainsi que l'adresse IP de l'hôte du viewer Unity, cette commande lançant le client TCP.

Les classes

Les classes de la librairie sont réparties entre trois pôles :
- L'interface utilisateur
  - *UnityFigure* qui permet d'initialiser et de supprimer les figures et les objets
  - *UnityObject* qui définit les actions possibles à réaliser sur les objets
- Le réseau
  - *TCPClient* qui gère la communication TCP avec le serveur
  - *Callback* qui définit le modèle des messages de retour permettant de synchroniser la partie Python et Unity
- Les animations
  - *Animation* qui permet de créer des animations, c'est à dire de mettre en mouvement plusieurs objets en même temps et de fluidifier les mouvements des objets

9.4.2 Installation

La librairie est disponible sur *pip* sous le nom **PyUnityVibes**. Plus d'informations peuvent être touvées sur la page GitHub du projet [19].

19. https://github.com/NoelieRamuzat/PyUnityVibes

9.4.3 Utilisation

Ci-dessous sont décrites les différentes fonctionnalités proposées par la librairie, qu'il suffit d'appeler dans un script après avoir importé la librairie.

Fonctionnalités	**Méthodes**
Initialisation de la figure	UnityFigure(UnityFigure.FIGURE_3D)
Création d'objet 2D/3D : Sphère, Cube, Cylindre, Capsule, Bateau, Sous-marin, Galion	figure.create(ObjType, coordX, coordY, coordZ, rotX, rotY, rotZ, dimX, dimY, dimZ, color)
Suppression	obj.delete()
Mise à jour : Position, rotation, couleur, taille	obj.updatePosition(x, y, z), obj.updateRotation(x, y, z), obj.updateColor(UnityFigure. COLOR_CYAN), obj.updateSize(x, y, z)
Récupération des informations	obj.getInfo()
Suivi de caméra	obj.track()
Animation	figure.createAnimation(dt), anim.addObject(obj), figure.animate(anim)

Table 4 – Fonctionnalités de la librairie Python

9.5 L'Executable Unity

9.5.1 Implémentation

L'intérêt de cette librairie est d'être totalement transparente pour l'utilisateur, les choix qui lui sont possibles sont définis de manière intrinsèque au sein du UnityEditor.

Chaque possibilité est définie sous la forme d'une énumération dans Unity, comme par exemple les différents objets 2D et 3D qu'il est possible d'insérer. Il est par exemple possible d'instancier une voiture, un bateau, un sous-marin etc... De même les couleurs de chaque objet sont modifiables.

Finalement, l'utilisateur peut déplacer sa caméra en cours de simulation à l'aide de la souris et des touches du clavier (par défaut les touches Z, Q, S, D, C et espace). La direction pointée par cette dernière est également contrôlable à condition que la caméra ne soit pas déjà en train de suivre un objet.

9.5.2 Installation

Il est possible de trouver les exécutables du viewer Unity ainsi que son code source sur la page GitHub du projet [20].

9.5.3 Utilisation

Il est nécessaire d'exécuter le viewer Unity avant de lancer tout script Python, le serveur TCP devant être lancé en premier.

Au démarrage, il est nécessaire de choisir une résolution d'affichage pour la fenêtre du viewer.

20. https://github.com/RemiRigal/Unity-Vibes

10 Appel à projet NEPTUNE

Partie réalisée par Sophie TUTON et Maha ABOUZAID

10.1 L'objectif du projet NEPTUNE

Le projet "Neptune"[1] lancé par la Région Bretagne est un projet relatif à la nouvelle exploration patrimoniale des univers nautiques engloutis. Son rôle est d'encourager la connaissance, de renforcer la conservation et de favoriser la découverte et l'innovation. En réalité, cet appel à projet a pour objectif d'accompagner les projets qui s'inscrivent dans l'une de ces 4 familles d'actions : Connaître, Conserver, Valoriser et Innover.

10.2 La demande

Pour cela, nous avons complété une future demande faisant partie de la famille d'action "Connaissance" qui servira aux futurs appels à projet de l'année 2018.

En réalité, le financement du programme « Neptune » s'inscrit dans la volonté de redéployer l'archéologie sous-marine en Bretagne et de mieux la développer.

Les votes de confirmation pour cet appel à projet n'ont pas encore eu lieu. Il convient donc d'attendre la décision finale prise par la commission avant de pouvoir en faire largement état.

Ce projet pourra ensuite être repris par d'autres personnes pour le prolonger, l'étendre à d'autres usages, ou le finaliser.

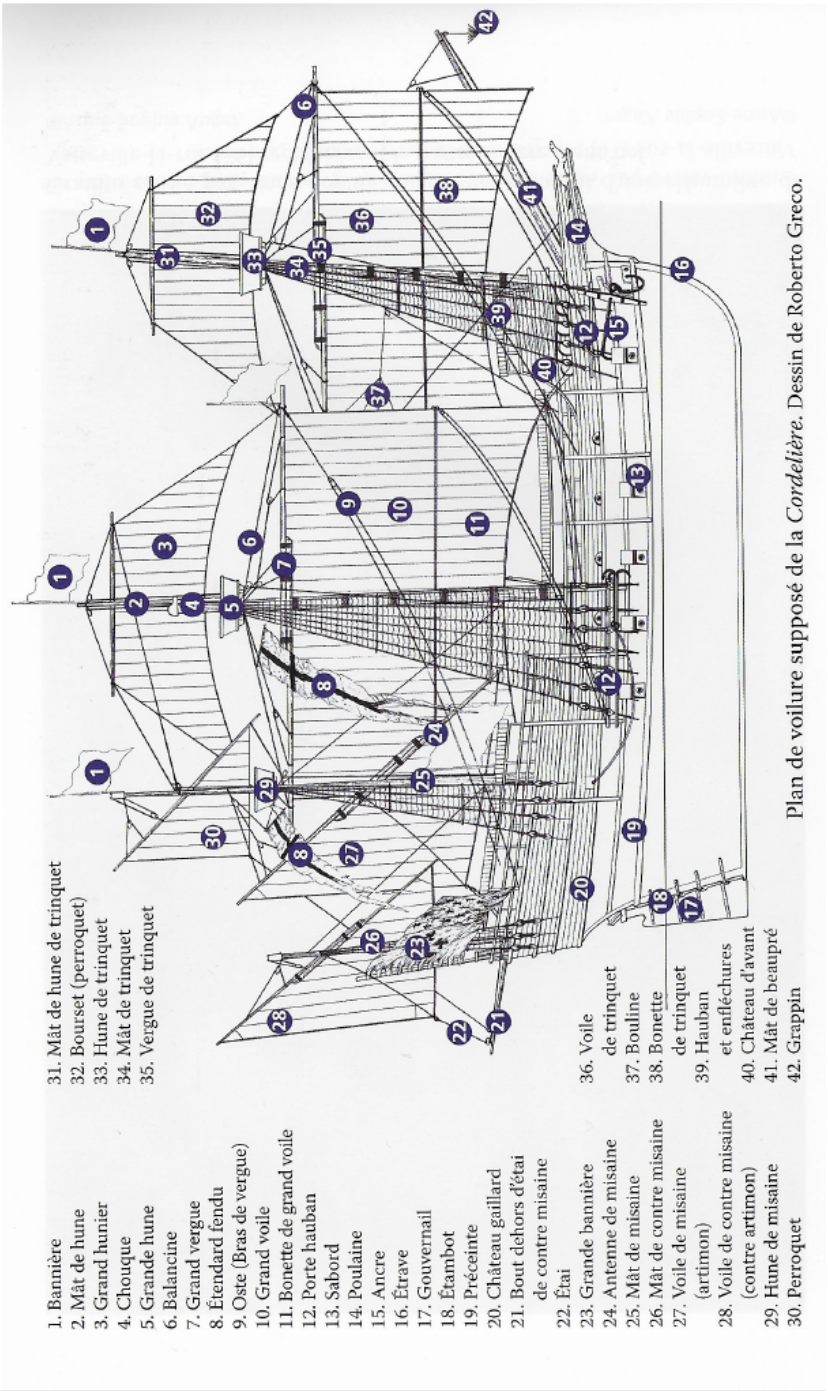
Conclusion

Ce projet fut très intéressant car pluridisciplinaire, culturel et ne nécessitant pas uniquement des compétences techniques. Il nous a permis de travailler en collaboration au sein d'un projet ambitieux, qui sera continué durant un PFE avec des essais prévus en Juin 2018.

L'essentiel du travail réalisé est disponible sur GitHub aux adresses suivantes :

- Le projet global :
  https://github.com/EnstaBretagneClubRobo/Cordeliere
- Le projet ROS :
  https://github.com/EnstaBretagneClubRobo/Cordeliere_ROSws
- Le projet PyUnityVibes :
  https://github.com/EnstaBretagneClubRobo/PyUnityVibes

Annexe A : Gréement et forme supposés de la Cordelière



Plan de voilure supposé de la *Cordelière*. Dessin de Roberto Greco.

Figure 36 – Gréement et forme supposés de la Cordelière ([18])

Annexe B : Cartes résumant les hypothèses actuelles concernant la bataille de Saint-Mathieu

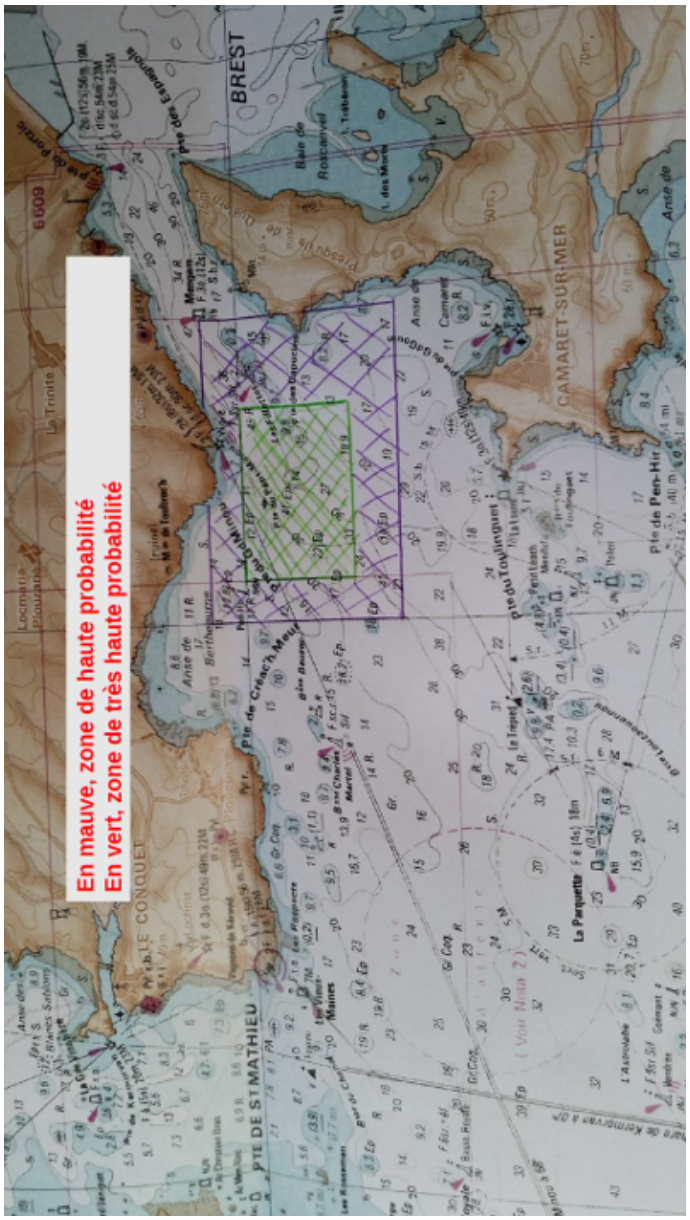


Figure 37 – Zone de recherche.

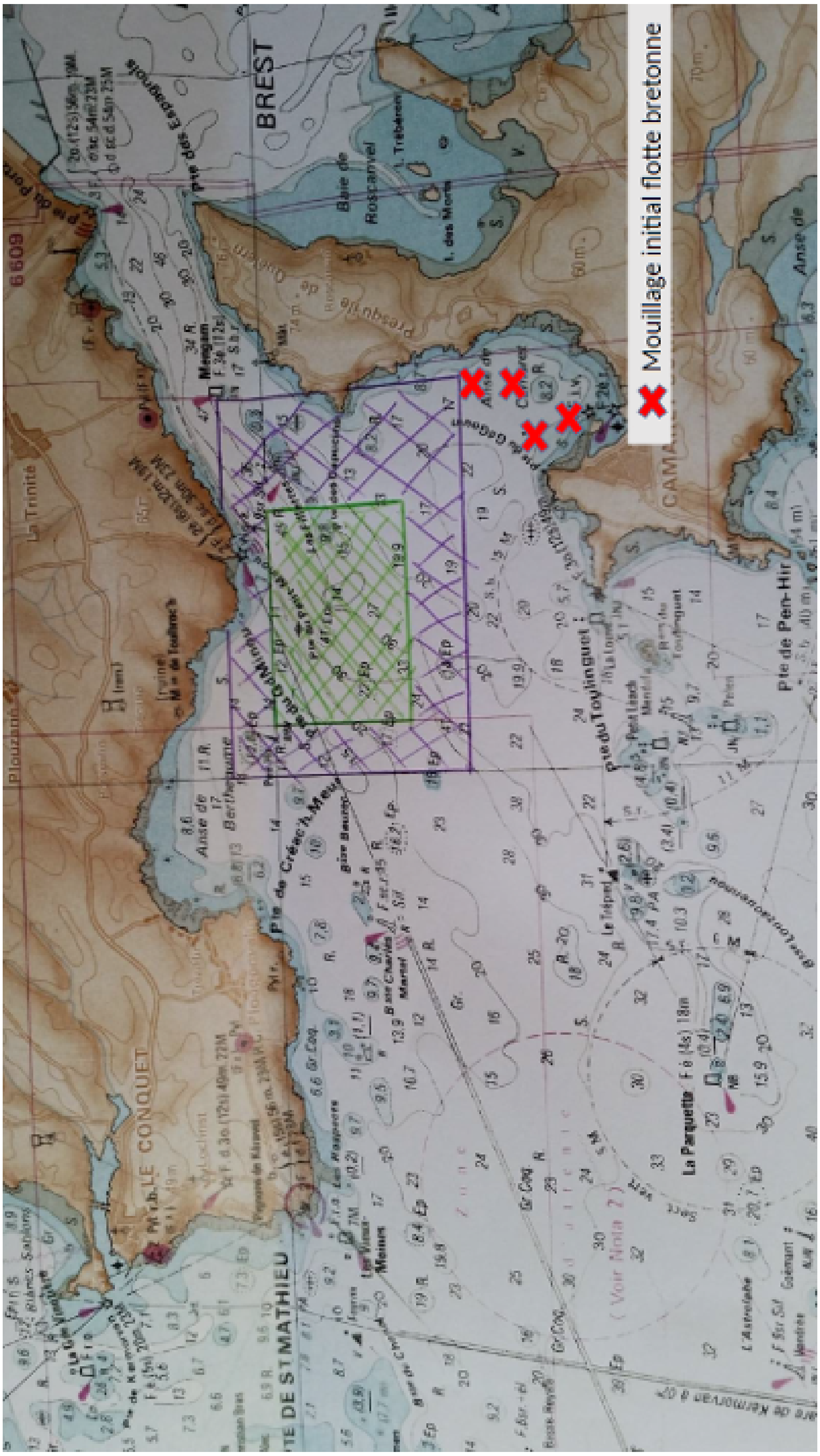


FIGURE 38 – Mouillage de la flotte.

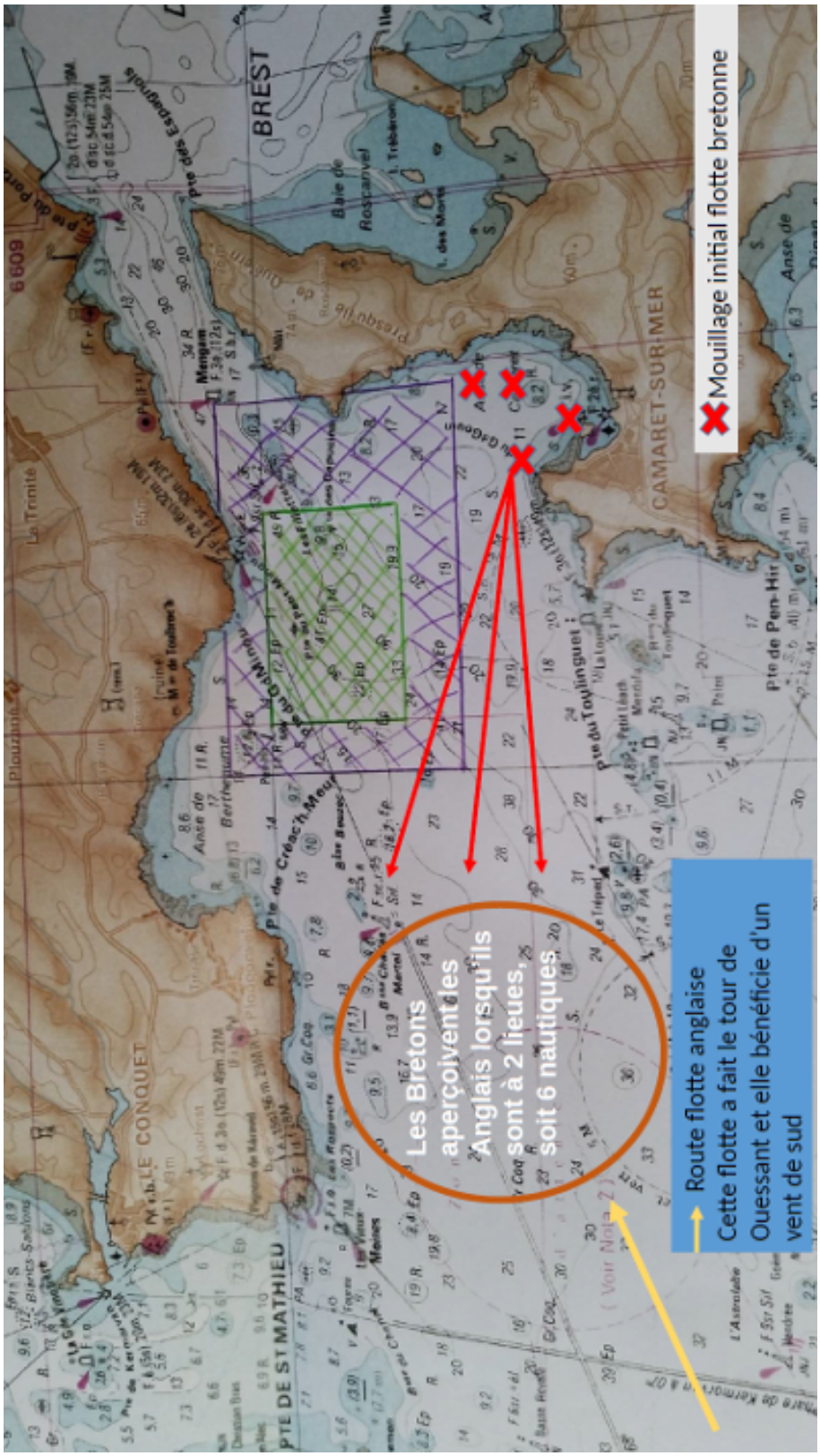


Figure 39 – Arrivée des Anglais vers 11h.

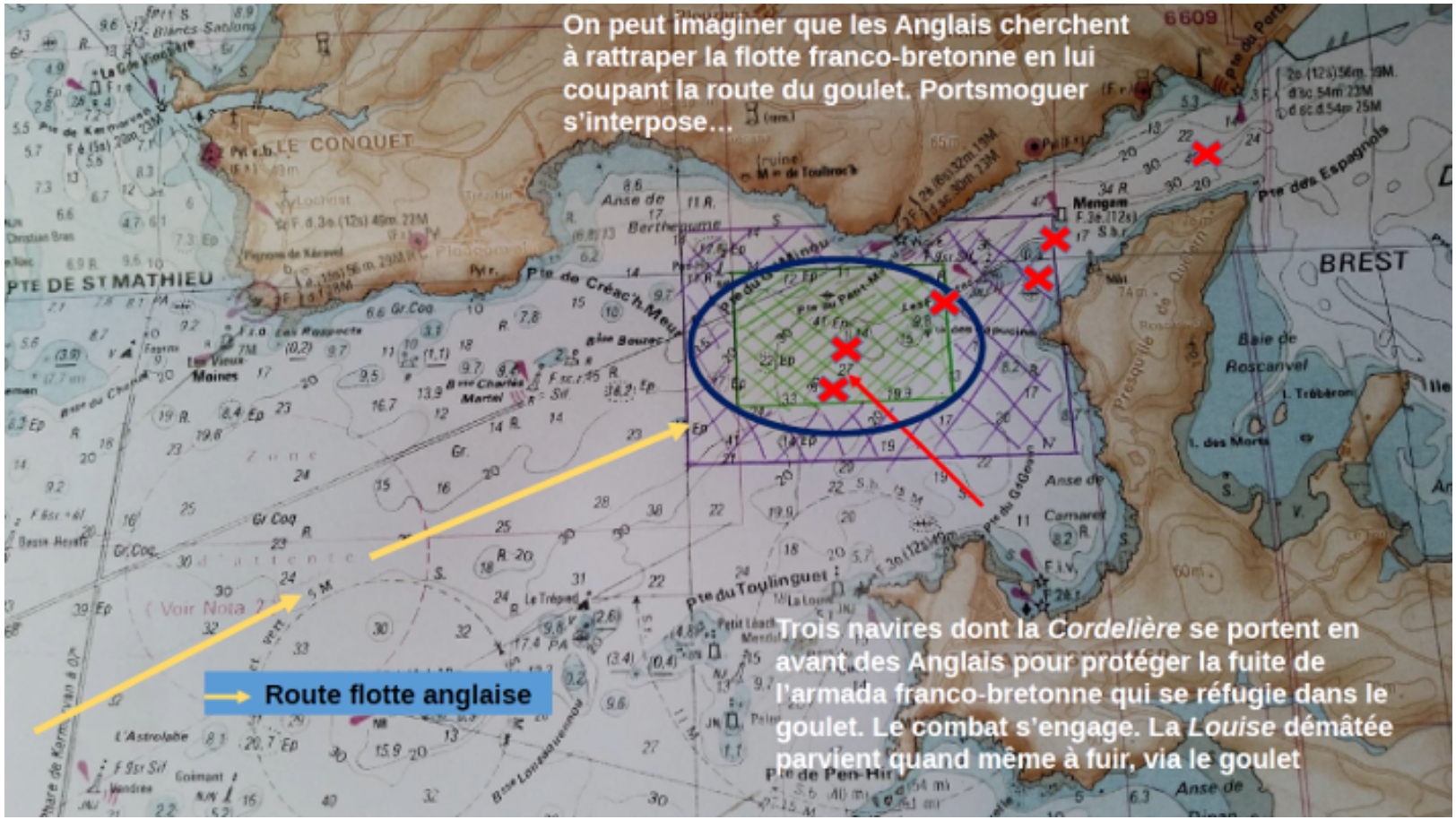


FIGURE 40 – SCÉNARIO DU DÉBUT DE BATAILLE

Annexe C : Architecture logicielle

Schéma global

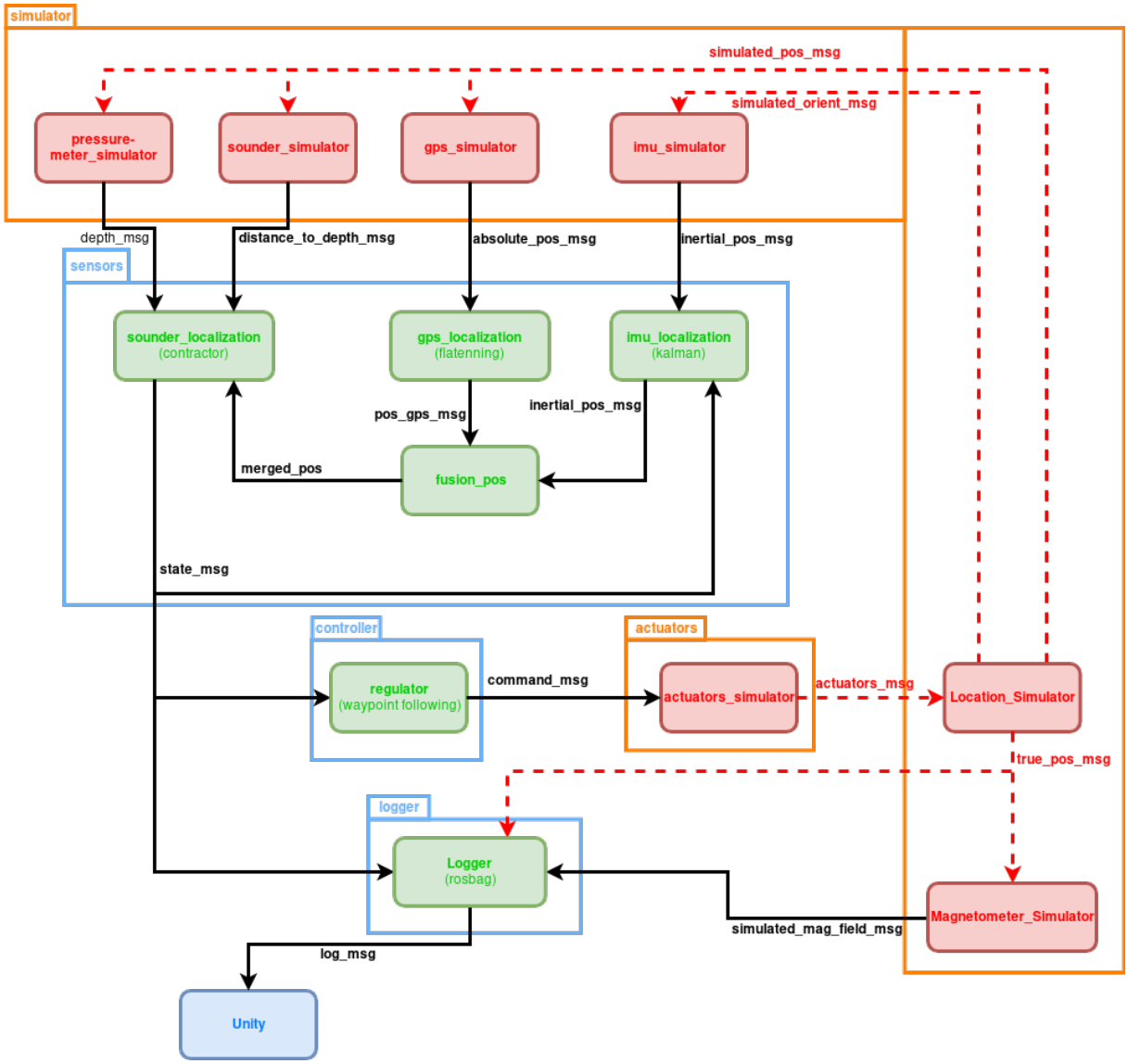


FIGURE 41 – Architecture logicielle (La légende est dans la Figure 42, page 63)

Légende du schéma

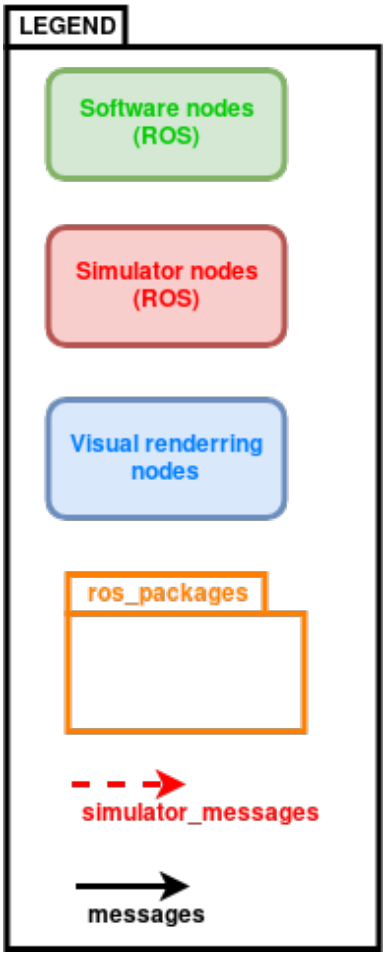


Figure 42 – Légende du schéma précédent (Figure 41, page 62)

Table des figures

Liste des tableaux

Glossaire

allure l'allure désigne la direction d'où provient le vent. Par exemple, le vent arrière est l'allure d'un voilier qui avance avec un vent soufflant sur son secteur arrière.. 7

AUV Autonomous Underwater Vehicle : sous-marin autonome. 17, 18

mâts d'artimon c'est le (où les) mâts les plus en arrière d'un bâtiment, souvent présents sur le château arrière.. 8

VCS Version Control System. 19

Références

[1] La Région Bretagne. Appel à projet neptune. http://www.bretagne.bzh/jcms/prod_417210/fr/patrimoine-neptune, 2017-2018.

[2] I. Friel. *The Good Ship. Ships, Shipbuilding and Technology in England 1200-1520*. 1995.

[3] A. Konstam. *The History of Shipwrecks*. New York : Lyons Press, 2002.

[4] C. Fernandez. *Naufragios de la Armada Española, Establecimiento Tipografico de Estrada*. Diaz y lopez, 1867.

[5] A. Spont. *Letters and Papers relating to The War in France 1512-1513*. Navy Records Society, 1897.

[6] P. Mardsen. *MARY ROSE :YOUR NOBLEST SHIPPE*. The Mary Rose Trust Ltd, 2009.

[7] A. Jal. *Architecture navale*. Arthus Bertrand, 1840.

[8] J. Flatman. *Ships and Shipping in Medieval Manuscripts*. 2009.

[9] A. Jal. *documents inédits pour l'histoire de la marine au XVI siècle*. Imprimerie Royale de paris, 1842.

[10] G. C. V. Holmes. *Ancient and Modern Ships. Part 1 : Wooden Sailing Ships*. Wyman and Sons, 1906.

[11] P. Marsden. *Ships of the Port of London :Twelfth to Seventeenth Centuries AD*. 1994.

[12] M. Oppenheim. *Naval account and inventories*. Navy Records Society, 1896.

[13] Open Source Robotics Foundation. ROS wiki. http://wiki.ros.org/, 2017.

[14] W. Harmon Ray Gary M. Scott, Jude W. Shavlik. Refining pid controllers using neural networks. https://papers.nips.cc/paper/503-refining-pid-controllers-using-neural-networks, 1991.

[15] D.KatzbR.MattKretchmara Charles W.AndersonaDouglas, C.HittlebAlon. Synthesis of reinforcement learning, neural networks and pi control applied to a simulated heating coil. https://www.sciencedirect.com/science/article/pii/S0954181097000046, 1997.

[16] Andrew Y. NgAdam CoatesMark DielVarun GanapathiJamie SchulteBen TseEric BergerEric Liang. Autonomous inverted helicopter flight via reinforcement learning. https://link.springer.com/chapter/10.1007/11552246_35, 2006.

[17] BRIAN DOLHANSKY. Artificial neural networks : Mathematics of backpropagation (part 4). http://briandolhansky.com/blog/2013/9/27/artificial-neural-networks-backpropagation-part-4, Je sais pas.

[18] M. Guérout. *Le dernier combat de la Cordelière*. Le Télégramme, 2012.