GPA791 : Projet Spécial

Identification d'un sous-marin autonome représenté à l'aide du quaternion et asservissement avec un contrôleur MPC

Par :

Lamarre, Alexandre

Desgagné, Alexandre

Montréal, le 22 avril 2021

REMERCIEMENTS

Bien évidemment, un projet de cette taille serait impossible sans le travail passionné des gens qui ont fait partie de près ou de loin du projet ainsi que le support moral de nos familles et amis. C'est pourquoi nous désirons prendre le soin d'adresser nos plus sincères remerciements aux personnes suivantes.

Tout d'abord, nous souhaitons inévitablement remercier M. Rachid Aissaoui qui nous a supervisés tout au long de notre projet. Il nous a grandement aidés dans nos réflexions, nos choix ainsi que l'acquisition de connaissances techniques. Il nous a offert un support inestimable pour la réalisation totale de notre projet.

Nous souhaitons également remercier nos amis et membres du club S.O.N.I.A pour le support qu'ils nous ont donné tout au long du projet. Tout d'abord, nous souhaitons remercier Francis Alonzo, le représentant électrique du club pour avoir établi la communication avec le DVL et son aide précieuse pour la mise en page de ce rapport. Nous remercions également Lucas Mongrain pour son travail sur la communication avec l'IMU et François Côté Raiche pour le développement des simulateurs. Nous remercions aussi l'aide d'Alexandre Leblanc, de Félix Aubin-Perron et de Louis-Guillaume Lemay pour la définition des constantes du modèle physique. Finalement, nous remercions l'entièreté des vétérans de S.O.N.I.A pour le travail effectué au préalable, car nous ne serions pas rendus où nous en sommes sans eux.

Nous aimerions remercier le directeur du département de génie de la production automatisée, M. Marc Paquet, pour l'approbation de ce projet. De plus, nous remercions aussi le professeur M. Rachid Aissaoui, le professeur M. René Jr. Landry, M. David Morgan, responsable des clubs scientifiques et technologiques ainsi que Mme Martine Guertin de nous avoir permis de suivre le cours de maîtrise SYS-823 même si nous n'avions pas le nombre de crédits suffisants.

Finalement, nous croyons important de mentionner le support moral de nos familles. Nous désirons souligner la patiente de nos partenaires, car nous savons que nous avons parfois travaillé très tard.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ACRONYMES

S.O.N.I.A : Système d'Opération Nautique Intelligent et Autonome

ROV: Remotely Operated Underwater Vehicle

AUV: Autonomous Underwater Vehicle

ROS: Robotic Operation System

MPC: Model Predictive Control

NED: North East Down

IMU: Inertial Measurement Unit

DVL: Doppler Velocity Log

$\mathbb{R}^{a\times b}$: Matrice de réel de « a » lignes par « b » colonnes

A^T : transposé de la matrice A.

$\dot{x}$: Dérivé de x.

PVC: Polychlorure de vinyle « Polyvinyl Chloride »

PWM: Modulation de largeur d'impulsion « Pulse-Width Modulation »

INTRODUCTION

S.O.N.I.A (Système d'Opération Nautique Intelligent et Autonome) est un club scientifique de l'École de Technologie Supérieure qui conçoit et fabrique des sous-marins autonomes depuis 20 ans. S.O.N.I.A a été fondé en 1999 et ses membres ont conçu un total de 8 sous-marins autonomes depuis. Le club est composé d'étudiants de programmes en mécanique, en électrique, en production automatisée et en logiciel. Le principal objectif de S.O.N.I.A est de participer à l'événement annuel de Robosub à San Diego (Californie). Cette compétition a pour but de tester l'autonomie des sous-marins et les compétences des regroupements étudiants en matière d'innovation en mettant en place diverses missions sous l'eau. Cette compétition permet entre autres de mesurer la progression réalisée durant l'année. Dans cette compétition, S.O.N.I.A doit réaliser plusieurs tâches complexes telles que l'identification d'obstacles, la prise et le dépôt d'objets, le suivi de signaux acoustiques et ce, sans intervention humaine. Présentement, S.O.N.I.A supporté deux prototypes (AUV7 et AUV8) et nous verrons en détail la présentation de ces sous-marins dans ce rapport. Voici un aperçu des deux sous-marins ci-bas :

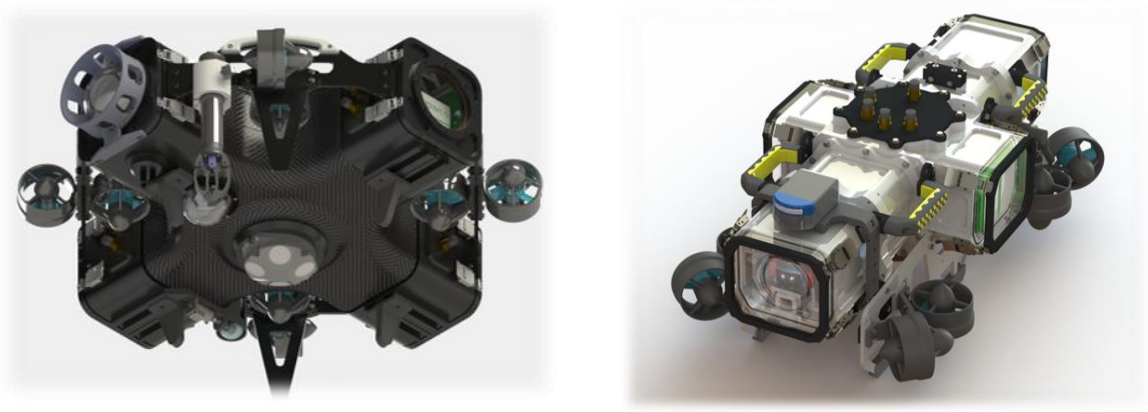


Figure 1: AUV7 et AUV8 (De gauche à droite)

Dans ce rapport, nous allons employer le terme AUV pour « Autonomous Underwater Vehicule ». Brièvement, la différence entre un ROV (Remotely Operated Underwater Vehicule) et un AUV provient du fait que le ROV est un véhicule connecté à un poste de commande à l'aide d'un lien filaire pour envoyer et extraire de l'information du sous-marin. En plus d'être une méthode pour contrôler le sous-marin, ce lien peut également servir d'alimentation pour le véhicule. Un AUV, quant à lui, fonctionne avec une batterie et peut opérer sans intervention externe.

Dans ce document, nous exposerons nos réflexions, nos choix et nos démarches relatives au développement d'un contrôleur pour le sous-marin. Ce dernier est divisé en plusieurs sections. Dans la première section, nous décrivons les motivations qui nous ont poussés à faire ce projet, le but du projet spécial et la philosophie du club S.O.N.I.A. Ensuite, dans la deuxième section, nous allons décrire l'identification du problème qui a mené à ce projet. Nous allons, entre autres, décrire les tâches de la compétition qui nécessitent la bonne conception d'un contrôleur. Dans la troisième section, nous allons faire une brève présentation du sous-marin, des capteurs et des actionneurs relatifs au contrôle. Dans la quatrième section, nous allons présenter l'identification du sous-marin. Celle-ci détaillera les hypothèses posées, les définitions utiles, l'ensemble des concepts entourant la modélisation du sous-marin ainsi que l'étude de stabilité du système. La cinquième section montrera la façon dont nous avons déterminé certaines constantes du modèle physique. Par la suite, la sixième section détaillera la méthode pour asservir le sous-marin. On y présentera l'algorithme de contrôle utilisé ainsi que l'implémentation de ce dernier dans Simulink. La sixième section présentera tout ce qui a à savoir sur l'implémentation logicielle de ce projet tel que l'intégration à l'architecture logicielle de S.O.N.I.A, l'utilisation de ROS dans Simulink ainsi que le déploiement du contrôle sur le sous-marin. Nous passerons ensuite au travers des fonctionnalités que nous avons développées autour du contrôleur tels que les différents modes de contrôle ainsi que la connexion à des simulateurs. Pour terminer, nous allons présenter et analyser des résultats de simulations.

1 Motivation

Le cours de GPA791 (Projets spéciaux) est une occasion de réaliser un projet d'ingénierie pour les étudiants qui participent à des compétitions dans le cadre d'un club technologique de l'ÉTS ou un projet d'initiation à la recherche. Ce cours permet à un étudiant de s'impliquer dans un projet d'ingénierie en participant à toutes les étapes de ce projet en passant par l'identification du problème, la recherche de solutions, l'analyse de ces solutions, l'élaboration d'un plan de travail pour mener à terme les objectifs établis, le développement de la solution ainsi que la documentation et la présentation du projet. Étant membre de S.O.N.I.A depuis quelques années déjà, nous avons rapidement sauté sur l'opportunité de faire un projet pour le club et le choix du projet n'a pas été difficile à faire.

Nous sommes deux étudiants passionnés par le contrôle et l'asservissement et le développement d'un tel projet pour S.O.N.I.A était exactement ce que nous avions envie de faire. Nous avons choisi de faire ce projet en respectant la philosophie de l'équipe. Les membres de S.O.N.I.A partagent des valeurs communes que nous trouvons également très importantes et que nous voulons respecter. Le club S.O.N.I.A est un regroupement qui désire innover et aller de l'avant dans les solutions qu'il développe. Nous désirons toujours faire l'amélioration de ce qui est en place pour rendre les prototypes meilleurs à chaque version. Nous croyons également que le partage de connaissances est très important et c'est pourquoi nous désirons faire de ce projet « un projet Open Source » pour permettre à d'autres équipes d'innover et de développer un meilleur sous-marin à leur tour. Le club partage déjà l'entièreté de ses dépôts GitHub pour aider d'autres équipes qui participe à la compétition ou toute autre personne désirant concevoir sous-marin autonome.

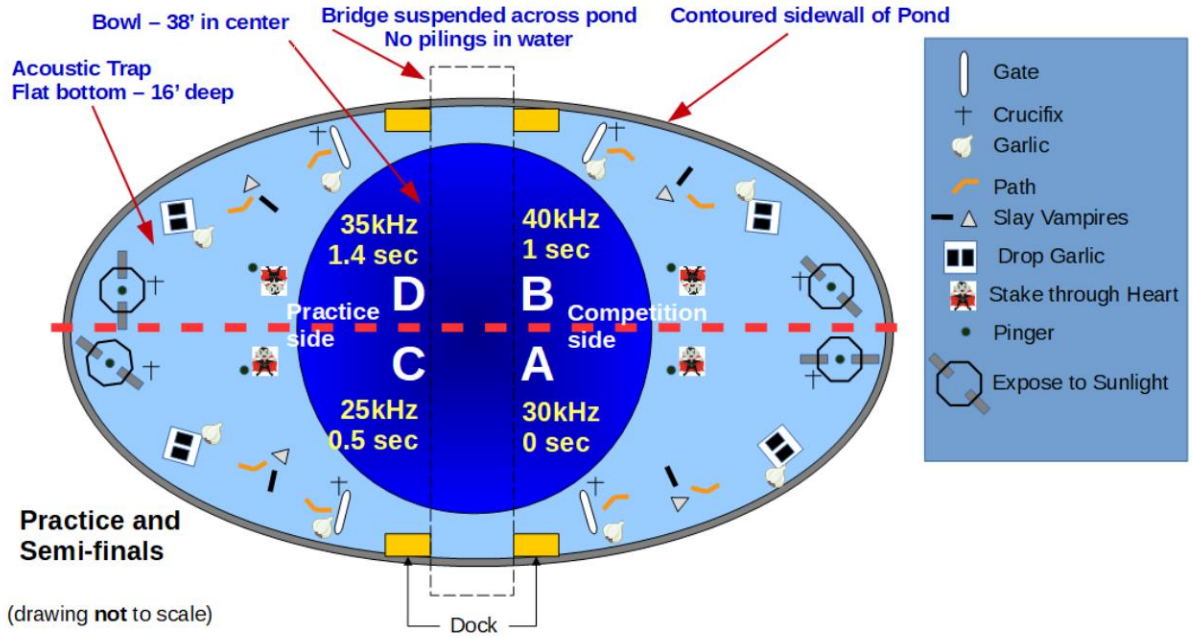
Le but de ce projet spécial est de concevoir un contrôle modulaire qui pourra être utilisé pour nos deux prototypes courants ainsi que nos prototypes futurs. Ce contrôle pourra s'adapter en donnant des constantes spécifiques que nous définirons plus tard dans ce rapport. Nous voulons également concevoir un contrôle simple à utiliser et modifier pour permettre aux autres membres ainsi qu'à la relève de S.O.N.I.A de l'utiliser tel qu'il se doit et c'est pourquoi nous allons également hautement documenter ce projet. Nous désirons documenter le développement, le fonctionnement ainsi que l'utilisation du contrôle en détail. Nous trouvons très important de documenter, mais il est également très important pour nous de partager nos réflexions afin de laisser une traçabilité de nos choix en rapport à notre projet. Les prochains membres pourront alors s'impliquer dans ce projet à leur tour et continuer d'innover en matière de contrôle pour en faire profiter le club S.O.N.I.A ainsi que la communauté des créateurs de sous-marins autonomes.

2 Identification du problème

Impact du contrôle du sous-marin sur les tâches

Dans cette section, nous allons discuter des spécifications du contrôleur dans les différentes tâches de la compétition. Nous allons regrouper les tâches de la compétition en fonction de la contribution du contrôleur. La figure suivante montre un aperçu des tâches de la compétition de 2019. Pour plus de détails,

2.1 vous pouvez consulter le document officiel de la compétition au lien suivant : https://robonationforum.vbulletin.net/filedata/fetch?id=2196. Les tâches ne varient pas énormément avec les années.



2.1.1 *Figure 2 : Répartition des tâches 2019* [1]

Les tâches d'alignement visuel

Durant la compétition, notre sous-marin doit effectuer plusieurs tâches qui nécessitent d'aligner le sous-marin avec des données visuelles obtenues par les caméras du devant et du bas. Nous devons utiliser des filtres de vision conventionnelle ou le « deep learning » pour reconnaître certaines formes ou images dans l'eau. Ces images ou formes vont donner de l'information sur l'orientation et la position que le sous-marin doit prendre pour s'aligner. Le bon alignement du sous-marin permettra d'accomplir les tâches visuelles et c'est pourquoi nous avons besoin d'un bon contrôle pour réaliser cela.

[1] Tirée de RoboSub Mission and Scoring (2019)

Les tâches d'alignement acoustique

Chaque parcours de la compétition contient 2 émetteurs acoustiques « Pinger » réglés à des fréquences différentes. Il y en a un à la tâche des torpilles et l'autre est à l'octogone. Les différentes fréquences ne sont pas activées au même moment. Elles sont toutes décalées à des intervalles de 0.5 seconde. Nous avons la possibilité de définir l'émetteur acoustique actif ou il peut être défini de façon aléatoire pour davantage de points.

2.1.2

Le sous-marin possède des hydrophones qui sont en mesure de capter les ondes acoustiques et de calculer l'angle d'élévation ainsi que l'angle de direction. La figure suivante montre le vecteur déterminer par les hydrophones :

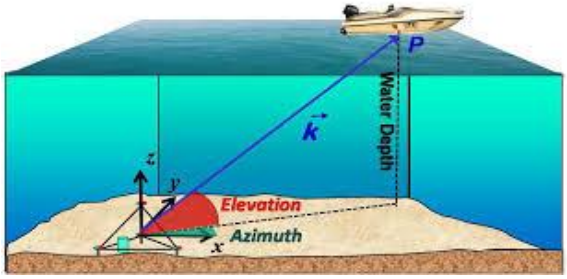


Figure 3 : Représentation des hydrophones [2]

Les émetteurs acoustiques sont placés à des endroits éloignés. Ceci nécessite que le sous-marin doive se déplacer rapidement sans accumuler trop d'erreurs. Pour la tâche des torpilles, le sous-marin doit seulement aller vers la direction de l'émetteur acoustique jusqu'à tant que les camera détecter la présence de l'obstacle. La tâche de l'octogone est beaucoup plus exigeante elle ne nécessite de s'aligner au-dessus de l'émetteur et de faire surface dans l'octogone. Cette tâche est risquée, car si nous faisons surface en dehors de l'octogone, notre essai est terminé.

2.1.3

Les tâches de suivi de trajectoire

Durant la compétition, le sous-marin doit également effectuer des tâches qui demandent de suivre des trajectoires spécifiques. À titre d'exemple, le sous-marin devra, dès le premier obstacle, traverser une large « porte » et il est possible d'obtenir des points supplémentaires de style si le sous-marin réussit à traverser la porte avec style. Pour ce faire, nous pourrions alors générer une trajectoire pour faire un tonneau ou une chandelle dans la porte. Les trajectoires seraient également utilisées pour les déplacements entre les tâches. Pour effectuer ce genre de tâche, nous avons bien entendu besoin d'un contrôleur adéquat.

[2] Tirée de Localization of small surface vessels through acoustic data fusion of two tetrahedral arrays of hydrophones (2014, p. 4)

Déroulement de la compétition

Dans cette section nous allons présenter le déroulement et les tâches de la compétition. La compétition se déroule sur le site du TRANSDEC, un basin du « Naval Information Warfare Center Pacific ». Cet ovale a un grand diamètre d'environ 90m et un petit diamètre d'environ 50m. La figure suivante **2.2** monte une vue arienne du site de la compétition :

Figure 4 : Vue aérienne du TRANSDEC [3]

La compétition se déroule sur 7 jours. Voici un horaire traditionnel:

- Jours 1 à 4 : Qualification et pratiques.
- Jours 5 à 6: Demi-Finales
- Jours 7 : Finale

Dans les prochaines sections, nous allons expliquer en détail le déroulement des étapes de la compétition.

[3] Tirée de RoboSub Mission and Scoring (2019)

Qualification

Pour qualifier notre sous-marin durant la compétition, il faut que celui-ci passe dans un portail en PVC de 10 pieds de large situé à une distance de 10 mètres du point de départ. Il faut donc que notre véhicule soit en mesure d'avancer en ligne droite sans accumuler beaucoup d'erreurs.

2.2.1 La compétition offre la possibilité de préqualifier notre sous-marin et d'avoir plus de temps de pratique durant la compétition. Cependant, la tâche est beaucoup plus complexe. Il faut que le sous-marin passe le portail, fasse le tour d'un poteau et repasse par le portail. La figure suivante montre en détail la tâche :

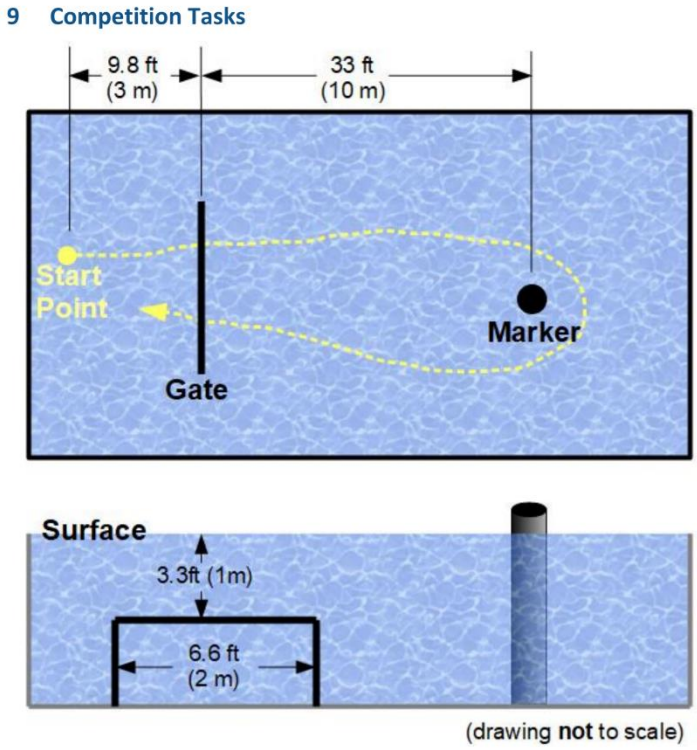


Figure 5 : Tâche de préqualification [4]

Nous devons filmer la performance du sous-marin pour que la compétition approuve la préqualification. Nous préqualifions à l'habitude notre sous-marin au mois de mars. Cette tâche nécessite le développement d'un contrôleur performant. Cette tâche est primordiale pour accéder aux tâches suivantes.

[4] Tirée de RoboSub Mission and Scoring (2019)

Demi-finales

Les équipes préqualifiées pourront ensuite se rendre en demi-finale. Les équipes auront alors la chance de faire les missions de la compétition afin de marquer le plus de points possible. L'essai en bassin pour l'équipe se terminer si le temps permis est écoulé, si l'équipe décide de mettre fin à la mission, si les
2.2.2 juges décident de terminer la mission ou si le sous-marin sort de l'eau pendant ou à la fin de la mission.

Finale

Une fois les demi-finales terminées, les meilleures équipes s'affrontent en finale. Leur pointage est alors remis à zéro et ils doivent accumuler des points en effectuant des tâches à nouveau. Ces points seront
2.2.3 attribués selon la performance de leur AUV. De plus, la finale se déroule sur la moitié du bassin et non sur le quart. Les obstacles sont donc plus éloignés. La finale permettra de classer les meilleures équipes entre elles et ce classement sera mis au-dessus des autres équipes qui n'ont pas passées en final.

3 Présentation du sous-marin

Comme mentionné plus tôt, ce travail porte particulièrement sur l'asservissement de notre nouveau sous-marin (AUV8). Dans cette section, nous allons brièvement décrire les caractéristiques de notre système que nous voulons asservir ainsi que son environnement. AUV8 est principalement constitué d'aluminium 6061-T6 pour toutes les parties étanches ainsi que pour le châssis. Il possède aussi des couvercles en acrylique pour avoir un visuel sur l'intérieur. Le sous-marin à une taille d'environs (30" x 15" x 12"). La figure suivante montre les différentes parties du sous-marin.

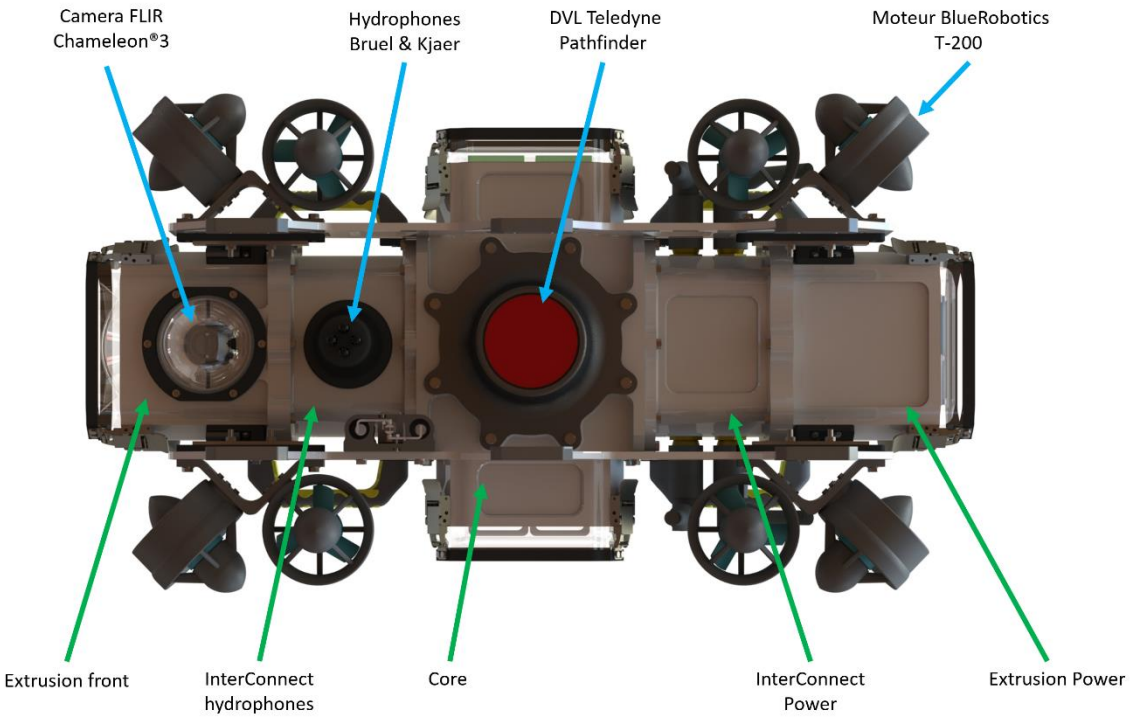


Figure 6: Anatomie du sous-marin

3.1 Sécurité et environnement

La compétition à laquelle nous participons impose des restrictions et des règles de sécurité sur la conception de nos sous-marins. Premièrement, il y a des contraintes sur le poids ainsi que la dimension du véhicule. De plus, à des fins de sécurité, la compétition exige que le sous-marin possède un arrêt d'urgence et qu'il flotte au repos. De cette façon, si le sous-marin fait une manœuvre dangereuse, le plongeur peut l'arrêter et il remontera à la surface.

Les tâches de la compétition n'exigent pas d'aller à une profondeur plus grande que 5 mètres. De plus, le sous-marin opère dans un bassin d'eau douce où le niveau de perturbation est faible. En dehors de la compétition, le véhicule opère principalement dans des piscines intérieures.

Capteurs

Afin d'assurer que nous pouvons asservir le sous-marin en boucle fermée, nous utilisons principalement un IMU (VectorNAV VN-100) et un DVL (Teledyne PathFinder). Ces deux capteurs permettent de mesurer les états du sous-marin. Nous avons suivi les recommandations de Teledyne pour

3.2 placer nos capteurs. La vue de coupe du dessus du sous-marin suivant montre l'emplacement des capteurs ainsi que leur référentiel.

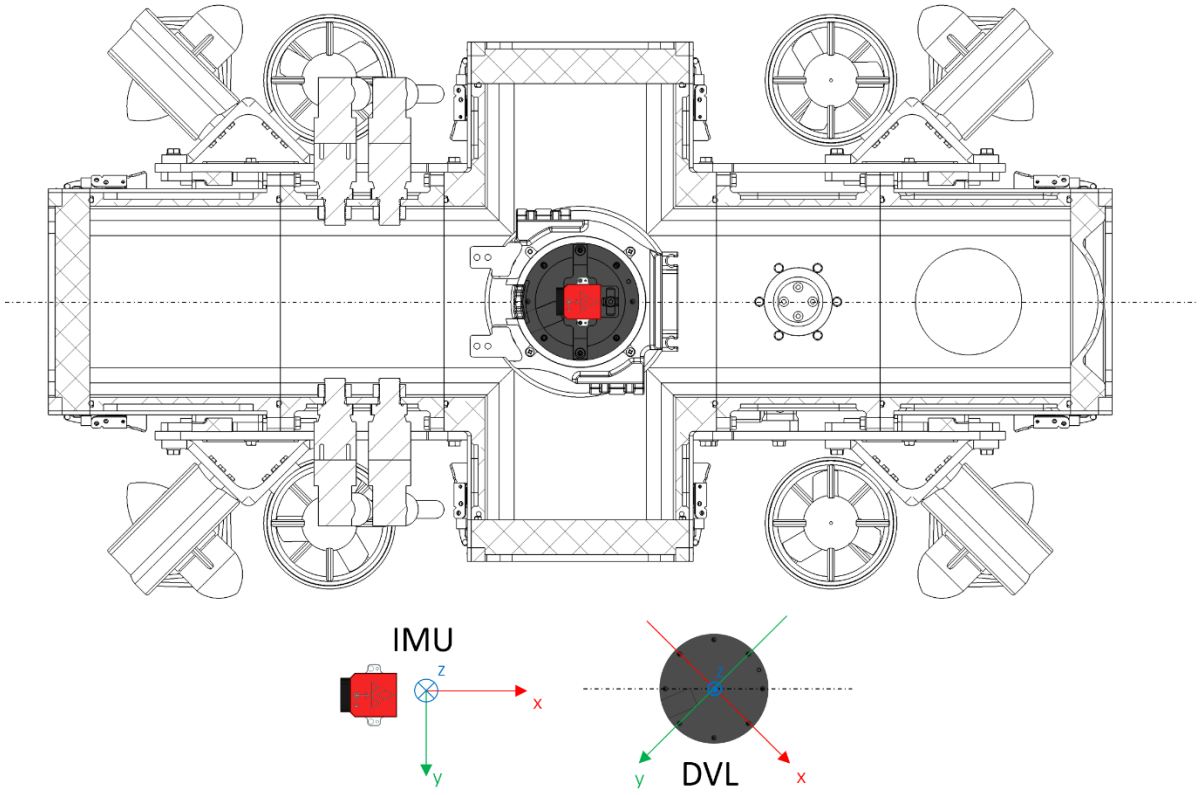


Figure 7: Emplacement des capteurs

Le DVL mesure la vitesse linéaire X Y Z. Aussi, avec la précision du capteur et les temps d'opération cours (environ 20 minutes), nous pouvons estimer une position en intégrant les vitesses sans accumuler trop d'erreurs. De plus, nous pouvons toujours corriger notre position rendue aux tâches avec nos caméras ou nos hydrophones. Notre IMU possède un filtre de karman intégré qui estime une orientation avec un quaternion. Il mesure aussi les vitesses angulaires avec son gyroscope. Nous verrons plus tard qu'avec ces informations, nous sommes en mesure de complètement décrire le sous-marin en 6 degrés de liberté.

Actionneurs

Depuis AUV7, S.O.N.I.A utilisé les moteurs T-200 de BlueRobotics, car ils sont abordables, petits et capables de développer une force considérable. Ces moteurs sont aussi reconnus pour être fiables et nous n'avons eu aucun problème avec ceux que nous possédons sur AUV7 depuis maintenant 5 ans. Les moteurs
3.3 T-200 sont utilisés par la quasi-totalité des équipes qui ont participé à RoboSub en 2019. Le moteur T-200 ne possède pas un ratio 50/50 ce qui signifie qu'il ne développe pas la même force quand il tourne sens horaire qu'antihoraire. BlueRobotics fournit les forces maximales que le moteur peut fournir avec une tension de 16V.

- 50 newtons horaires
- 40 newtons antihoraires

Les moteurs viennent donc avec une paire d'hélices qui nous permet de choisir la direction la force en fonction de la rotation. Toutefois, les moteurs à plein régime consomment une grande quantité d'énergie (25A à 16v). Les cartes d'alimentation sur le sous-marin ne sont pas en mesure de fournir cette quantité de courant. Il faudra prendre en compte la capacité de ces cartes pour borner la commande afin d'éviter qu'ils prennent feu. BlueRobotics fournit les forces développées par le moteur selon le signal PWM envoyé. Cela va nous permettre de développer un contrôleur en newton et de convertir la commande en PWM par la suite. Le graphique suivant présenté par BlueRobotics montre les courbes de forces selon la commande PWM et la tension d'opération.

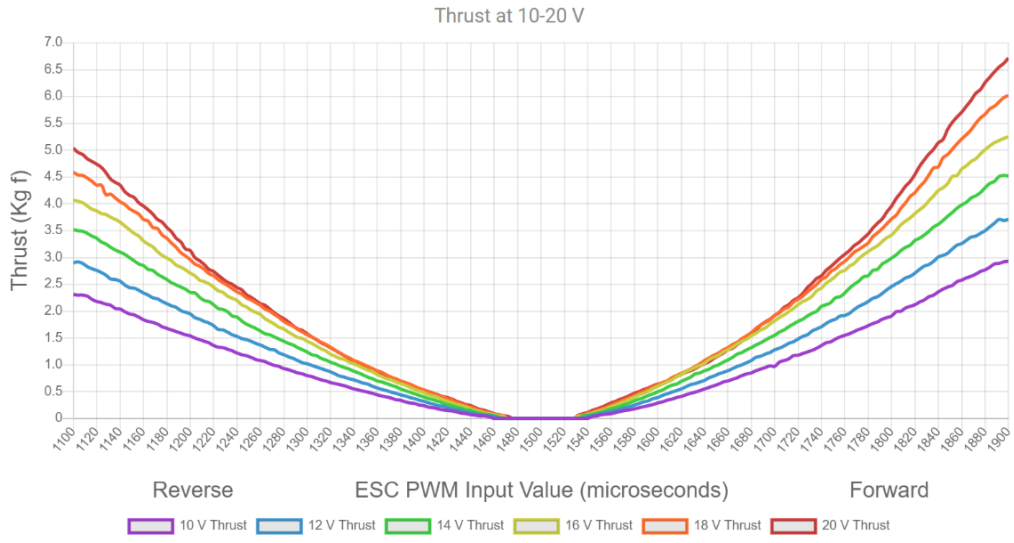


Figure 8: Force développée selon la commande PWM pour un moteur T-200 [5]

[5] Tirée du site de BlueRobotics (2019)

Pour être en mesure de bien contrôler le sous-marin en 6 dégrée de liberté, Il est important de répartir les moteurs ainsi que définir les hélices de façon stratégique pour être stable et efficace. Depuis AUV7, S.O.N.I.A utilise une configuration présentée par le site « ardusub » et nommée « vectored6dof ». Nous avons été très satisfaits de cette configuration que nous avons décidé de la réutiliser avec AUV8. La figure suivante montre la répartition des moteurs présentés par « ardusub »

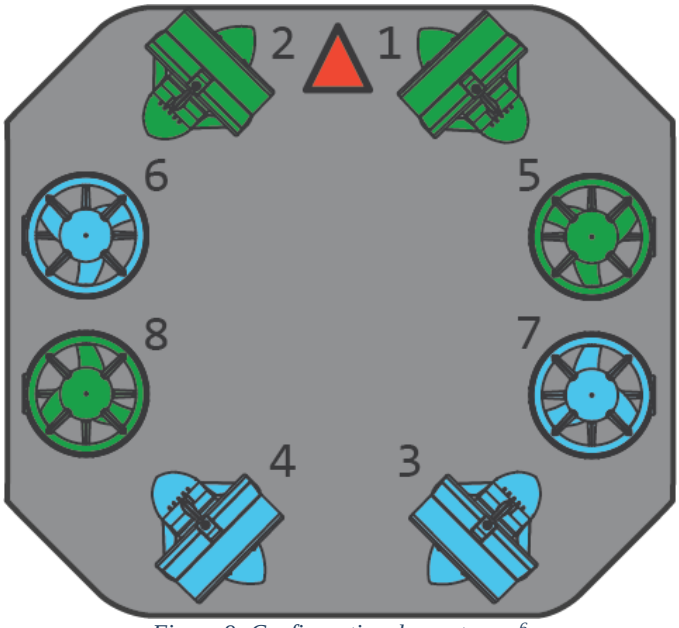


Figure 9: Configuration des moteurs. [6]

Où les moteurs en vert possèdent une hélice horaire et les moteurs en bleu possèdent une hélice antihoraire.

[6] Tirée du site de ArduSub

Considération mécanique pour faciliter l'asservissement

L'équipe mécanique considérée plusieurs éléments lors de la conception de AUV8 pour faciliter le contrôle du sous-marin. Dans un premier temps, nous voulons minimiser la masse et l'inertie du sous-marin pour réduire la force nécessaire à le déplacer. Pour réduire l'inertie, nous avons rapproché le plus possible
3.4 les composantes lourdes (batterie, ordinateur, DVL) du centre masse. Ces composantes se retrouvent toutes dans le « core ». Pour réduire la masse, nous n'avons pas le choix de réduire le volume pour que le sous-marin « ne flotte pas trop ». De plus, nous voulons aussi que la masse du volume d'eau déplacer soit légèrement supérieure à la masse du sous-marin. Cela a pour but de réduire la force des moteurs nécessaire à stabiliser le sous-marin sur l'axe des Z.

Deuxièmement, nous verrons dans la prochaine section, que de concevoir un sous-marin le plus symétrique possible nous permettra de réduire considérablement le nombre constant hydrodynamique à déterminer. AUV8 n'est pas parfaitement dans les 3 axes, mais il l'est suffisamment pour pouvoir l'assumer plus tard. La figure suivante montre les trois plans de symétrie du sous-marin.

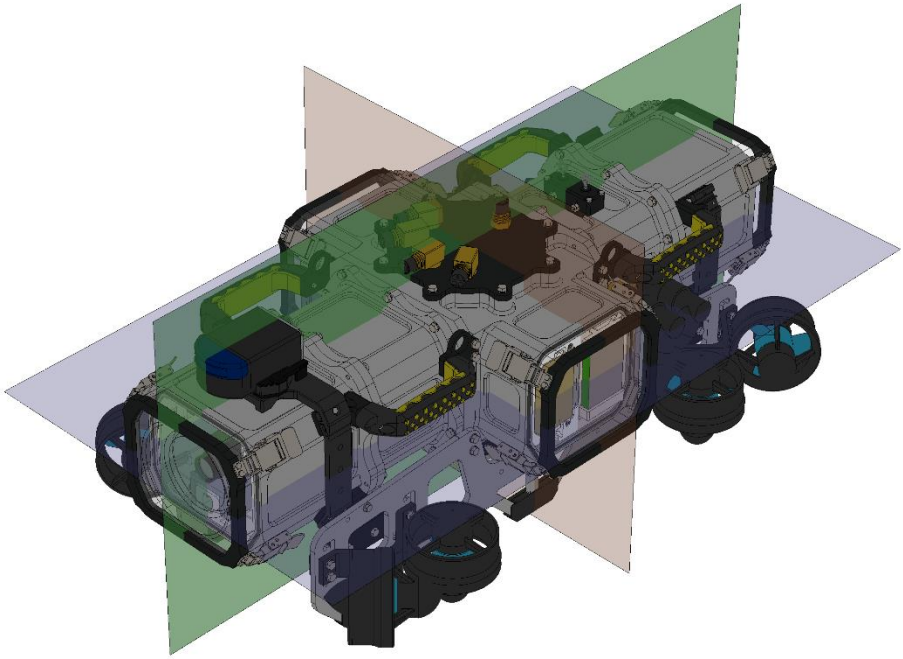


Figure 10: Plan de symétrie de AUV8

4 Identification du sous-marin

Dans cette section, nous discuterons de la modélisation mathématique de notre sous-marin. Ceci va nous permettre de concevoir notre contrôleur ainsi que de générer un modèle physique réaliste pour notre simulateur sur « Unity ». Le modèle physique est principalement basé sur le modèle de Fossen présenté dans le livre G*uidance and Control of Ocean Vehicules de* Thor I. Fossen (1994). Dans ce rapport, nous voulons présenter la matière de façon claire et intuitive afin de créer une lecture fluide et de nous familiariser avec les concepts. De plus, nous voulons que ce rapport soit un bon point de départ. Nous ne ferons pas de preuves et nous n'irons pas trop en détail dans les différents concepts. Si vous voulez approfondir vos connaissances, le livre de Thor I. Fossen sur lequel on s'est basé est disponible à la bibliothèque de l'ÉTS. Son code est « VK 543 F67 1994 ».

Les prochaines sous-sections vont se dérouler dans l'ordre suivant :

- Définition des hypothèses pour réduire la complexité du modèle.
- Définition des référentiels utilisés.
- Définition de la représentation mathématique de l'orientation du sous-marin.
- Déclaration des variables et des constantes utilisées.
- Construction de l'équation cinématique.
- Construction de l'équation dynamique.
- Construction du modèle d'états.

4.1 Hypothèses et suppositions

Afin de mieux nous guider dans la modélisation de notre sous-marin, nous avons faite trois suppositions afin d'alléger l'équation dynamique.

Premièrement, notre sous-marin possède une grande rigidité en raison de ses parois en aluminium qui possède une épaisseur allant jusqu'à 10mm. De plus, nos vitesses sont relativement faibles soit autour de 1 m/s. Nous pouvons donc supposer que les déformations dimensionnelles sous l'effet des forces externes sont négligeables. Ceci permet de définir notre sous-marin comme un corps rigide.

Deuxièmement, les dimensions du bassin de la compétition ne sont pas très grandes. Les déplacements que le sous-marin effectuera seront relativement faibles soit de 1 à 100m et donc négligeables par rapport à la courbure de la terre. Nous pouvons donc présumer que la terre est plate. Ceci va nous permettre d'utiliser un référentiel cartésien (X Y Z) et non le référentiel de la terre (longitude latitude altitude).

Troisièmement, nous supposons que les forces et moments hydrodynamiques sont complètement découplés. La négligence des effets de couplage va grandement simplifier l'identification des constantes du modèle physique. Dans le livre *Guidance and Control of Ocean Vehicules (section 2.4, p.37),* Thor I. Fossen (1994) explique que dans notre cas, nous pouvons négliger les forces de couplages, car notre sous-marin fonctionne à basse vitesse et qu'il possède 3 plans de symétrie. Dans la section précédente, nous avons vu que l'équipe mécanique a pris en compte ces affirmations et on conçut AUV8 le plus symétrique possible.

Référentiels

Dans le livre *Guidance and Control of Ocean Vehicules (section 2.1, p.6),* Thor I. Fossen (1994) suggère d'utiliser deux référentiels distincts lorsque nous voulons faire l'analyse dynamique d'un véhicule nautique en 6 degrés de liberté (DDL). La figure suivante montre bien les deux référentiels que nous allons **4.2** utiliser.

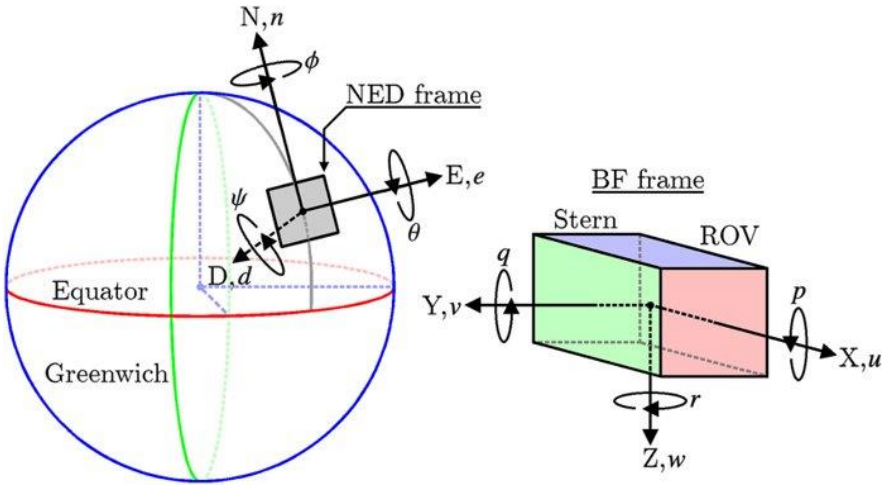


Figure 11 : Représentation des référentiels [7]

Vu que nous supposons que la terre est plate, nous pouvons créer un plan tangent sur la surface de la Terre selon notre position géographique et considérer ce plan pour créer un référentiel cartésien fixe. C'est pour cette raison qu'il est important de calibrer notre IMU lorsque nous allons en compétition. Nous pouvons aussi considérer ce référentiel comme étant inertiel. Un référentiel est inertiel si la somme des forces d'un objet qui se déplace à vitesse constante par rapport à ce référentiel est nulle. Il faut donc que la première loi de Newton soit respectée dans le référentiel pour qu'il soit considéré inertiel. En outre, nous avons décidé d'utiliser la convention « NED » pour « North East Down » de façon que les axes aient la configuration suivante :

- X+ : le nord
- Y+ : l'est
- Z+ : le bas

[7] Tirée de Path Generation for High-Performance Motion of ROVs Based on a Reference Model (2015, p. 89)

Le référentiel mobile couramment appelé BF pour « Body Fixed » ou référentiel objet est un référentiel qui suit le sous-marin durant ces déplacements. Pour les véhicules mobiles, il est recommandé de placer l'origine de référentiel au centre de masse et que les axes aient la configuration suivante:

- X : l'axe longitudinal
- Y : l'axe transversal
- Z : l'axe normal

Étant donné que notre référentiel fixe est considéré comme étant inertiel, il est possible de définir la dynamique du référentiel « BF » selon le référentiel fixe. Avec les suppositions que nous venons de définir, dans le livre *Guidance and Control of Ocean Vehicules,* Thor I. Fossen (1994) recommande donc de définir la position et l'orientation en fonction du référentiel inertiel ainsi que les vitesses linéaires et angulaires en fonction du référentiel « BF ».

4.3 Représentation de l'orientation dans l'espace

Pour représenter l'orientation du sous-marin, nous avons décidé d'opter pour le quaternion unitaire au lieu des angles d'Euler autant pour des raisons mathématiques que pour des raisons d'implémentation. Même si le quaternion n'est pas aussi intuitif que les angles d'Euler, il possède de gros avantages qui expliquent sa popularité. Afin de concorder avec Matlab, nous avons décidé d'utiliser la convention où la partie scalaire se trouve au début et que la partie imaginaire seconde. Le quaternion s'écrit donc de la façon suivante :

$$q = \eta + \varepsilon_1 \boldsymbol{i} + \varepsilon_2 \boldsymbol{j} + \varepsilon_1 \boldsymbol{k}$$

D'un point de vue mathématique, vu que le quaternion possède quatre paramètres au lieu de trois, il est en mesure de définir n'importes qu'elles rotation en évitant toute singularité. Ce phénomène est plus connu sous le nom de « Gimbal lock ». De plus, le quaternion ne contient pas de discontinuité. Avec les angles d'Euler, ce problème est plus connu sous « wrap around » et nous pouvons retrouver ce problème lorsqu'un angle approche $\pm\pi$ radian. La figure suivante montre ce phénomène.

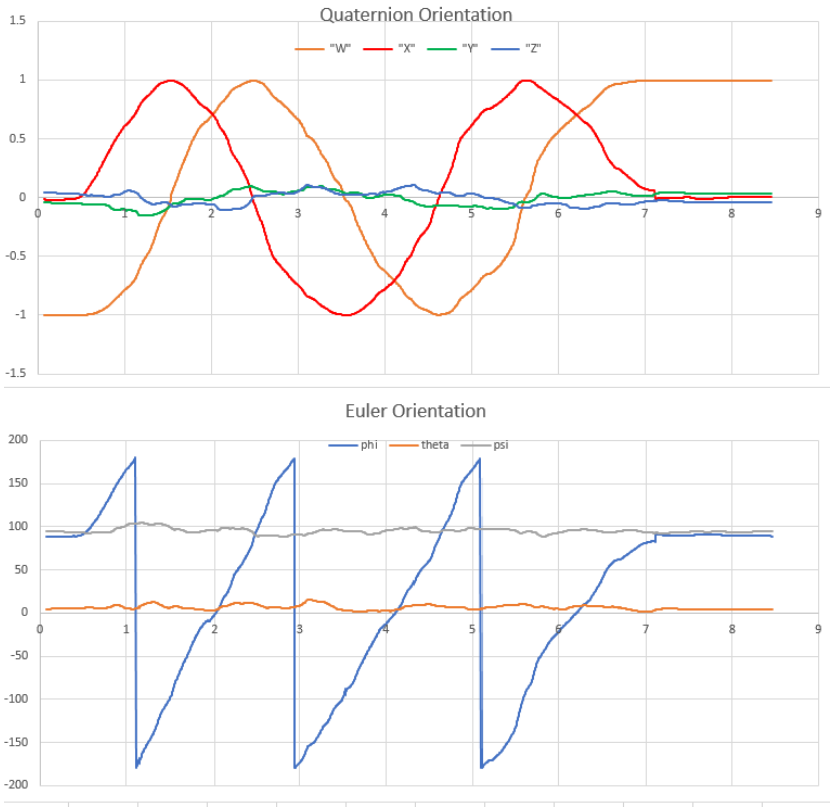


Figure 12 : Effet de discontinuité sur des angles d'Euler [8]

En ce qui concerne l'implémentation de notre modèle, nous avons avantage à utiliser le quaternion, car il est plus stable numériquement. Ceci explique sa grande popularité dans les applications numériques. La preuve, les logiciels que nous utilisons (ROS, Gazebo, Unity, etc.) utilisent le quaternion. Ceci est une raison supplémentaire pour l'utilisation de celui-ci. Cependant, définir le modèle physique en quaternion n'oblige pas que la commande doive l'utiliser aussi.

[8] Scheidler, tirée du blog ENDAQ (2018)

Définition des variables et constantes

Dans cette section, nous définirons les variables qui pourront être utiles pour la modélisation. Pour être le plus cohérents possible avec le livre *Guidance and Control of Ocean Vehicules,* Thor I. Fossen (1994), nous utiliserons la même notation que le livre, soit celle définie par SNAME (1950). Le tableau **4.4** suivant définit les variables liées au degré de liberté.

Tableau 1: Variables d'états

DDL		Référentiel Objet		Référentiel fixe
		Forces et moments	Vitesses linéaires et angulaires	Position et orientation (Euler)
1	Mouvement selon l'axe x (surge)	Y	u	x
2	Mouvement selon l'axe y (sway)	Y	v	y
3	Mouvement selon l'axe z (heave)	Z	w	z
4	Rotation selon l'axe *x* (roll)	K	p	ϕ
5	Rotation selon l'axe *y* (pitch)	M	q	θ
6	Rotation selon l'axe *z* (yaw)	N	r	ψ

Dans le cadre de ce projet, nous allons utiliser la représentation basée sur le quaternion. Nous pouvons le définir avec le vecteur suivant :

$$e = [\eta \;\; \varepsilon_1 \;\; \varepsilon_2 \;\; \varepsilon_3]^T$$

Nous pouvons définir le vecteur de commande pour nos 8 moteurs avec le vecteur suivant :

$$u = [U_1 \quad U_2 \quad U_3 \quad U_4 \quad U_5 \quad U_6 \quad U_7 \quad U_8]^T$$

Nous allons, par la suite, regrouper les variables sous la forme vectorielle. Nous pouvons définir le vecteur vitesse avec les vecteurs suivants :

$$v = [v_1^T, v_2^T]; \qquad v_1 = [u\ v\ w]^T; \qquad v_2 = [p\ q\ r]^T;$$

Nous pouvons définir le vecteur pose avec la représentation d'angle d'Euler avec les vecteurs suivants :

$$n_e = [n_1^T, n_2^T]; \qquad n_1 = [x, y, z]^T; \qquad n_2 = [\phi, \theta, \psi]^T;$$

Nous pouvons aussi définir le vecteur pose avec la représentation des quaternions de la façon suivante.

$$n_q = [n_1^T, e^T]$$

Nous allons définir les constantes mécaniques qui vont nous être utiles pour la modélisation du sous-marin. Le tableau suivant montre les constantes nécessaires pour bien décrire le corps rigide.

Tableau 2: Constante mécanique du sous-marin

Constantes	Description	Dimension	Unité
m	Masse	$\mathbb{R}^1$	kg
∇	Volume submergé	$\mathbb{R}^1$	m^3
H	Hauteur	$\mathbb{R}^1$	m
L	Longueur	$\mathbb{R}^1$	m
B	Largeur	$\mathbb{R}^1$	m
Rg	Centre de gravité	$\mathbb{R}^{3\times1}$	m
Rb	Centre de flottaison	$\mathbb{R}^{3\times1}$	m
Af	Air de surface transversale	$\mathbb{R}^{3\times1}$	m^2
I	Tenseur d'inertie	$\mathbb{R}^{3\times3}$	$kg\ m^3$
d_T	Distance du moteur selon Rg	$\mathbb{R}^{3\times1}$	m

Il reste qu'à définir les constantes hydrodynamiques de notre sous-marin. Étant donné qu'on néglige les effets de couplage pour les effets hydrodynamique, nous allons définir que les termes découplés. Le tableau suivant montre les constantes hydrodynamiques nécessaires pour notre modélisation

Tableau 3: Constantes hydrodynamiques du sous-marin.

	DDL 1	DDL 2	DDL 3	DDL 4	DDL 5	DDL 6
Masse ajouté	$X_{\dot{u}}$	$Y_{\dot{v}}$	$Z_{\dot{w}}$	$K_{\dot{p}}$	$M_{\dot{q}}$	$X_{\dot{r}}$
Constantes de drag linéaire	X_u	Y_v	Z_w	K_p	M_q	X_r
Constantes de drag quadratique	$X_{u\|u\|}$	$Y_{v\|v\|}$	$Z_{w\|w\|}$	$K_{p\|p\|}$	$M_{q\|q\|}$	$X_{r\|r\|}$

Équation cinématique du mouvement

Dans cette section, nous allons définir l'équation différentielle de la cinématique du sous-marin. Cette équation transforme les vitesses linéaires et angulaires du référentiel objet au référentiel inertiel. De plus, les vitesses angulaires seront transformées afin de pouvoir être intégrées dans leur représentation
4.5 respective. De cette façon, nous serons en mesure de déterminer la position et l'orientation du sous-marin par rapport au référentiel fixe. Cette section est importante, car tous les capteurs présents sur le sous-marin donnent les valeurs dans le référentiel objet. Il nous est donc impossible de mesurer les états du sous-marin directement à partir du référentiel fixe. La figure suivante montre la relation entre les variables liées au référentiel objet et ceux du référentiel fixe.

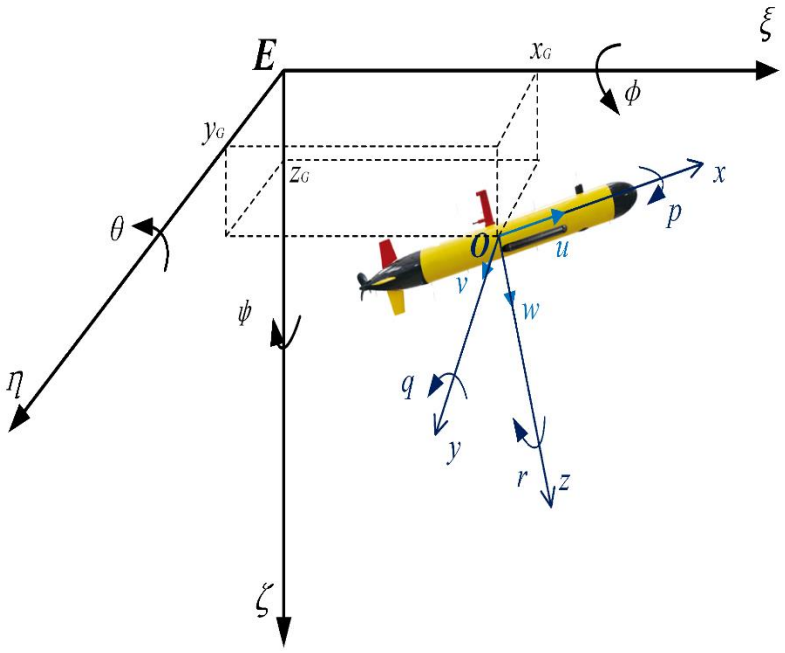


Figure 13: Représentation du référentiel objet dans le référentiel fixe [9]

Nous commencerons par définir l'équation différentielle cinématique avec la représentation d'angle d'Euler, car la plupart des concepts utilisés sont abordés dans le cours GPA546. Il sera plus facile de comprendre l'idée derrière cette formule avant de construire la même équation cinématique, mais en utilisant les quaternions.

[9] Tirée de 3DOF Adaptive Line-Of-Sight Based Proportional Guidance Law for Path Following of AUV in the Presence of Ocean Currents (2019, p.3)

Équation cinématique du mouvement représenté avec les angles d'Euler

Dans cette section, nous définirons une équation cinématique utilisant les angles d'Euler. Nous allons utiliser l'ordre de rotation ZYX. Nous pouvons exprimer cette équation différentielle sous la forme matricielle suivante.

4.5.1

$$\dot{n}_e = J(n_e)v \Leftrightarrow \begin{bmatrix} \dot{n}_1 \\ \dot{n}_2 \end{bmatrix} = \begin{bmatrix} J_1(n_2) & 0_{3x3} \\ 0_{3x3} & J_2(n_2) \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \in \mathbb{R}^6$$

Afin d'alléger l'écriture des matrices, nous allons utiliser la notation « c » pour le cosinus, « s » pour le sinus et « t » pour la tangente.

Transformation de la vitesse linéaire avec Euler

Dans cette section, nous allons déterminer la transformation de la vitesse linéaire, soit la matrice J_1. Dans un premier temps, il faut définir les matrices de rotation de base pour chaque axe de rotation.

$$R_{x,\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\phi) & s(\phi) \\ 0 & -s(\phi) & c(\phi) \end{bmatrix} \quad R_{y,\theta} = \begin{bmatrix} c(\theta) & 0 & -s(\theta) \\ 0 & 1 & 0 \\ s(\theta) & 0 & c(\theta) \end{bmatrix} \quad R_{z,\psi} = \begin{bmatrix} c(\psi) & s(\psi) & 0 \\ -s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Maintenant, si nous voulons la matrice de rotation qui suit l'ordre ZYX, il suffit de multiplier les transposés des matrices de rotation de base dans le même ordre. Il faut se rappeler que le produit des matrices de rotation n'est pas commutatif. L'ordre est donc très important. Cela donne l'équation suivante :

$$J_1(n_2) = {R_{z,\psi}}^T {R_{y,\theta}}^T {R_{x,\phi}}^T$$

Nous pouvons développer cette équation, ce qui nous donne:

$$J_1(n_2) = \begin{bmatrix} c(\psi)\,c(\theta) & c(\psi)\,s(\phi)\,s(\theta) - c(\phi)\,s(\psi) & s(\phi)\,s(\psi) + c(\phi)\,c(\psi)\,s(\theta) \\ c(\theta)\,s(\psi) & c(\phi)\,c(\psi) + s(\phi)\,s(\psi)\,s(\theta) & c(\phi)\,s(\psi)\,s(\theta) - c(\psi)\,s(\phi) \\ -s(\theta) & c(\theta)\,s(\phi) & c(\phi)\,c(\theta) \end{bmatrix}$$

Transformation de la vitesse angulaire avec Euler

Dans cette section, nous définirons une relation pour transformer les vitesses angulaires du référentiel objet $v_2 = [p, q, r]^T$ en vecteur de rotation instantanée $\dot{n}_2 = [\dot{\phi}, \dot{\theta}, \dot{\psi}]^T$. Nous pouvons définir une matrice de transformation J_2 pour nous permettre cette transformation. Cela donne l'équation suivante :

$$\dot{n}_2 = J_2(n_2)\, v_2$$

Comparativement à des vitesses linéaires, nous ne pouvons pas directement intégrer les vitesses angulaires afin $v_2 = [p, q, r]^T$ d'obtenir son orientation. Cependant, il est possible d'intégrer un vecteur de rotation instantanée $\dot{n}_2 = [\dot{\phi}, \dot{\theta}, \dot{\psi}]^T$ afin d'obtenir l'orientation du sous-marin dans le référentiel inertiel. Pour effectuer cette conversion, nous pouvons utiliser la relation suivante.

$$v_2 = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + R_{x,\phi} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R_{x,\phi} R_{y,\theta} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} = J_2^{-1}(n_2)\dot{n}_2$$

Nous pouvons développer cette relation afin de déterminer la matrice J_2^{-1} et l'inverser pour avoir la matrice J_2.

$$J_2^{-1} = \begin{bmatrix} 1 & 0 & -s(\theta) \\ 0 & c(\phi) & c(\theta)\, s(\phi) \\ 0 & -s(\phi) & c(\phi)\, c(\theta) \end{bmatrix} \quad \Rightarrow \quad J_2 = \begin{bmatrix} 1 & \dfrac{s(\phi)\, s(\theta)}{c(\theta)} & \dfrac{c(\phi)\, s(\theta)}{c(\theta)} \\ 0 & c(\phi) & -s(\phi) \\ 0 & \dfrac{s(\phi)}{c(\theta)} & \dfrac{c(\phi)}{c(\theta)} \end{bmatrix}$$

Matrice de transformation avec la représentation d'Euler

Nous avons donc tous éléments nécessaires pour construire notre matrice J. Nous pouvons la développer, ce qui donne le résultat suivant.

$$J = \begin{bmatrix} c(\psi)\,c(\theta) & c(\psi)\,s(\phi)\,s(\theta) - c(\phi)\,s(\psi) & s(\phi)\,s(\psi) + c(\phi)\,c(\psi)\,s(\theta) & 0 & 0 & 0 \\ c(\theta)\,s(\psi) & c(\phi)\,c(\psi) + s(\phi)\,s(\psi)\,s(\theta) & c(\phi)\,s(\psi)\,s(\theta) - c(\psi)\,s(\phi) & 0 & 0 & 0 \\ -s(\theta) & c(\theta)\,s(\phi) & c(\phi)\,c(\theta) & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \frac{s(\phi)\,s(\theta)}{c(\theta)} & \frac{c(\phi)\,s(\theta)}{c(\theta)} \\ 0 & 0 & 0 & 0 & c(\phi) & -s(\phi) \\ 0 & 0 & 0 & 0 & \frac{s(\phi)}{c(\theta)} & \frac{c(\phi)}{c(\theta)} \end{bmatrix}$$

Nous pouvons remarquer que cette matrice n'est pas définie pour l'ensemble des angles $\theta \in [-\pi, \pi]$. En effet, nous pouvons constater que nous avons des divisions par 0 lorsque l'angle θ (pitch) vaut $\pm\frac{\pi}{2}$ radians. Ceci crée alors des zones de singularités. Ceci n'est pas une surprise, car toutes les représentations d'orientation à trois paramètres contiennent des zones singulières. Cependant, la plupart des applications maritimes ne travaillent pas autour des zones de singularités que nous avons déterminées. Il est donc possible d'utiliser cette représentation dans notre application.

Équation cinématique représentée avec le quaternion unitaire

Dans cette section nous allons définir l'équation cinématique basée sur le quaternion. L'équation différentielle va avoir la même forme que pour les angles d'Euler, mais la matrice ne sera plus carrée, car le quaternion contient quatre paramètres. Cela donne l'équation suivante :

4.5.2

$$\dot{n}_q = E(n_q)v \Leftrightarrow \begin{bmatrix}\dot{n}_1\\ \dot{e}\end{bmatrix} = \begin{bmatrix}E_1(e) & 0_{3x3}\\ 0_{4x3} & E_2(e)\end{bmatrix}\begin{bmatrix}v_1\\ v_2\end{bmatrix} \in \mathbb{R}^7$$

Transformation de la vitesse linéaire avec le quaternion

Dans cette section, nous allons déterminer la transformation de la vitesse linéaire. Cette équation prend la forme suivante:

$$\dot{n} = E_1(e)\, v_1$$

Nous pouvons transformer un vecteur d'un référentiel à un autre en fonction d'un quaternion décrivant la transformation entre les deux référentiels en utilisant la relation suivante.

$$E_1 = I_{3X3} + 2\eta \boldsymbol{S}(\varepsilon) + 2\boldsymbol{S}^2(\varepsilon)$$

Où I_{3X3} est une matrice identitaire et **S** est une matrice antisymétrique $\mathbb{R}^{3x3}$. Nous pouvons développer la relation précédente pour l'avoir sous la forme matricielle.

$$E_1 = \begin{bmatrix} -2\,{\varepsilon_2}^2 - 2\,{\varepsilon_3}^2 + 1 & 2\,\varepsilon_1\,\varepsilon_2 - 2\,\varepsilon_3\,\eta & 2\,\varepsilon_1\,\varepsilon_3 + 2\,\varepsilon_2\,\eta \\ 2\,\varepsilon_1\,\varepsilon_2 + 2\,\varepsilon_3\,\eta & -2\,{\varepsilon_1}^2 - 2\,{\varepsilon_3}^2 + 1 & 2\,\varepsilon_2\,\varepsilon_3 - 2\,\varepsilon_1\,\eta \\ 2\,\varepsilon_1\,\varepsilon_3 - 2\,\varepsilon_2\,\eta & 2\,\varepsilon_2\,\varepsilon_3 + 2\,\varepsilon_1\,\eta & -2\,{\varepsilon_1}^2 - 2\,{\varepsilon_2}^2 + 1 \end{bmatrix}$$

Transformation de la vitesse angulaire avec le quaternion

Dans cette section, nous allons définir l'équation de la transformation de la vitesse angulaire. Cette équation prend la forme suivante :

$$\dot{e} = E_2(e)\, v_2$$

De la même façon que pour les angles d'Euler, nous devons transformer les vitesses angulaires du référentiel objet $v_2 = [p, q, r]^T$ en quaternion instantané $\dot{e} = [\dot{\eta}\, \dot{\varepsilon_1}\dot{\varepsilon_2}\dot{\varepsilon_3}]^T$ afin d'être en mesure de l'intégrer dans le référentiel fixe. Pour effectuer cette conversion, nous pouvons utiliser la relation suivante.

$$E_2 = \frac{1}{2}\begin{bmatrix} -\varepsilon_1 & -\varepsilon_2 & -\varepsilon_3 \\ \eta & -\varepsilon_3 & \varepsilon_2 \\ \varepsilon_3 & \eta & -\varepsilon_1 \\ -\varepsilon_2 & \varepsilon_1 & \eta \end{bmatrix}$$

Matrice de transformation avec la représentation du quaternion

Nous avons donc tous éléments nécessaires pour construire notre matrice E. Nous pouvons la développer, ce qui donne le résultat suivant.

$$E = \begin{bmatrix} -2\,{\varepsilon_2}^2 - 2\,{\varepsilon_3}^2 + 1 & 2\,\varepsilon_1\,\varepsilon_2 - 2\,\varepsilon_3\,\eta & 2\,\varepsilon_1\,\varepsilon_3 + 2\,\varepsilon_2\,\eta & 0 & 0 & 0 \\ 2\,\varepsilon_1\,\varepsilon_2 + 2\,\varepsilon_3\,\eta & -2\,{\varepsilon_1}^2 - 2\,{\varepsilon_3}^2 + 1 & 2\,\varepsilon_2\,\varepsilon_3 - 2\,\varepsilon_1\,\eta & 0 & 0 & 0 \\ 2\,\varepsilon_1\,\varepsilon_3 - 2\,\varepsilon_2\,\eta & 2\,\varepsilon_2\,\varepsilon_3 + 2\,\varepsilon_1\,\eta & -2\,{\varepsilon_1}^2 - 2\,{\varepsilon_2}^2 + 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{\varepsilon_1}{2} & -\frac{\varepsilon_2}{2} & -\frac{\varepsilon_3}{2} \\ 0 & 0 & 0 & \frac{\eta}{2} & -\frac{\varepsilon_3}{2} & \frac{\varepsilon_2}{2} \\ 0 & 0 & 0 & \frac{\varepsilon_3}{2} & \frac{\eta}{2} & -\frac{\varepsilon_1}{2} \\ 0 & 0 & 0 & -\frac{\varepsilon_2}{2} & \frac{\varepsilon_1}{2} & \frac{\eta}{2} \end{bmatrix}$$

Nous pouvons remarquer que cette matrice est définie pour l'ensemble des valeurs $[\eta, \varepsilon_1, \varepsilon_2, \varepsilon_3] \in [-1,1]$. Par conséquent, la matrice de transformation E ne contient aucune zone de singularité. Cependant nous rajoutons un quatrième paramètre. Nous pouvons aussi constater que la Matrice E ne contient que des multiplications et des additions, ce qui est plus stable numériquement que des fonctions trigonométriques. De plus, nous pouvons remarquer que la matrice E_2 est plus linéaire que la matrice J_2.

4.6 Normalisation du quaternion

Il ne faut pas oublier qu'une rotation est considérée pure (que la transformation ne contient aucune déformation) seulement si le quaternion est unitaire. Lorsque nous implémentons le quaternion dans des contextes réels, plusieurs phénomènes comme le bruit ou les erreurs d'arrondissement numérique font en sorte que le quaternion ne soit plus unitaire. Il important de normaliser le quaternion provenant de la IMU et celui intégré par le modèle physique afin de respecter la contrainte du quaternion unitaire, soit :

$$e^T e = \eta^2 + \varepsilon_1^2 + \varepsilon_2^2 + \varepsilon_3^2 = 1$$

Afin de normaliser un quaternion, nous pouvons utiliser la relation suivante.

$$e_{\text{normalisé}} = \frac{e}{\|e\|}$$

L'erreur sur courte période peut être négligeable, mais devient plus grande avec le temps.

Équation dynamique d'un corps rigide

Dans cette section, nous allons développer l'équation différentielle dynamique du sous-marin. Nous avons décidé d'utiliser la représentation de Newton, car elle est intuitive et bien connue par tous les étudiants de l'ÉTS. Étant donné que nos deux référentiels sont cartésiens, il sera facile d'utiliser la représentation de
4.7 vectoriel. De plus, la représentation de Newton est compatible avec une orientation décrite en quaternion. Nous n'avons pas avantage à utiliser une approche énergétique comme celle de Lagrange.

Le modèle de Fossen que nous avons utilisé dans ce projet est basé sur la formulation de Newton-Euler. Cette représentation décrit la dynamique en 6 dégrée de liberté (soit en translation et en rotation) sous forme matricielle d'un corps rigide dans l'espace. Il nous est possible d'utiliser cette relation, car nous avons supposé que notre sous-marin soit corps rigide. De plus, vu que la plupart des forces sont décrites selon le référentiel objet, l'équation dynamique sera donc exprimée selon ce référentiel. Le modèle de Fossen peut donc être écrit de la façon suivante.

$$M\dot{v} + C(v)v + D(v)v + g(n) = \tau$$

Où :

M : La matrice d'inertie du sous-marin (incluant la masse ajoutée) $\mathbb{R}^{6x6}$.

C : La matrice Coriolis $\mathbb{R}^{6x6}$.

D : Matrice d'amortissement $\mathbb{R}^{6x6}$.

g : Vecteur gravité $\mathbb{R}^{6x1}$.

τ : Vecteur de la commande $\mathbb{R}^{6x1}$.

Cette équation va nous être utiles afin de développer un algorithme de contrôle. Nous allons aussi définir une autre équation qui sera utiliser pour notre simulateur. Nous allons reprendre l'équation précédente en rajoutant des perturbations externes. Cela donne la relation suivante :

$$M\dot{v} + C(v)v + D(v)v + g_b(n) = \tau_E + \tau$$

Où :

g_b: Vecteur gravité borné $\mathbb{R}^{6x1}$.

τ_E : Vecteur perturbation causé par l'environnement $\mathbb{R}^{6x1}$.

Nous allons discuter de chaque élément de l'équation dans les prochaines sous-sections.

Matrice d'inertie

Dans cette section, nous allons définir la matrice M de l'équation dynamique. Dans notre contexte, la masse du fluide ne peut pas être négligée. La masse du système sera donc la masse du sous-marin plus la masse virtuelle de l'eau déplacée. La figure suivante montre la répartition de la masse ajoutée qui entoure le sous-marin lorsqu'il se déplace.

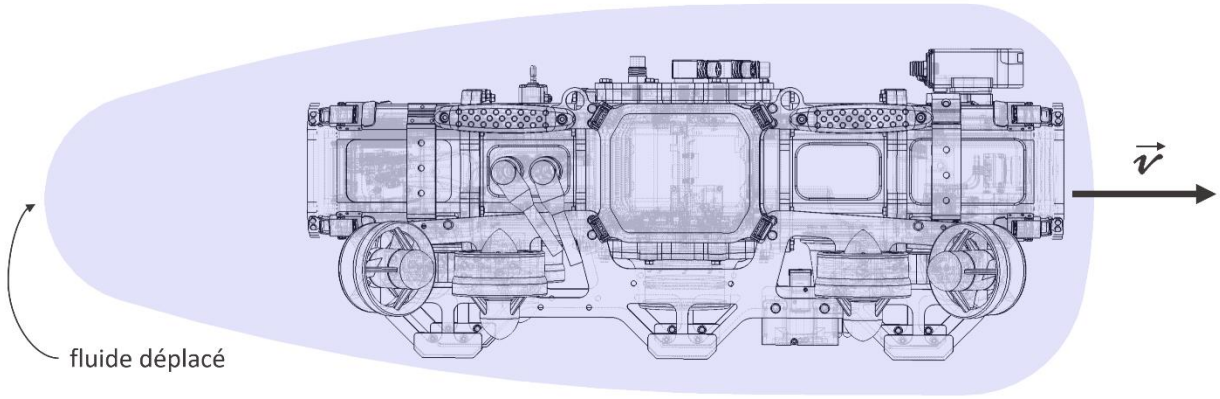


Figure 14: Répartition de la masse ajoutée autour du sous-marin

Dans le livre *Guidance and Control of Ocean Vehicules* (section 2.4.1 p.32), Thor I. Fossen (1994) explique que si nous voulons être rigoureux nous ne pourrions pas définir le système comme une masse plus lourde puisque ce concept est faux. Cependant, vu que nous négligeons plusieurs paramètres, cette affirmation devient acceptable. Cette matrice se trouve à être la somme de la matrice d'inertie du corps rigide et de la matrice d'inertie de la masse ajoutée.

$$M = M_{\mathrm{RB}} + M_A$$

Où:

M_{RB}: La matrice d'inertie du corps rigide.

M_A: la matrice des masses ajoutée crée par les forces hydrodynamiques.

Dans le livre *Guidance and Control of Ocean Vehicules*, Thor I. Fossen (1994) suggère de séparer les deux matrices, car les constantes de la matrice d'inertie du corps rigide sont déterminées en utilisant la représentation de Newton tandis que celles de la matrice de masse ajoutée est déterminé avec une approche énergétique.

La matrice M_{RB} peut-être paramétrée avec la forme suivante.

$$M_{\mathrm{RB}} = \begin{bmatrix} mI_{3\mathrm{x}3} & -\mathrm{mS(rg)} \\ \mathrm{mS(rg)} & I \end{bmatrix} = \begin{bmatrix} m & 0 & 0 & 0 & \mathrm{RG}_z\, m & -\mathrm{RG}_y\, m \\ 0 & m & 0 & -\mathrm{RG}_z\, m & 0 & \mathrm{RG}_x\, m \\ 0 & 0 & m & \mathrm{RG}_y\, m & -\mathrm{RG}_x\, m & 0 \\ 0 & -\mathrm{RG}_z\, m & \mathrm{RG}_y\, m & I_{x,x} & I_{x,y} & I_{x,z} \\ \mathrm{RG}_z\, m & 0 & -\mathrm{RG}_x\, m & I_{y,x} & I_{y,y} & I_{y,z} \\ -\mathrm{RG}_y\, m & \mathrm{RG}_x\, m & 0 & I_{z,x} & I_{z,y} & I_{z,z} \end{bmatrix}$$

Où :

m : masse du sous-marin

RG : centre de masse $\mathbb{R}^{3\mathrm{x}3}$.

I : tenseur d'inertie $\mathbb{R}^{3\mathrm{x}3}$.

S : matrice antisymétrique $\mathbb{R}^{3\mathrm{x}3}$.

Dans le livre *Guidance and Control of Ocean Véhicules* (section 2.4.1 p.32), Thor I. Fossen (1994) explique que la masse ajoutée est une masse virtuelle. Cette masse représente une pression proportionnelle à l'accélération qui est appliquée sur le sous-marin. La matrice de masse ajoutée peut être écrite avec la matrice suivante.

$$M_A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = -\begin{bmatrix} X_{\dot{u}} & X_{\dot{v}} & X_{\dot{w}} & X_{\dot{p}} & X_{\dot{q}} & X_{\dot{r}} \\ Y_{\dot{u}} & Y_{\dot{v}} & Y_{\dot{w}} & Y_{\dot{p}} & Y_{\dot{q}} & Y_{\dot{r}} \\ Z_{\dot{u}} & Z_{\dot{v}} & Z_{\dot{w}} & Z_{\dot{p}} & Z_{\dot{q}} & Z_{\dot{r}} \\ K_{\dot{u}} & K_{\dot{v}} & K_{\dot{w}} & K_{\dot{p}} & K_{\dot{q}} & K_{\dot{r}} \\ M_{\dot{u}} & M_{\dot{v}} & M_{\dot{w}} & M_{\dot{p}} & M_{\dot{q}} & M_{\dot{r}} \\ X_{\dot{u}} & X_{\dot{v}} & X_{\dot{w}} & X_{\dot{p}} & X_{\dot{q}} & X_{\dot{r}} \end{bmatrix}$$

Cette matrice utilise la notation SNAME (1950). Par exemple, $Y_{\dot{u}}$ représente la masse sur l'axe Y pour une accélération sur l'axe x. Nous allons estimer des paramètres dans une section suivante.

Comme mentionné plus tôt, nous négligeons les effets de couplage sur les effets hydrodynamiques, car nous pouvons supposer que le sous-marin à 3 plans de symétrie ainsi qu'il se déplace à basse vitesse. Ceci revient à dire que les valeurs des éléments non diagonaux sont négligeables par rapport aux valeurs des éléments diagonaux. Nous pouvons donc simplifier la matrice sous la forme suivante.

$$M_A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = -\begin{bmatrix} X_{\dot{u}} & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_{\dot{v}} & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_{\dot{w}} & 0 & 0 & 0 \\ 0 & 0 & 0 & K_{\dot{p}} & 0 & 0 \\ 0 & 0 & 0 & 0 & M_{\dot{q}} & 0 \\ 0 & 0 & 0 & 0 & 0 & X_{\dot{r}} \end{bmatrix}$$

Matrice de Coriolis

4.9 Prenons par exemple que vous regardez le soleil et que vous êtes un observateur qui la regarde depuis la terre. Avec le temps vous auriez l'impression que le soleil se déplace dans le ciel. En réalité, c'est vous qui tournez, car vous êtes dans un référentiel en rotation. Cela s'explique, car la terre tourne sur elle-même. Nous appelons cela l'effet de Coriolis en nom au mathématicien qui a quantifié ce phénomène. Les forces de Coriolis apparaissent lorsque les lois de Newton sont transformées dans un référentiel en rotation. Dans notre cas, le référentiel inertiel qu'on suppose fixe est en réalité un référentiel rotatif, car il tourne avec la terre. Les forces de Coriolis sont dites « fictives », car elles sont la résultante de plusieurs forces, comme la force gravitationnelle et centripète. La figure suivante représente les forces de Coriolis sur un système en rotation.

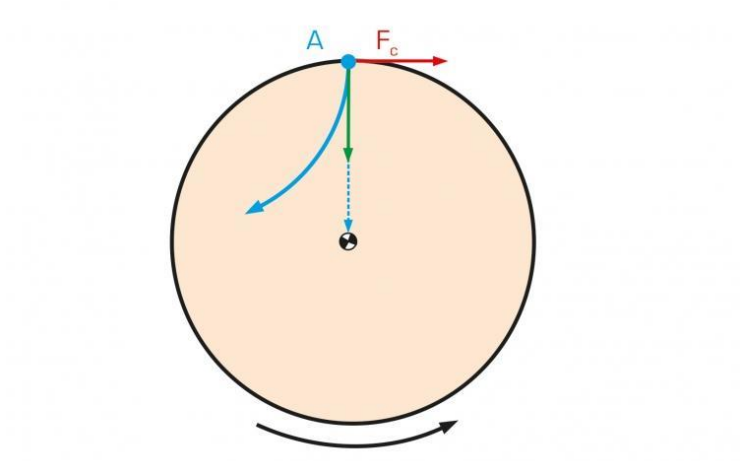


Figure 15: Représentation visuelle de l'effet de Coriolis [10]

[10] Tirée du site de G.U.N.T

Les forces de Coriolis sont appliquées perpendiculairement à la vitesse du corps en mouvement et sont proportionnelles à la vitesse et la masse de celui-ci. Nous pouvons donc déterminer les forces de Coriolis pour le corps rigide ainsi que pour la masse ajoutée de la façon suivante.

$$C(v) = C_{\mathrm{RB}}(v) + C_A(v)$$

Pour un corps rigide, les forces de Coriolis peuvent être paramétrées avec la forme antisymétrique suivante.

$$C_{\mathrm{RB}}(v) = \begin{bmatrix} 0_{3\times3} & -m\,\boldsymbol{S}(v_1) - m\,\boldsymbol{S}(v_1)\boldsymbol{S}(r_G) \\ -m\,\boldsymbol{S}(v_1) + m\,\boldsymbol{S}(v_1)\boldsymbol{S}(r_G) & -\boldsymbol{S}(Iv_2) \end{bmatrix}$$

Où:

m : masse du sous-marin

RG : centre de masse. $\mathbb{R}^{3x1}$

I : tenseur d'inertie $\mathbb{R}^{3x3}$

S : matrice anti- symétrique

Nous pouvons développer la matrice, ce qui nous donne la forme suivante.

$$C_{\mathrm{RB}}(v) = \begin{bmatrix} 0 & 0 & 0 & m\,(\mathrm{RG}_2\,q + \mathrm{RG}_3\,r) & m\,(w - \mathrm{RG}_1\,q) & -m\,(v + \mathrm{RG}_1\,r) \\ 0 & 0 & 0 & -m\,(w + \mathrm{RG}_2\,p) & m\,(\mathrm{RG}_1\,p + \mathrm{RG}_3\,r) & m\,(u - \mathrm{RG}_2\,r) \\ 0 & 0 & 0 & m\,(v - \mathrm{RG}_3\,p) & -m\,(u + \mathrm{RG}_3\,q) & m\,(\mathrm{RG}_1\,p + \mathrm{RG}_2\,q) \\ -m\,(\mathrm{RG}_2\,q + \mathrm{RG}_3\,r) & m\,(w + \mathrm{RG}_2\,p) & -m\,(v - \mathrm{RG}_3\,p) & 0 & I_{3,1}\,p + I_{3,2}\,q + I_{3,3}\,r & -I_{2,1}\,p - I_{2,2}\,q - I_{2,3}\,r \\ -m\,(w - \mathrm{RG}_1\,q) & -m\,(\mathrm{RG}_1\,p + \mathrm{RG}_3\,r) & m\,(u + \mathrm{RG}_3\,q) & -I_{3,1}\,p - I_{3,2}\,q - I_{3,3}\,r & 0 & I_{1,1}\,p + I_{1,2}\,q + I_{1,3}\,r \\ m\,(v + \mathrm{RG}_1\,r) & -m\,(u - \mathrm{RG}_2\,r) & -m\,(\mathrm{RG}_1\,p + \mathrm{RG}_2\,q) & I_{2,1}\,p + I_{2,2}\,q + I_{2,3}\,r & -I_{1,1}\,p - I_{1,2}\,q - I_{1,3}\,r & 0 \end{bmatrix}$$

Pour corps rigide qui se déplace dans un fluide idéal, les forces de Coriolis hydrodynamique peuvent être paramétrées avec la forme antisymétrique suivante.

$$C_A(v) = \begin{bmatrix} 0_{3\times3} & -\boldsymbol{S}(A_{11}v_1 + A_{11}v_2) \\ -\boldsymbol{S}(A_{11}v_1 + A_{11}v_2) & -\boldsymbol{S}(A_{21}v_1 + A_{22}v_2) \end{bmatrix}$$

Où:

A_{ij} : Sous matrice de la masse ajoutée.

S : matrice antisymétrique.

Nous pouvons développer la matrice, ce qui nous donne la forme suivante.

$$C_A(v) = \begin{bmatrix} 0 & 0 & 0 & 0 & -Z_{\dot{w}}\dot{w} & Y_{\dot{v}}\dot{v} \\ 0 & 0 & 0 & Z_{\dot{w}}\dot{w} & 0 & -X_{\dot{u}}\dot{u} \\ 0 & 0 & 0 & -Y_{\dot{v}}\dot{v} & X_{\dot{u}}\dot{u} & 0 \\ 0 & -Z_{\dot{w}}\dot{w} & Y_{\dot{v}}\dot{v} & 0 & -N_{\dot{r}}\dot{r} & M_{\dot{q}}\dot{q} \\ Z_{\dot{w}}\dot{w} & 0 & -X_{\dot{u}}\dot{u} & N_{\dot{r}}\dot{r} & 0 & -K_{\dot{p}}\dot{p} \\ -Y_{\dot{v}}\dot{v} & X_{\dot{u}}\dot{u} & 0 & -M_{\dot{q}}\dot{q} & K_{\dot{p}}\dot{p} & 0 \end{bmatrix}$$

Matrice d'amortissement

4.10 Dans cette section, nous allons construire la matrice qui représente l'amortissement qu'applique l'eau sur le sous-marin. De la même façon que pour la masse ajoutée, nous pouvons négliger les effets de couplage sur les effets hydrodynamique, car nous allons à basse vitesse et que nous pouvons assumer que notre sous-marin a trois plans de symétrie. Aussi, dans le livre *Guidance and Control of Ocean Vehicules* (section 2.4.2 p.43) Thor I. Fossen (1994) explique que tous les termes de degré supérieur à 2 sont négligeables. Il explique aussi que les forces d'amortissement peuvent être superposées en terme quadratique et linéaire. La matrice D peut être définie de la fonction suivante:

$$D(n) = D_l(v) + D_q(v)$$

Où:

D_l : la matrice des termes linéaire.

D_q : la matrice des termes quadratique.

La matrice des termes linéaire peut être écrite de la façon suivante.

$$D_l(v) = \begin{bmatrix} X_u & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_v & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_w & 0 & 0 & 0 \\ 0 & 0 & 0 & K_p & 0 & 0 \\ 0 & 0 & 0 & 0 & M_q & 0 \\ 0 & 0 & 0 & 0 & 0 & X_r \end{bmatrix}$$

La matrice des termes quadratique peut être écrite de la façon suivante.

$$D_q = \begin{bmatrix} X_{u|u|}|u| & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_{v|v|}|v| & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_{w|w|}|w| & 0 & 0 & 0 \\ 0 & 0 & 0 & K_{p|p|}|p| & 0 & 0 \\ 0 & 0 & 0 & 0 & M_{q|q|}|q| & 0 \\ 0 & 0 & 0 & 0 & 0 & X_{r|r|}|r| \end{bmatrix}$$

La matrice des termes quadratique peut être écrite de la façon suivante.

$$X_{u|u|} = \frac{1}{2}\rho C_D A$$

Où:

ρ : Densité de l'eau

C_D : constante d'amortissement (drag)

A : Aire transversale

Nous allons déterminer les constantes d'amortissement dans une prochaine section.

4.11 Matrice de gravité

Dans cette section, nous allons définir le vecteur de flottabilité. Les deux forces sont appliquées sur le sous-marin soit la force gravitationnelle au centre de masse Rg ainsi que la poussée d'Archimède au centre de flottaison Rb. Ces deux forces s'opposent sur l'axe des Z pour qu'un objet soit en mesure de flotter, il faut que sa masse soit plus faible que la masse d'eau qu'il déplace. Selon SNAME (1950) nous pouvons définir les deux forces sous la forme vectorielle suivante.

$$W = \begin{bmatrix} 0 \\ 0 \\ \text{mg} \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 0 \\ g\rho\nabla \end{bmatrix}$$

Où:

m : la masse du sous-marin.

g : la constante gravitationnelle.

ρ : densité de l'eau.

∇ : volume submergé

Étant donné que notre sous-marin est toujours submergé, nous pouvons affirmer que le volume submergé est constant. Cependant, ces forces sont décrites dans le référentiel fixe et sont indépendantes au référentiel objet. La prochaine figure représente la force de gravité et la poussée d'Archimède sur le sous-marin en fonction de son orientation.

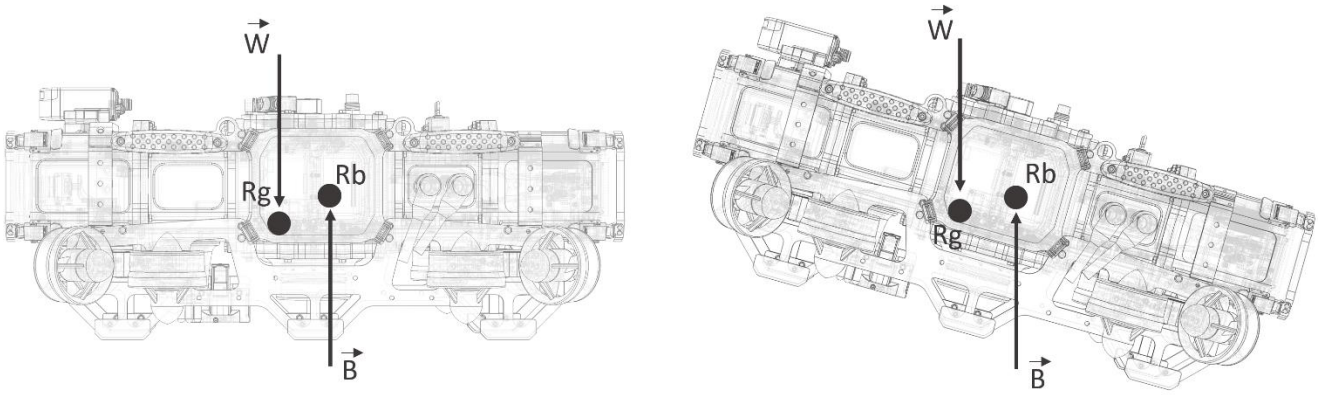


Figure 16: Représentation de la force gravitationnelle et de la poussée d'Archimède sur le sous-marin

Étant donné que notre équation dynamique est décrite dans le référentiel objet, il faut donc transformer ces forces en utilisant l'orientation du sous-marin. La prochaine figure montre la transformation de ces forces dans le référentiel objet.

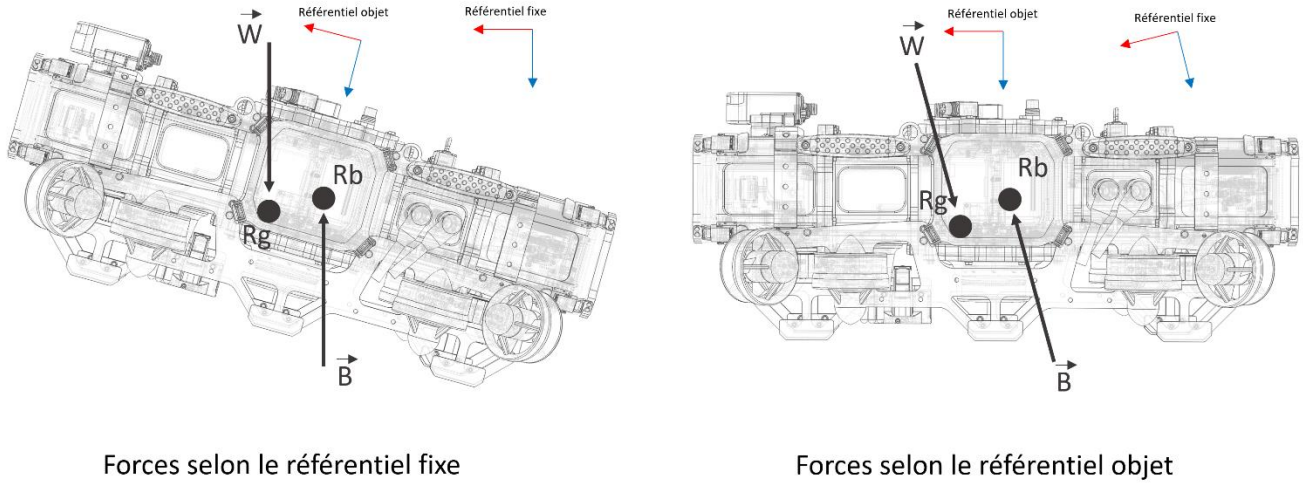


Figure 17: Transformation des forces dans le référentiel objet

Nous pouvons remarquer que les deux forces ont des composantes selon les trois axes (X Y Z). De plus, vu que les forces ne sont pas appliquées au même point, ceci va aussi créer des moments de force dans les trois axes (Rx Ry Rz). Nous pouvons transformer ces deux forces avec le quaternion décrivant l'orientation du sous-marin avec la relation suivante. Nous pouvons réutiliser la matrice E_1 que nous avons défini plus tôt.

$$f_G(e) = E_1^T(e)W \qquad f_B = E_1^T(e)B$$

Nous pouvons aussi transformer ces forces avec la représentation des angles d'Euler avec la relation suivante.

$$f_G(n_2) = J_1^T(n_2)W \qquad f_B(n_2) = J_1^T(n_2)B$$

Nous pouvons donc écrire l'équation d'équilibre en 6 dégrée de liberté qui prend la forme suivante. Ici, f_B , f_G peut dépendre du quaternion ou des angles d'Euler.

$$g_{(n)} = \begin{bmatrix} f_B - f_G \\ r_B \times f_B - r_G \times f_G \end{bmatrix}$$

Nous pouvons développer cette équation avec la représentation basée sur le quaternion. Ceci donne la forme suivante.

$$g(n_q) = \begin{bmatrix} -2\,g\,(\text{mass} - \rho\,\text{volume})\,(\varepsilon_1\,\varepsilon_3 - \varepsilon_2\,\eta) \\ -2\,g\,(\text{mass} - \rho\,\text{volume})\,(\varepsilon_2\,\varepsilon_3 + \varepsilon_1\,\eta) \\ g\,(\text{mass} - \rho\,\text{volume})\,(2\,{\varepsilon_1}^2 + 2\,{\varepsilon_2}^2 - 1) \\ \text{RG}_2\,g\,\text{mass}\,(2\,{\varepsilon_1}^2 + 2\,{\varepsilon_2}^2 - 1) + \text{RG}_3\,g\,\text{mass}\,(2\,\varepsilon_2\,\varepsilon_3 + 2\,\varepsilon_1\,\eta) - \text{RB}_2\,g\,\rho\,\text{volume}\,(2\,{\varepsilon_1}^2 + 2\,{\varepsilon_2}^2 - 1) - \text{RB}_3\,g\,\rho\,\text{volume}\,(2\,\varepsilon_2\,\varepsilon_3 + 2\,\varepsilon_1\,\eta) \\ \text{RB}_1\,g\,\rho\,\text{volume}\,(2\,{\varepsilon_1}^2 + 2\,{\varepsilon_2}^2 - 1) - \text{RG}_3\,g\,\text{mass}\,(2\,\varepsilon_1\,\varepsilon_3 - 2\,\varepsilon_2\,\eta) - \text{RG}_1\,g\,\text{mass}\,(2\,{\varepsilon_1}^2 + 2\,{\varepsilon_2}^2 - 1) + \text{RB}_3\,g\,\rho\,\text{volume}\,(2\,\varepsilon_1\,\varepsilon_3 - 2\,\varepsilon_2\,\eta) \\ \text{RG}_2\,g\,\text{mass}\,(2\,\varepsilon_1\,\varepsilon_3 - 2\,\varepsilon_2\,\eta) - \text{RG}_1\,g\,\text{mass}\,(2\,\varepsilon_2\,\varepsilon_3 + 2\,\varepsilon_1\,\eta) + \text{RB}_1\,g\,\rho\,\text{volume}\,(2\,\varepsilon_2\,\varepsilon_3 + 2\,\varepsilon_1\,\eta) - \text{RB}_2\,g\,\rho\,\text{volume}\,(2\,\varepsilon_1\,\varepsilon_3 - 2\,\varepsilon_2\,\eta) \end{bmatrix}$$

Nous pouvons développer cette équation avec la représentation basée sur les angles d'Euler. Ceci donne la forme suivante.

$$g(n_e) = \begin{bmatrix} g\sin(\theta)\,(m - \rho\,\text{volume}) \\ -g\cos(\theta)\sin(\phi)\,(m - \rho\,\text{volume}) \\ -g\cos(\phi)\cos(\theta)\,(m - \rho\,\text{volume}) \\ g\cos(\theta)\,(\text{RG}_3\,m\sin(\phi) - \text{RG}_2\,m\cos(\phi) + \text{RB}_2\,\rho\,\text{volume}\cos(\phi) - \text{RB}_3\,\rho\,\text{volume}\sin(\phi)) \\ g\,(\text{RG}_3\,m\sin(\theta) - \text{RB}_3\,\rho\,\text{volume}\sin(\theta) + \text{RG}_1\,m\cos(\phi)\cos(\theta) - \text{RB}_1\,\rho\,\text{volume}\cos(\phi)\cos(\theta)) \\ -g\,(\text{RG}_2\,m\sin(\theta) - \text{RB}_2\,\rho\,\text{volume}\sin(\theta) + \text{RG}_1\,m\cos(\theta)\sin(\phi) - \text{RB}_1\,\rho\,\text{volume}\cos(\theta)\sin(\phi)) \end{bmatrix}$$

Il est important de noter que Matlab ne simplifie pas de la même façon que certains livres, mais ce sont des équations équivalentes.

Étant donné que nous supposons que le volume submergé est constant et que notre sous-marin flotte, l'équation que nous venons de définir va toujours vouloir faire monter le sous-marin. En boucle ouverte, cela signifie que le sous-marin va toujours vouloir remonter à la surface et même sortir de l'eau. Cela n'est pas désirable pour le modèle physique de notre simulateur. Cependant, calculer le volume submergé en fonction de la profondeur peut être une tâche compliquée et dans notre cas, nous voulons juste une approximation simple pour que le sous-marin ne sorte pas de l'eau en simulation. Nous avons décidé d'utiliser une équation d'erreur gaussienne, car elle possède deux asymptotes horizontales. Cette équation a la forme suivante.

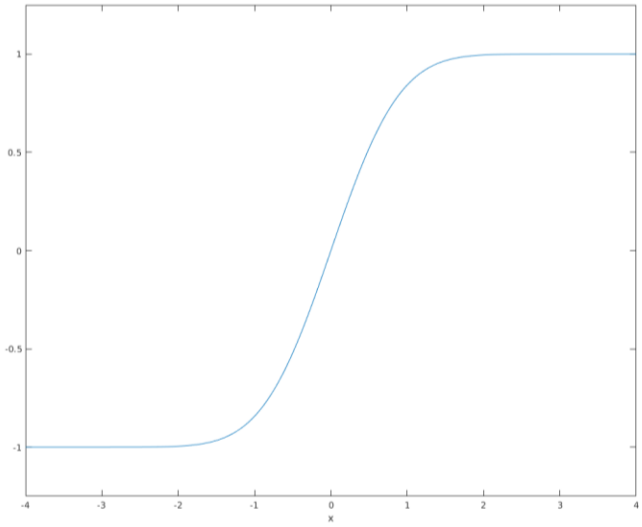


Figure 18: Graphique de la fonction équation d'erreur gaussienne

Nous pouvons donc paramétrer cette équation afin d'avoir un volume submergé nul quand $z \leq 0$ et d'avoir notre volume submergé quand z = la hauteur du sous-marin. L'équation peut prendre la forme canonique suivante.

$$\nabla(z) = a \operatorname{erf}(b(x-h)) + k$$

$$\nabla(z) = \frac{\nabla}{2}\operatorname{erf}(\frac{4}{H}(z-\frac{H}{2})) + \frac{\nabla}{2}$$

Où:

H : la hauteur du sous-marin

Par la suite, nous pouvons remplacer le volume submergé ∇ par la relation $\nabla(z)$ dans le vecteur de flottabilité. Ceci apporte de bons résultats dans le simulateur, mais on ne recommande pas de s'en servir dans la loi de commande, car cette approximation n'est pas représentative d'un point de vue physique.

Matrice de moteurs

Dans cette section, nous allons définir le vecteur τ qui représente la force résultante de la commande pour chaque degré de liberté. Dans notre cas, il s'agit de la somme des forces et moments de tous les moteurs présents sur le sous-marin. Étant donné que l'orientation des moteurs est fixe, nous pouvons nous faire une **4.12** faveur et déjà représenter le vecteur τ sous la représentation d'état linéaire suivant.

$$\tau = Bu$$

Où:

B : est la matrice de commande $\mathbb{R}^{6xm}$

u : vecteur de la commande $\mathbb{R}^{mx1}$

m : le nombre de moteurs

Étant donné que cette matrice B ne sera pas notre véritable matrice de commande de notre système, nous allons la définir T_M afin d'éviter des confusions dans la section d'analyse. Nous pouvons définir l'orientation de la force du moteur selon le référentiel suivant.

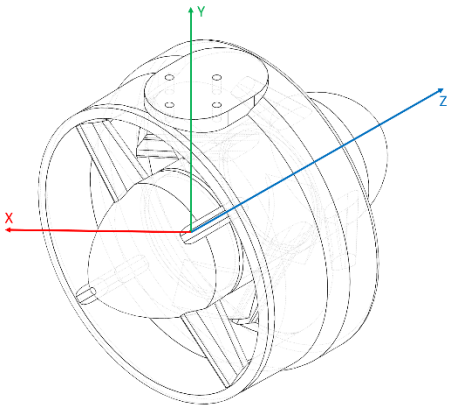


Figure 19: Référentiel moteur

Dans le cas où l'hélice horaire est installée sur le moteur, nous pouvons définir l'orientation du vecteur force par rapport au moteur de la façon suivante :

$$o_T = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Dans le cas où l'hélice antihoraire est installée, il faut inverser l'orientation du moteur. Cependant, ce qui nous intéresse dans la matrice de commande, c'est la réaction des moteurs sur le sous-marin et non la force de poussée du moteur. La figure suivante montre la différence entre les deux.

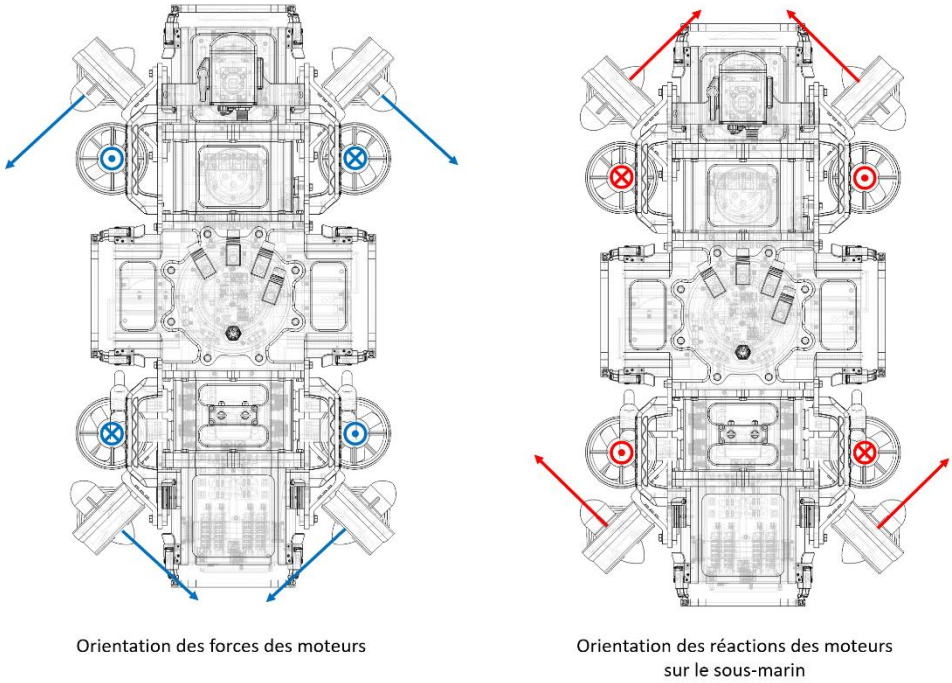


Figure 20: Comparaison entre la force de poussée et la réaction sur le sous-marin

Nous savons que la réaction du moteur sur le sous-marin est inverse à la force de poussée. Nous pouvons donc définir l'orientation du vecteur réaction par rapport au moteur de la façon suivante :

$$o_R = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

Comme pour le vecteur gravité, nous pouvons orienter le vecteur réaction en fonction d'un quaternion qui représente l'orientation du moteur selon le référentiel objet. Nous pouvons utiliser la relation suivante.

$$r_t(e) = E_1(e)o_R$$

Où:

E_1: La matrice de transformation de l'orientation du moteur par rapport référentiel objet $\mathbb{R}^{3x3}$.

Nous pouvons donc écrire la contribution de chaque moteur pour les 6 degrés de liberté avec la relation suivante.

$$T = \begin{bmatrix} r_t(e) \\ d_t \times r_t(e) \end{bmatrix}$$

Où:

d_t: la distance entre le moteur et le centre de masse r_G $\mathbb{R}^3$.

Nous pouvons développer le vecteur T. Cela donne le vecteur suivant :

$$T = \begin{bmatrix} -2\,\varepsilon_1\,\varepsilon_3 - 2\,\varepsilon_2\,\eta \\ 2\,\varepsilon_1\,\eta - 2\,\varepsilon_2\,\varepsilon_3 \\ 2\,{\varepsilon_1}^2 + 2\,{\varepsilon_2}^2 - 1 \\ d_{\mathrm{T2}}\,(2\,{\varepsilon_1}^2 + 2\,{\varepsilon_2}^2 - 1) + d_{\mathrm{T3}}\,(2\,\varepsilon_2\,\varepsilon_3 - 2\,\varepsilon_1\,\eta) \\ -d_{\mathrm{T1}}\,(2\,{\varepsilon_1}^2 + 2\,{\varepsilon_2}^2 - 1) - d_{\mathrm{T3}}\,(2\,\varepsilon_1\,\varepsilon_3 + 2\,\varepsilon_2\,\eta) \\ d_{\mathrm{T2}}\,(2\,\varepsilon_1\,\varepsilon_3 + 2\,\varepsilon_2\,\eta) - d_{\mathrm{T1}}\,(2\,\varepsilon_2\,\varepsilon_3 - 2\,\varepsilon_1\,\eta) \end{bmatrix}$$

Étant donné que la matrice de commande a un nombre de colonnes égales au nombre de moteurs. Nous pouvons répéter le vecteur T pour chaque moteur.

$$B = \forall_i \in m : T_i = \begin{bmatrix} r_t(e_i) \\ d_{t(i)} \times r_t(e_i) \end{bmatrix}$$

Où

m : le nombre de moteurs.

i : index du moteur.

Perturbations

Les forces de perturbations sont des forces externes au système sur lequel nous n'avons pas de contrôle et qui sont imprévisibles. La modélisation des perturbations permet de bien évaluer la robustesse du contrôleur. En raison du contexte actuel, nous ne pouvons pas aller faire des tests expérimentaux en
4.13 piscine. Nous voulons donc que notre modèle soit le plus fiable possible afin d'éviter des surprises non désirables lorsqu'on va pouvoir retourner à l'eau. Nous aimerions modéliser les perturbations en utilisant le spectre de Pierson-Moskowitz comme présenté dans le livre *Guidance and Control of Ocean Véhicules* (section 2.3.1 p.63) Thor I. Fossen (1994). Cependant, par contrainte de temps, nous allons appliquer cette méthode ultérieurement. Pour l'instant, nous avons utilisé des fonctions sinus avec des amplitudes et des fréquences aléatoires pour modéliser les forces des vagues sur les 6 dégrées de liberté. Aussi, nous avons utilisé des fonctions de nombre aléatoire qui sont saturés en fonction des paramètres min et max aussi définit de façon aléatoire pour modéliser les forces de dérive sur les axes x et y. Nous avons linéairement superposé ces deux forces pour créer notre vecteur de perturbation τ_E. En boucle ouverte avec une commande nulle, il est désormais impossible de prédire la position et l'orientation du sous-marin. La figure suivante montre la réponse en boucle ouverte pour deux simulations.

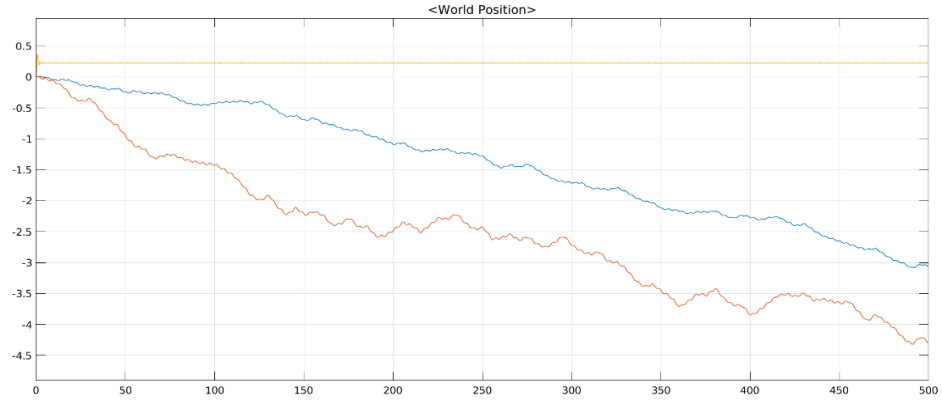


Figure 21: Perturbations (Simulation 1)

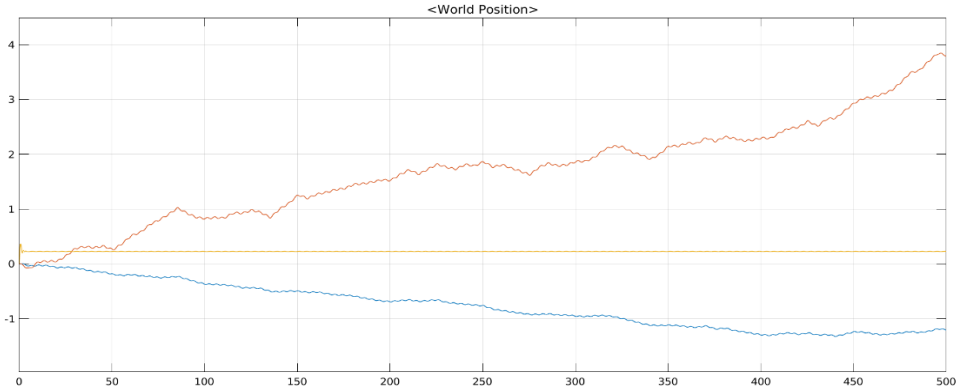


Figure 22: Perturbations (Simulation 2)

Équation différentielle avec la représentation basée sur les quaternions

Pour le modèle avec la représentation basé sur les quaternions, nous pouvons définir les variables d'états du système comme suit :

$$x_q = [\, {n_q}^T \, v^T \,]^T = [x \;\; y \;\; z \;\; \eta \;\; \varepsilon_1 \;\; \varepsilon_2 \;\; \varepsilon_3 \; u \;\; v \;\; w \;\; p \;\; q \;\; r \,]^T \tag{4.14}$$

Avec cette représentation, notre système possède 13 états et conséquemment 13 équations différentielles. Nous pouvons écrire le modèle d'état comme suit :

$$\begin{bmatrix} \dot{n}_q \\ \dot{v} \end{bmatrix} = \begin{bmatrix} E(e)\, v \\ M^{-1}[-C(v)v - D(v)v - g(n_q) + B\, u] \end{bmatrix}$$

Nous pouvons aussi définir un deuxième modèle d'états pour notre simulateur ou on ajoute les perturbations ainsi que le vecteur gravité borné.

$$\begin{bmatrix} \dot{n}_q \\ \dot{v} \end{bmatrix} = \begin{bmatrix} E(e)\, v \\ M^{-1}[-C(v)v - D(v)v - g_b(n_q) + \tau_E + B\, u] \end{bmatrix}$$

Équation différentielle avec la représentation basée sur les angles d'Euler

Pour le modèle avec la représentation basé sur les angles d'Euler, nous pouvons définir les variables d'états du système comme suit :

$$x_e = [\, {n_e}^T \, v^T \,]^T = [x \;\; y \;\; z \;\; \phi \;\; \theta \;\; \psi \;\; u \;\; v \;\; w \;\; p \;\; q \;\; r \,]^T \tag{4.15}$$

Avec cette représentation, notre système possède 12 états et conséquemment 12 équations différentielles. Nous pouvons écrire le modèle d'état comme suit :

$$\begin{bmatrix} \dot{n}_e \\ \dot{v} \end{bmatrix} = \begin{bmatrix} J(n_2)\, v \\ M^{-1}[-C(v)v - D(v)v - g(n_e) + B\, u] \end{bmatrix}$$

5 Définition des constantes

Dans cette section, nous allons expliquer comment nous avons déterminé certaines constantes du modèle physique. Aussi, nous allons expliquer comment nous voulons déterminer certains paramètres avec des tests expérimentaux. En raison du contexte actuel, nous n'avons pas pu faire des expérimentations, car les piscines sont fermées. Cette section va être brève, car elle s'écarte un peu du mandat du projet. De plus des anciens membres du club ont déjà faite cette étude sur notre ancien sous-marin (voir Aubin-Perron, F., Lemay, L 2019).

5.1 Masse, centre de masse et tenseur d'inertie.

Pour déterminer le tenseur d'inertie et le centre masse, nous avons utilisé le logiciel SolidWorks qui est en mesure de déterminer cette matrice pour un assemblage. Afin d'avoir un tenseur d'inertie précis, nous avons pesé toutes les pièces du sous-marin avec une balance. Nous avons, par la suite, remplacé les masses estimées par SolidWorks par les valeurs que nous avons pesées.

Figure 23: Composantes AUV8

Une fois que les masses ont été modifiées dans le SolidWorks, nous pouvons effectuer une analyse de propriété de masse sur l'assemblage du sous-marin. Cette analyse nous donnera la masse, le centre de masse et le tenseur d'inertie. Ensuite, nous pouvons comparer la masse calculée par SolidWorks par la masse que nous avons mesurée afin de valider les constantes estimées par SolidWorks.

La figure suivante montre le résultat de l'analyse que SolidWorks a effectuée sur AUV8.

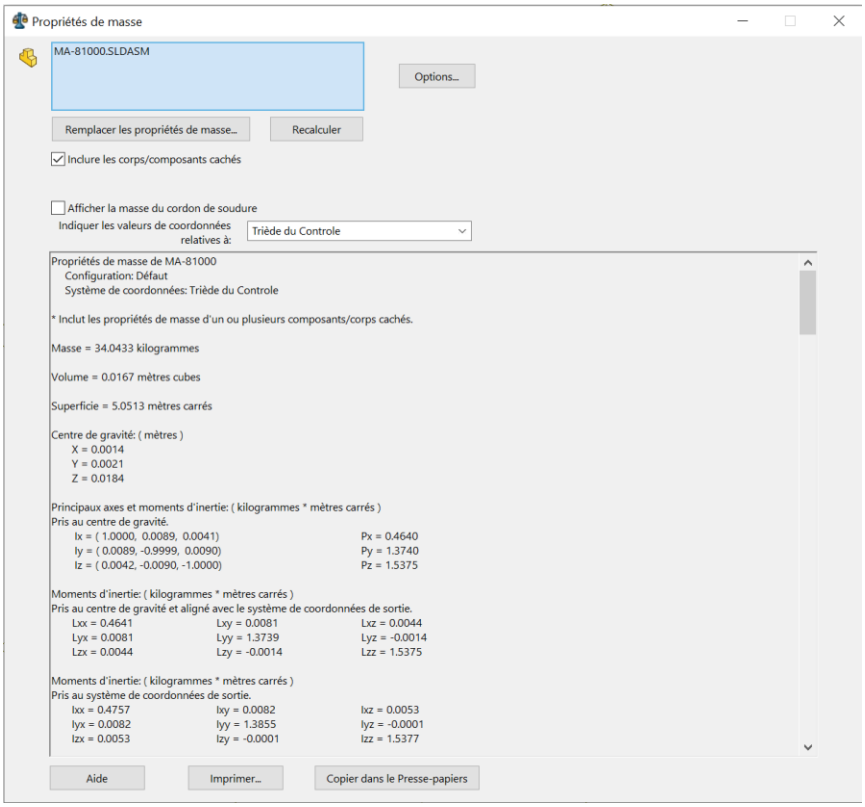


Figure 24: Analyse de AUV8 (SolidWorks 2019)

Après avoir pesé notre sous-marin, nous avons mesuré une masse de 33kg. Nous pouvons donc confirmer que les valeurs calculées par SolidWorks sont valides.

5.2 Volume du sous-marin

Nous pouvons remarquer dans l'analyse de la section précédente, SolidWorks est en mesure de calculer le volume. Cependant, ce volume est erroné, car SolidWorks ne calcule que le volume occupé par la matière. Il ne prendra pas en compte le volume de l'intérieur du sous-marin. Pour pallier ce problème nous avoir dessiner une pièce conceptuelle qui remplir l'intérieur du sous-marin. Nous pouvons à présent déterminer un volume de 27 litres. Nous pouvons remarquer qu'avec ces paramètres le sous-marin coule. L'équipe mécanique travaille présentement sur un système de flotteur.

Masse ajoutée

Dans le livre *Guidance and Control of Ocean Vehicules* (section 2.4 p.38) Thor I. Fossen (1994) nous explique que nous pouvons utiliser la théorie de bande « strip theory » pour déterminer les constantes de masse ajoutée. Cette méthode permet de diviser le sous-marin en plusieurs bandes et de calculer la masse ajouter pour chaque bande et par la suite additionner toutes les bandes pour avoir la masse ajoutée totale. Cependant par manque de temps, nous allons calculer la masse ajoutée dans un futur proche.

5.3

Constante d'Amortissement et centre de flottaison

Pour déterminer les constantes d'amortissement, nous voulons mesurer en piscine des réponses en boucle fermer du sous-marin. Par la suite, nous voulons effectuer de l'ajustement de courbe « curve fitting » afin de déterminer des constantes d'amortissement expérimental. Pour le centre de flottaison, nous pouvons mesurer le quaternion quand le sous-marin flotte au repos. Par la suite, nous pouvons résoudre la matrice gravitée avec le quaternion mesuré afin de déterminer le centre de flottaison. Cependant, en raison de la pandémie, nous n'avons pas été en mesure de faire des tests expérimentaux durant le projet. Nous avons donc estimé des constantes en fonction de celles de AUV7.

5.4

6 Étude linéaire du système en boucle ouverte

Maintenant que nous avons substitué nos constantes mécaniques, nous allons étudier notre système afin de déterminer sa stabilité, sa contrôlabilité et son observabilité. Afin de vérifier ces critères, selon les méthodes vues dans le cours SYS823, nous devons dans un premier temps linéariser le système, car cette méthode ne fonctionne que sur des modèles linéaires. Pour l'étude du sous-marin, nous n'allons pas utiliser la représentation basée sur le quaternion, car il ne possède pas une représentation généralisée (4 paramètres pour 3 degrés de liberté). Pour être en mesure de bien représenter une orientation avec quatre paramètres, le quaternion doit posséder une contrainte afin d'offrir une solution unique. Le quaternion doit donc être unitaire afin de bien décrire une orientation dans l'espace. La contrainte peut être définie de la façon suivante.

$$e^T e = \eta^2 + \varepsilon_1^2 + \varepsilon_2^2 + \varepsilon_3^2 = 1$$

Cependant, lors de la linéarisation du système, cette contrainte disparait et va donc erroné l'étude du système. Il est possible de réduire le quaternion à 3 paramètres en remplaçant la partie scalaire η par la contrainte. Ce qui donne:

$$\eta = \sqrt{-\varepsilon_1^2 - \varepsilon_2^2 - \varepsilon_3^2 + 1}$$

Cependant, comme toute représentation à 3 paramètres, cela va créer des zones de singularité. Dans le cas du quaternion réduit, les zones de singularité apparaissent lorsque $\eta = 0$. Cette représentation n'est pas désirable, car elle contient des zones de singularité qui se trouve dans notre plage d'application. Voici deux exemples de zone de singularité : $\phi = \theta = 0 \,\text{et}\, \psi = \pm\pi$. Nous trouvons donc qu'il est plus avantageux de faire l'étude du système basé sur les angles d'Euler, car ses zones de singularité ne se trouvent pas dans notre plage d'application. De plus, cette représentation est plus intuitive.

Déterminer les points d'équilibres

Afin que la linéarisation soit pertinente, il faut linéariser le système à un point d'équilibre. Un point d'équilibre est un état du système pour lequel le résultat de toutes les équations différentielles est nul. Cela signifie que le système n'évolue plus. Nous pouvons résoudre les équations différentielles de la façon suivante pour déterminer le ou les points d'équilibres. **6.1**

$$0 = f(x)$$

Étant donné que nous étudions le système en boucle ouverte, cela signifie que le vecteur commande doit être constant. Nous pouvons donc assumer que pour toutes les positions x et y, toutes les orientations ψ ainsi que pour toutes les positions $z \geq 0$, le système peut être en condition d'équilibre. Ceci est possible, car les variables d'états $x, y,$ et ψ n'ont pas d'impact sur l'équation dynamique. De plus, cela va nous permettre de nous stabiliser partout dans l'environnement, ce qui est désirable. Nous allons arbitrairement poser le vecteur commande u, la position x et y ainsi que l'orientation ψ à 0.

Après analyse numérique, nous trouver le point d'équilibre qui représente le sous-marin qui flotte au repos.

$$x_{ss} = \begin{bmatrix} n_{ess} \\ v_{ss} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0.2243 \\ -0.02362 \\ -0.00787 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad u_{ss=} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Où:

x_{ss}: le point d'équilibre
u_{ss}: la commande d'équilibre

Nous pouvons remarquer que le balancement du sous-marin n'est pas parfaitement balancé, car il ne se stabilise pas à $\phi = \theta = 0$.

Linéariser au point d'équilibre

La linéarisation d'un modèle d'états autour d'un point d'équilibre permet de représenter une approximation linéaire de ce système autour de ce point. Plus on s'en éloigne de ce point et plus notre approximation sera fausse. De plus, l'origine de ce système linéarisé devient le point d'équilibre $0 = x_{ss}$.

6.2 Il faut donc soustraire le vecteur d'état réel $x(t)$ par le point d'équilibre x_{ss}. Nous allons appeler ce changement de variable $\tilde{x}(t)$. Le modèle linéarisé peut être écrit sous la forme matricielle suivante.

$$\dot{\tilde{x}}(t) = A\tilde{x}(t) + B\tilde{u}(t)$$
$$\tilde{y}(t) = C\tilde{x}(t) + D\tilde{u}(t)$$

Avec : $\tilde{x} = x(t) - x_{ss}$, $\tilde{u} = u(t) - u_{ss}$ et $x(t) = [x\ y\ z\ \phi\ \theta\ \psi\ u\ v\ w\ p\ q\ r\ ]^T$

Où :

A : La matrice de d'états $\mathbb{R}^{n\times n}$

B : La matrice de commande $\mathbb{R}^{n\times m}$

C : La matrice d'observabilité $\mathbb{R}^{p\times n}$

D: La matrice d'action direct $\mathbb{R}^{p\times m}$

n : Le nombre d'état (12)

m : Le nombre d'entrées (moteur, 8)

p : Le nombre de sorties (valeur mesuré, 9)

Pour déterminer la matrice d'état A, nous pouvons calculer la matrice jacobienne en fonction des variables d'états. Puisqu'il y a autant d'équations différentielles que de variable d'états, la matrice jacobienne va être carré et de dimension n. Cela donne la matrice suivante :

$$A(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} f_1 & \cdots & \frac{\partial}{\partial x_n} f_1 \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_1} f_n & \cdots & \frac{\partial}{\partial x_n} f_n \end{bmatrix} | x = x_{ss}$$

Dans notre cas, la matrice A au point d'équilibre ressemble à la matrice ci-dessous :

$$A(x_{ss}) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1.0 & 4.186e-5 & -0.007873 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.0001441 & 0.9997 & 0.02362 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.007872 & -0.02362 & 0.9997 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.0 & 0.0001859 & -0.00787 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.9997 & 0.02362 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.02362 & 0.9998 \\ 0 & 0 & 0 & -0.002208 & 1.562 & 0 & -1.344 & -0.000179 & 0.001057 & -0.0003471 & 0.1064 & 0.03069 \\ 0 & 0 & 0 & -1.703 & 0.001437 & 0 & -0.0001217 & -1.797 & -0.003845 & -0.1831 & 5.751e-5 & 0.01242 \\ 0 & 0 & 0 & -0.1139 & -0.03262 & 0 & 0.000681 & -0.003295 & -2.062 & -0.02033 & -0.003936 & 0.0002428 \\ 0 & 0 & 0 & -31.2 & 0.02103 & 0 & -0.001098 & -1.099 & -0.1424 & -6.78 & 0.002356 & 0.07658 \\ 0 & 0 & 0 & 0.02793 & -25.92 & 0 & 0.6843 & 0.001112 & -0.03928 & 0.007185 & -3.943 & 0.01303 \\ 0 & 0 & 0 & 0.4847 & -0.7096 & 0 & 0.09207 & 0.04966 & 0.001133 & 0.05105 & 0.006103 & -10.35 \end{bmatrix}$$

Nous pouvons remarquer que notre matrice contient quatre colonnes de zéros. Comme nous l'avons mentionné plus haut, ceci confirme que les variables d'état $x, y, \text{et}\, \psi$ n'ont pas d'impact sur le système. Cependant, nous ne pouvons pas réduire le nombre d'états, car nous n'avons pas des rangées complètes de zéros. Il est possible et viable d'avoir une matrice A avec des colonnes de zéros.

Pour la matrice de commande B, nous n'avons pas besoin calculer la matrice jacobienne en fonction des variables d'entrée, car nous avons déjà défini le vecteur de commande sous la forme $\tau = Bu$ dans l'équation dynamique. De plus, nous savons que la commande n'a aucun impact direct sur l'équation cinématique. Nous pouvons donc écrire la matrice de commande de la façon suivante.

$$B = \begin{bmatrix} 0_{6\times m} \\ T_M \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.02123 & 0.02123 & 0.01984 & 0.01984 & 0.002735 & 0.002785 & -0.002796 & -0.002724 \\ -0.02018 & 0.02075 & -0.02074 & 0.02019 & 0.003023 & -0.00302 & -0.002803 & 0.0028 \\ 0 & 0 & 0 & 0 & 0.02969 & -0.02989 & 0.02925 & -0.02904 \\ 0.01516 & -0.01167 & 0.0117 & -0.01514 & 0.1119 & -0.1118 & -0.1038 & 0.1037 \\ 0.01144 & 0.01147 & 0.01085 & 0.01088 & -0.1012 & -0.1032 & 0.1035 & 0.101 \\ -0.2357 & -0.2377 & 0.2326 & 0.2306 & -0.0007 & 0.001002 & 0.0006211 & -0.0009375 \end{bmatrix}$$

La matrice d'observabilité C ne retourne que les états que nous sommes de mesurer avec nos capteurs. Cette matrice contient généralement que des 1 et des 0. Il est possible de la définir manuellement, mais il est plus simple de calculer la matrice jacobienne des variables de sortie en fonction des variables d'état. Puisque nous pouvons estimer la position du sous-marin, nous pouvons définir les 2 vecteurs de variable et la matrice C comme suit.

$$x = [x\ y\ z\ \phi\ \theta\ \psi\ u\ v\ w\ p\ q\ r\ ]^T$$
$$y = [x\ y\ z\ \phi\ \theta\ \psi\ u\ v\ w\ p\ q\ r\ ]^T$$
$$C = \begin{bmatrix} \frac{\partial}{\partial x_1} y_1 & \cdots & \frac{\partial}{\partial x_n} y_1 \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_1} y_p & \cdots & \frac{\partial}{\partial x_n} y_p \end{bmatrix}$$

Dans notre cas, vu que nous pouvons mesurer ou estimer tous nos états, la matrice C revient à être identitaire.

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Nous pouvons annuler la matrice D, car le système ne possède pas d'action directe. Cependant, Matlab a besoin d'une matrice D pour créer un objet « State Space ». Nous allons donc définir une matrice nulle.

$$D = [0_{p \times m}]$$

Étude de stabilité

Dans cette section, nous allons étudier la stabilité du système linéarisé. Pour trouver les pôles d'un système représenté sous la forme d'état, nous pouvons calculer les valeurs propres de la matrice A de notre système au point d'équilibre. Nous avons déterminé les pôles suivants.

6.3

$$pôles_{ss} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -10.3468 \\ -1.9938 + 4.6887i \\ -1.9938 - 4.6887i \\ -3.4244 - 4.4289i \\ -3.4244 + 4.4289i \\ -1.3006 \\ -2.0618 \\ -1.7292 \end{bmatrix}$$

Nous pouvons remarquer que le système est stable à l'exception de quatre pôles à zéros. Un pôle à réel à zéro indique un intégrateur pur (aucune amortissement, oscillation et divergence). Cela s'explique à cause des colonnes de zéros dans la matrice A, car les états x, y, z et ψ sont des intégrations directes des états $u\ v\ w\ r$ et n'ont aucun impact sur le système. Nous pouvons donc confirmer que le système est localement stable. Nous pouvons afficher les pôles dans le plan complexe.

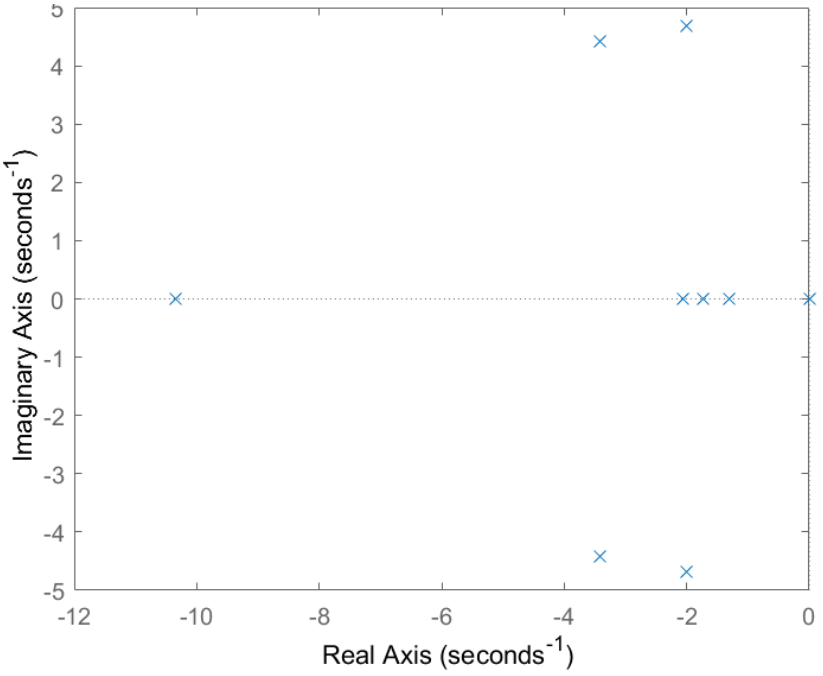


Figure 25: Pôles du système

Étude de commandabilité

Dans la plupart des systèmes MIMO (multi-input, multi-output), la commandabilité devient une propriété aussi importante que la stabilité. Un système est dit « commandable » si le vecteur de commande u sont capables de bouger tous les états du système d'un point initial à un point final dans un intervalle de temps fini. Dans notre cas, il est important que la configuration de nos moteurs puisse appliquer une commande sur les 6 degrés de liberté. Nous pouvons calculer la matrice de commandabilité de la façon suivante.

6.4

$$\mathcal{C} = [B \ \text{AB} \ A^2B \ \dots \ A^{n-1}B \ ]$$

Afin de vérifier la commandabilité, il faut vérifier que le rang de la matrice de commandabilité soit égal au nombre d'états du système (n). Dans notre cas, le rang de matrice est de 12

$$\text{rank}(\mathcal{C}) = n = 12$$

Notre système est donc commandable. Cependant, cela ne signifie pas que le système peut atteindre toutes les orientations demandées, car l'étude de commandabilité ne prend pas en compte la saturation du vecteur de commande u.

Étude d'Observabilité

6.5

Un système est dit « observable » s'il est possible de mesurer ou d'estimer toutes les variables d'états internes d'un système. Dans notre cas, nous ne sommes pas en mesure de mesurer les positions linéaires du sous-marin. Cependant, nous ne sommes pas obligés de concevoir un observateur, car la relation entre les états est assez évidente. Nous pouvons simplement multiplier les vitesses du DVL par la matrice $E_1(e)$ où e est le quaternion mesuré par l'IMU. Nous sommes donc en mesurer et estimer tous les états du système. Si nous voulons être rigoureux, nous pouvons le vérifier en calculant la matrice d'observabilité de la façon suivante.

$$\mathcal{O} = [C \ \text{CA} \ CA^2 \ \dots \ CA^{n-1} \ ]^T$$

Afin de vérifier l'observabilité, il faut vérifier que le rang de la matrice d'observabilité est égal au nombre d'états du système (n). Dans notre cas, le rang de matrice est de 12

$$\text{rank}(\mathcal{O}) = n = 12$$

Le système est donc observable.

7 Asservissement du sous-marin

La section suivante présentera la méthode que nous avons utilisée afin d'asservir le sous-marin. Nous allons décrire les raisons menant à notre choix de l'algorithme de contrôle, la description de celui-ci ainsi que les méthodes et l'implémentation à l'aide de Matlab et Simulink.

7.1 Choix et description

Afin de réaliser l'asservissement du sous-marin que nous avons modélisé, nous avons opté pour une méthode avancée de contrôle. Nous avons fait l'utilisation d'un Model Predictive Control (MPC) pour asservir, la position, l'orientation, la vitesse linéaire et la vitesse angulaire du sous-marin. Un MPC permet d'utiliser un modèle de référence pour effectuer des prédictions sur le comportement futur à l'aide d'un filtre de Kalman intégré. Le MPC résout un modèle d'optimisation linéaire à chaque instant pour déterminer la meilleure commande à envoyer au système afin que ce dernier respecte la référence donnée et des contraintes précises. Voici un schéma explicatif du fonctionnement d'un MPC.

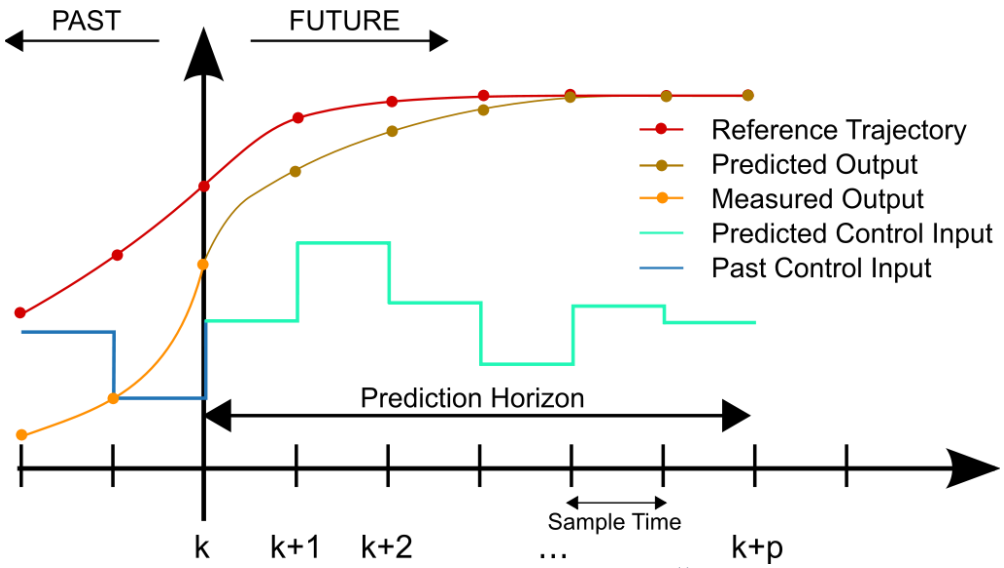


Figure 26: Fonctionnement du MPC [11]

Nous avons choisi d'utiliser un MPC puisqu'il s'agit d'une commande que nous considérons plutôt facile à maintenir et qui est à la fois très robuste. Le MPC peut également prendre des prédictions en entrée, ce qui lui permet de bien fonctionner avec un suivi de trajectoire. De plus, le MPC nous a permis de concevoir un seul contrôleur sans avoir à découpler les axes de notre sous-marin.

[11] Tirée de Wikipédia (Modifié en 2021)

Méthode

Par défaut, la commande MPC est une commande linéaire. Cependant, plusieurs outils sont offerts afin de travailler avec des systèmes non linéaires. Dans notre cas, le sous-marin est un système non linéaire. Cependant, les non-linéarités ne varient pas énormément sur une courte période. On peut donc affirmer que
7.2 le système n'est pas hautement non linéaire. De plus, le nombre d'états et le temps d'échantillonnage ne varieront pas avec le temps. Dans un cas comme celui-ci, nous pouvons généralement utiliser un MPC adaptatif. Cette variante utilise les Jacobiennes du modèle linéarisé au point d'opération actuel. Cependant, nous avons fait l'essai de cette méthode sur notre modèle représenté avec le quaternion, mais l'utilisation d'un filtre de Kalman avec le quaternion n'était pas concluante. L'erreur de l'estimation était trop grande. Afin de pallier à ce problème, nous pouvons utiliser un filtre de Kalman externe de type étendu (FKE) qui estime mieux les non-linéarités. Pour faire plus simple, nous avons fait l'utilisation de l'objet MPC non linéaire de Matlab qui utilise un FKE ainsi avec une option pour le traiter comme un MPC linéaire un solveur non linéaire. L'estimation des états est non linéaire, mais la commande est linéaire. Bien que ce type de MPC avec un filtre de Kalman étendu soit plus demandant en ressources, nos tests ont démontré des résultats notables.

7.3 Implémentation

Cette sous-section présente l'implémentation du MPC à l'aide de Matlab et de Simulink. Nous présenterons les spécifications, les contraintes, les poids du contrôleur, les caractéristiques du contrôleur lui-même ainsi que la « Model Predictive Control toolbox ».

7.3.1 *Spécifications du système*

Pour débuter, nous devons définir certaines spécifications du système et du MPC dont nous aurons besoin pour l'implémentation du contrôleur. Nous avons besoin de certains paramètres de base pour le MPC. On y retrouve les paramètres suivants.

Tableau 4 : Paramètres pour le MPC

Description	Valeur
Nombre d'états	13
Nombre de sorties	13
Nombre d'entrées	8
Période d'échantillonnage	0.25
Horizon de prédiction	4
Horizon de contrôle	2

Pour définir les valeurs de l'horizon de contrôle ainsi que l'horizon de prédiction, nous avons effectué des tests pour tenter de trouver les meilleures valeurs qui fonctionnent bien pour notre système. Pour la période d'échantillonnage, nous avons choisi un temps arbitraire de 0.25 seconde étant donné que

notre système est plutôt lent. De plus, cela permet également de réduire la quantité de ressources utilisées par le MPC.

Contraintes et conditions initiales

Afin de respecter certaines contraintes mécaniques et électriques, nous avons donné des contraintes au MPC afin que ce dernier en tienne compte lors de son optimisation. Ces contraintes que nous avons définies permettent au MPC de donner des commandes dans un ordre de grandeur raisonnable pour le sous-marin. Nous avons donc défini des limites pour les forces des moteurs ainsi que pour la vitesse du sous-marin. Voici les contraintes des moteurs. Notez bien que tous les moteurs ont la même contrainte de force.

7.3.2

Tableau 5: Contrainte de force pour les moteurs

Force minimale (N)	Force maximale (N)
-24	29

Voici également les contraintes pour la vitesse du sous-marin. Notez bien ici que les limites présentées ci-bas sont les mêmes pour chacun des axes.

Tableau 6: Contrainte de vitesse du sous-marin

Axes	Vitesse minimale	Vitesse maximale
Translation (u v w)	-1 m/s	1 m/s
Rotation (p q r)	-0.2 rad/s	-0.2 rad/s

Par la suite, nous avons également donné les conditions initiales de la commande et des états du système au MPC. Pour les moteurs, nous avons mis des forces nulles initiales et pour les états du système, nous avons mis notre point d'équilibre initial.

7.3.3

Poids du contrôleur

Le choix des poids dans la conception d'un MPC est une étape importante puisqu'elle permet de définir les comportements du système relatifs à la rapidité et la robustesse du contrôleur. Il existe 3 types de poids qui peuvent être changés pour un MPC. Le premier type, **« OutputVariables »**, permet de donner une plus grande importance à certaines sorties du système. Il s'agit de la fonction objective que le contrôleur veut minimiser. Le deuxième type, **« ManipulatedVariables »**, permet de donner un poids à chacune des entrées du système et permet également à ces dernières de varier temporairement par rapport à leur cible pour aider les sorties à atteindre leur cible. Le troisième type, **« ManipulatedVariablesRates »**, permet quant à lui de donner un poids sur l'agressivité des entrées du système. Il s'agit en quelque sorte des ratios de changement de chacune des entrées et plus ce poids est grand, plus petit seront les incréments.

Nous trouvons que les poids du MPC sont beaucoup plus intuitifs que les gains que l'on peut retrouver sur des contrôleurs plus traditionnels. Cela nous permet de les modifier plus rapidement et simplement. Aussi, il sera plus facile pour les nouveaux membres de modifier les gains du contrôleur, car aucune notion d'équations différentielles n'est requise pour comprendre ces gains. En addition, les poids du MPC peuvent être changés durant l'exécution. Pour plus d'information, nous vous invitons à consulter la documentation suivante : https://www.mathworks.com/help/mpc/ug/tuning-weights.html. Voici les poids de base que nous avons définis pour le contrôleur.

Tableau 7 : Poids initiaux du contrôleur

Description	Valeurs
Output Variables	[70, 60, 70, 50, 50, 50, 50, 0, 0, 0, 0, 0, 0]
Manipulated Variables	[0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2]
Manipulated Variables Rates	[0.1, 0.1, 0.1, 0.1, 0.3, 0.3, 0.3, 0.3]

Pour les « **OutputVariables** », nous avons donnée des gains pour les positions pour le contrôle en position. Si nous désirons contrôler le sous-marin en vitesse, nous allons plutôt mettre les poids pour les vitesses. Les **« ManipulatedVariables »** ont tous été mis à la même valeur recommandée par la documentation Matlab du MPC. Finalement, nous avons donné des gains pour les **« ManipulatedVariablesRates »** afin de définir l'agressivité des moteurs. Ici, on a choisi d'augmenter les gains pour les moteurs 5 à 8 qui correspondent aux moteurs horizontaux. Ce sont généralement ces moteurs qui contrôlent la profondeur, le roulis (roll) et le tangage (pitch).

7.3.4

Contrôleur MPC

Dans Matlab, il est possible de créer un objet MPC afin de l'utiliser dans Simulink. Nous avons affecté les spécifications du système, les contraintes, les conditions initiales du système et les poids du contrôleur à cet objet Matlab.

Nous avons également spécifié les équations du modèle physique ainsi que les Jacobienne du système afin que le MPC puisse les prendre en compte pour faire son calcul d'optimisation.

```
nlobj.Model.StateFcn = "AUVQuatSimFcn";
nlobj.Jacobian.OutputFcn="AUVQuatSimFcn";
nlobj.Jacobian.StateFcn = @AUVQuatJacobianMatrix;
```

Finalement, nous avons spécifié quelques paramètres du solveur pour optimiser notre MPC. Nous avons choisi de diminuer les tolérances du contrôleur pour permettre au contrôleur de converger vers des valeurs avec un degré de précision moindre. De plus, nous avons également activé le traitement parallèle afin de rendre l'optimisation plus rapide. Un dernier paramètre important que nous avons ajouté permet de traiter le MPC non linéaire comme un MPC tel que nous l'avons mentionné plus haut.

```
nlobj.Optimization.SolverOptions.ConstraintTolerance = 0.02;
nlobj.Optimization.SolverOptions.OptimalityTolerance = 0.02;
nlobj.Optimization.SolverOptions.FunctionTolerance = 0.02;
nlobj.Optimization.SolverOptions.StepTolerance=0.1;
nlobj.Optimization.SolverOptions.UseParallel=true();
nlobj.Optimization.SolverOptions.Algorithm='sqp';
nlobj.Optimization.RunAsLinearMPC='adaptive';
```

7.3.5 *Matlab et Simulink*

Afin d'implémenter le MPC dans le cadre de notre projet, nous avons fait l'utilisation de la « toolbox » MPC dans Matlab et Simulink. Cette boîte à outils offre plusieurs outils afin de concevoir des contrôleurs MPC (linéaires ou non linéaires). Afin d'en apprendre davantage sur la « toolbox », nous vous invitons à consulter la documentation suivante : https://www.mathworks.com/products/model-predictive-control.html. Voici les différents blocs disponibles dans la « toolbox ».

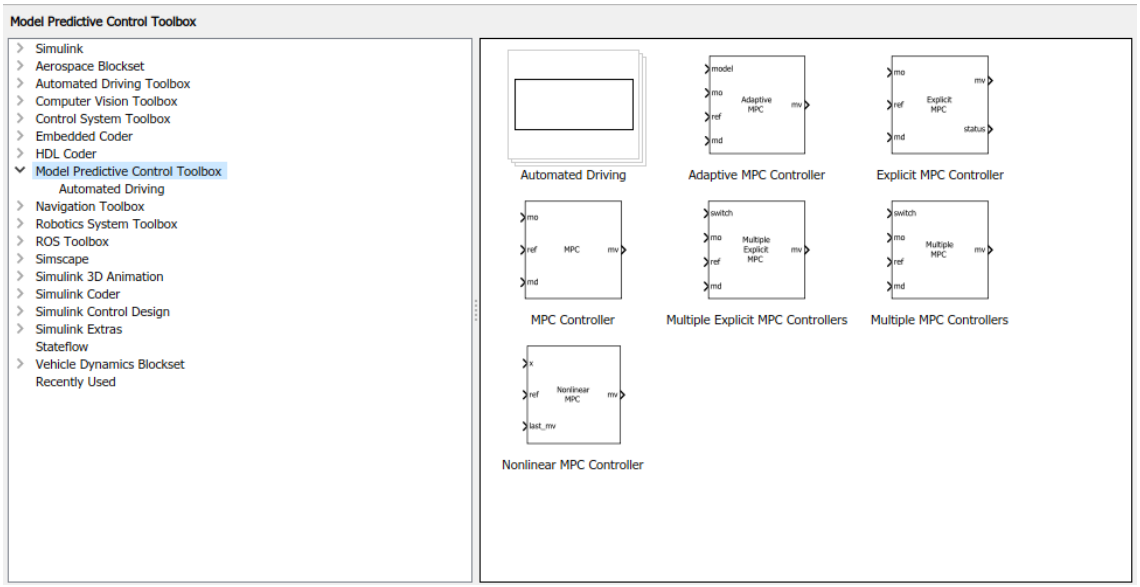


Figure 27 : Librairie MPC toolbox (Matlab 2020a)

Pour notre part, nous avons utilisé le bloc « Nonlinear MPC Controller ». Voici l'implémentation de ce bloc dans notre contrôle. Dans ce schéma, nous pouvons observer que nous donnons en entrée les contraintes, les conditions initiales ainsi que les poids du contrôleur. À cela s'ajoutent les états, la référence ainsi que la dernière commande envoyée au système. Les sorties du bloc correspondent à la commande ainsi que le statut du contrôleur. Ce statut nous renseigne sur le bon ou le mauvais fonctionnement du MPC.

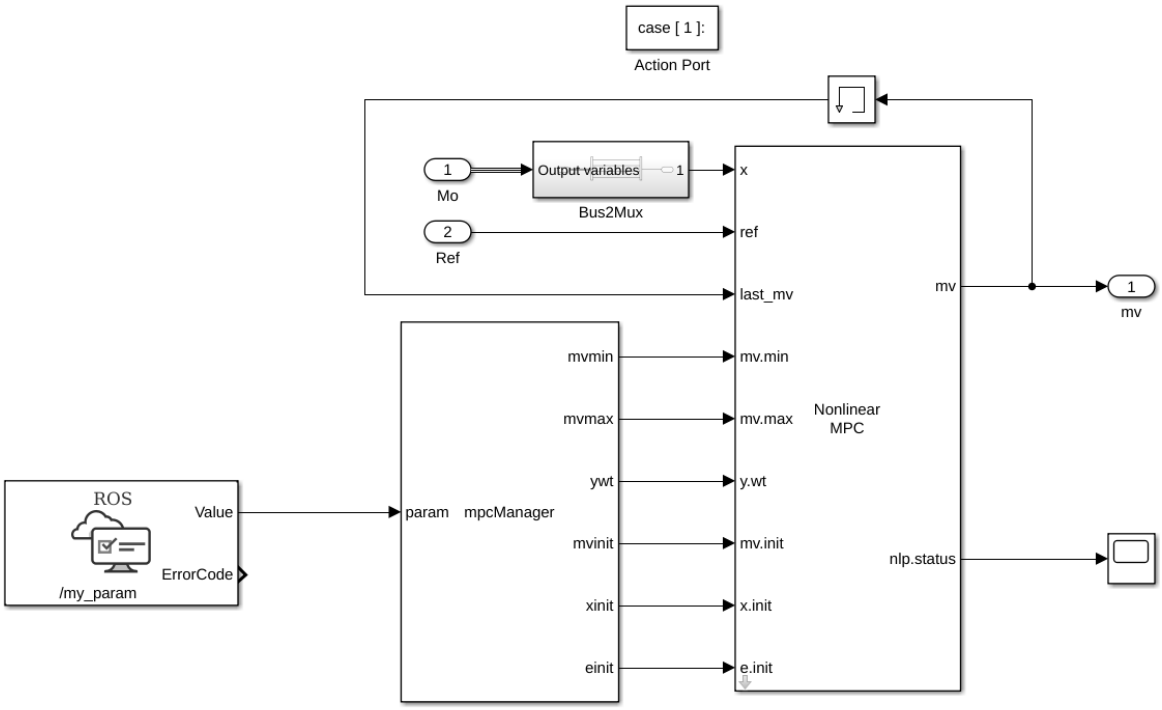


Figure 28 : Utilisation du bloc « Nonlinear MPC Controller »

8 Implémentation logicielle

Dans cette section, nous allons aborder le développement et l'implémentation logicielle de notre contrôle. Nous allons discuter de l'architecture logicielle actuellement en place sur le sous-marin, du choix des technologies que nous avons utilisées, de la méthode de travail pour déployer le modèle mathématique, de l'utilisation de ROS avec Simulink et du déploiement du logiciel de notre contrôle sur notre sous-marin.

8.1 Architecture logicielle

Dans le cadre d'un projet d'envergure tel que le sous-marin, il est primordial de mettre en place une architecture logicielle efficace pour regrouper l'ensemble des nœuds nécessaires au bon fonctionnement du sous-marin. S.O.N.I.A a mis en place une architecture logicielle bien définie que nous devons respecter afin de mettre en œuvre le logiciel de notre contrôleur. Nous tenons à préciser qu'au moment de la lecture de cet article, il est possible que l'architecture logicielle ait changé. Les membres S.O.N.I.A travaillent présentement sur l'amélioration de cette architecture. Voici l'architecture logicielle de S.O.N.I.A que nous avons actuellement. Ce schéma est tiré de la documentation officielle du club S.O.N.I.A.

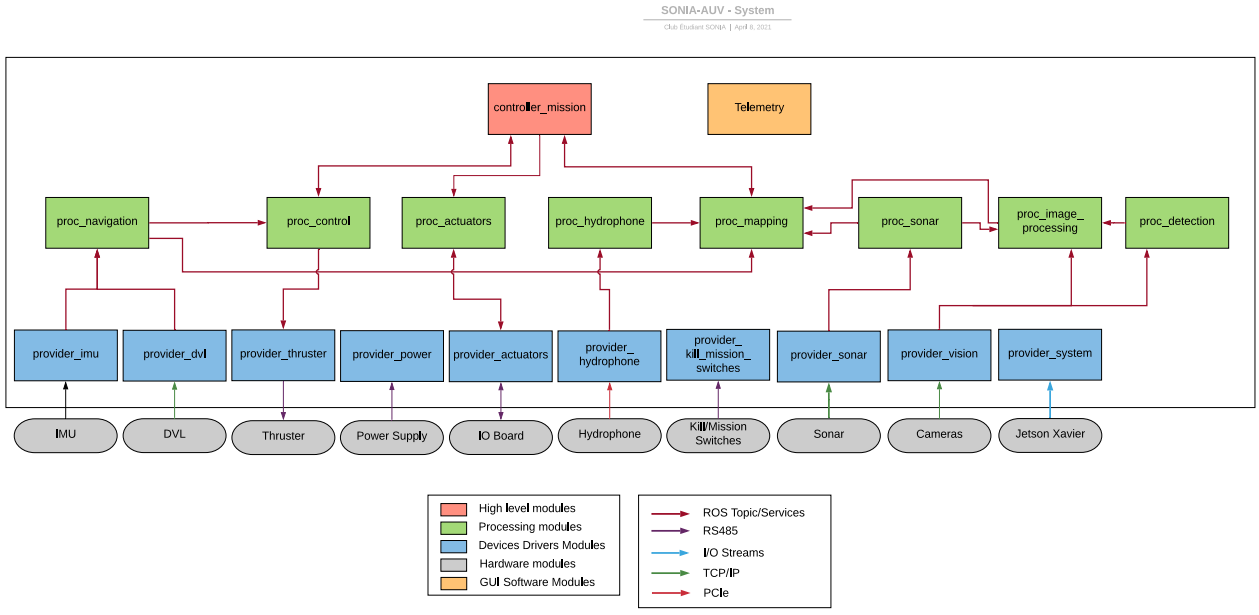


Figure 29 : Architecture logicielle du sous-marin [12]

[12] Tirée de la documentation du club SONIA

Dans la légende du schéma ci-haut, nous retrouvons les différentes technologies de communication utilisées sur le sous-marin. De plus, on retrouve aussi la liste des différents types de modules que l'équipe logicielle a conçus pour le sous-marin. Le premier type, en gris, correspond aux modules bas niveaux directement sur le matériel du sous-marin. Le code est généralement en C/C++ et est compilé sur des microcontrôleurs. Le deuxième type, en bleu, rassemble tous les modules de type pilote qui communique avec les microcontrôleurs via les protocoles spécifiés dans la légende. Les données recueillies sont ensuite envoyées au prochain type à l'aide de messages ou service ROS. Le troisième type, en vert, correspond quant à lui aux modules de traitement. Ces modules permettent de faire le traitement des données reçues par les modules pilotes que l'on vient de décrire. Le traitement permettra entre autres de convertir les données ou formater celles-ci pour être utilisé par la télémétrie ou d'autres modules de traitement. Le quatrième type, en rouge, correspond aux modules de haut niveau. Ce type de module regroupe les logiciels hauts niveaux que l'on retrouve sur le sous-marin. Ici, on fait référence surtout aux missions qui sont constituées de plusieurs états programmés en Python. Le cinquième et dernier type, en jaune, correspond à la télémétrie. La télémétrie est une interface Web qui permet d'afficher des données du sous-marin en temps réel durant les tests en piscine. Elle permet aux équipes mécanique, logicielle et électrique de connaître l'état du sous-marin tel que la tension des batteries, la force des moteurs, l'orientation, la position, l'image des caméras et plus encore.

Les nœuds en vert, en jaune, en bleu et rouge fonctionnent tous à l'intérieur de « containers » Docker sur le sous-marin afin de limiter l'installation de dépendances sur l'ordinateur de bord du AUV. Docker est en quelque sorte un conteneur qui regroupe toutes les dépendances nécessaires pour un projet. Chacun des modules est déployé dans son propre conteneur et l'ensemble des modules sont ensuite reliés par un outil appelé Docker Compose. De plus, S.O.N.I.A a choisi de faire l'utilisation de ROS (Robot Operating System) comme « framework » pour concevoir le logiciel des modules. ROS est un outil très utilisé dans le domaine de la robotique et offre un large éventail d'outils pour aider dans le développement logiciel d'un robot. Ce « framework » possède également une riche documentation mise à jour par une grande communauté.

Enfin, nous devons s'adapter à l'ensemble des technologies utilisées par S.O.N.I.A pour faire l'implémentation logicielle du contrôleur (ROS et Docker). Nous devons également tenir compte de l'architecture logicielle pour s'assurer que l'intégration du contrôleur soit faite facilement et indépendamment d'autres modules. Dans la prochaine section, nous discuterons des choix technologiques que nous avons effectués pour faire le développement du contrôle.

Choix technologiques

Pour faire le développement logiciel du contrôle du sous-marin, nous avons opté pour l'utilisation de Matlab et de Simulink. Nous avons fait ce choix étant donné la multitude d'outils que Matlab et Simulink offrent pour développer des logiciels complexes tels qu'une « toolbox » spécifique pour la robotique. Cet outil permet de développer des nœuds ROS pour communiquer directement avec l'ensemble des modules de S.O.N.I.A. Une autre « toolbox » permet aussi de faire la conception d'un MPC que nous avons utilisé pour faire notre contrôleur. Étant donné qu'il s'agit d'un projet qui requiert l'usage de beaucoup de mathématiques, il devient également très intéressant de faire l'utilisation de Matlab. Enfin, à l'aide de Simulink, nous sommes capables de faire de nombreuses simulations pour tester notre modèle physique et l'ensemble du contrôleur.

8.2

Une autre raison, moins technique, a également influencé notre choix. Étant donné le nombre réduit de membres au sein de S.O.N.I.A, nous croyons que l'utilisation de logiciels tels que Matlab et Simulink peut permettre de simplifier le développement d'un algorithme aussi complexe que notre contrôleur. Aussi, la plupart des membres qui ont travaillé sur le contrôle dans le passé sont inscrits au programme de Génie de la production automatisé à l'ÉTS et ont fait l'utilisation de Matlab et Simulink dans certains de leurs cours. Ils sont donc déjà familiers avec l'outil de développement. Un tel développement directement en C++ aurait pris beaucoup plus de temps à développer sans l'utilisation de Matlab et Simulink. Avec ces outils, nous sommes en mesure de déployer directement notre code en C++ pour l'utiliser sur le sous-marin. Nous allons discuter plus en détail du déploiement logiciel dans les sections suivantes.

Nous désirons mentionner un outil très pertinent de Matlab que nous avons utilisé. Il s'agit de LiveScript. Les LiveScripts nous ont permis de bien documenter notre développement à même le code. Un LiveScript permet directement de faire l'ajout de texte et de figure à même le code. De plus, le code créé génère également de la documentation. Bien que les LiveScripts ont contribué à la génération de ce document, nous croyons que cet outil est parfait pour laisser une documentation complète de notre développement aux futurs membres.

Modèle physique

La programmation du modèle physique est une étape très importante dans le cadre de ce projet, car elle permet de modéliser la physique du sous-marin pour pouvoir l'utiliser pour faire des simulations. Afin de générer le modèle d'état non linéaire du sous-marin, nous avons procédé en plusieurs étapes.

8.3

Tout d'abord, nous avons symboliquement écrit équations dynamiques à l'aide de la « Symbolic Math Toolbox » de Matlab. Ensuite, nous avons substitué les constantes du modèle physique spécifiques au sous-marin avec les valeurs numériques. Pour terminer, nous avons été en mesure de générer des fonctions Matlab à partir des équations symboliques pour utiliser dans Simulink. Nous avons généré une fonction pour l'équation non linéaire du sous-marin avec prise en compte des perturbations, une autre fonction sans prise en compte des perturbations et une fonction pour générer les matrices Jacobiennes en un point d'opération. Ces fonctions Matlab ont donc pu être utilisées à l'intérieur de Simulink pour effectuer des simulations. Voici comment nous avons fait la génération d'une fonction Matlab appelée « AUVQuatPerturbedSimFcn » à l'aide de la fonction matlabFunction().

```
% Générer l'équation non linéaire (quaternion) avec perturbation pour simulation
matlabFunction(transpose(simqpc),'File',
'../simulink/auv_plant/script/AUVQuatPerturbedSimFcn',
'Vars', {[Nq;V],F_ext,U});
```

Cette fonction permet de convertir une équation symbolique en fonction Matlab. Elle prend en paramètre l'équation symbolique, le nom du fichier Matlab pour la fonction ainsi que les noms des variables symboliques de la fonction. Afin d'en apprendre davantage sur cette fonction, veuillez consulter la documentation suivante : https://www.mathworks.com/help/symbolic/matlabfunction.html.

ROS et Simulink

Matlab met en place plusieurs outils fort intéressants pour permettre de créer des nœuds ROS (ROS et ROS2). Pour développer notre contrôleur, nous avons fait usage de la librairie « ROS Toolbox » afin d'être en mesure de créer les modules ROS nécessaires pour notre contrôleur. La communication dans ROS

8.4 se fait principalement à l'aide de messages (Topics) que l'on va « publish » pour envoyer ou « subscribe » pour recevoir. Il est également possible de faire appel à des services ROS qui sont en fait composés de deux messages: un pour la requête et un pour la réponse. De plus, il est possible de jouer un enregistrement des messages transmis sur le réseau ROS que l'on appelle « ROS bag ». Nous terminons avec une dernière fonctionnalité de ROS parmi tant d'autres que nous jugeons importantes est la définition et l'utilisation de paramètres ROS. Ces paramètres peuvent être définis, modifiés et être lus par n'importe quel nœud ROS qui en a besoin. Simulink met à notre disposition une série de blocs utilisables pour communiquer avec ROS. Ces blocs nous permettent d'envoyer et recevoir des messages, faire l'appel à des services, lancer la lecture de « ROS bag » ainsi que modifier et lire les paramètres ROS.

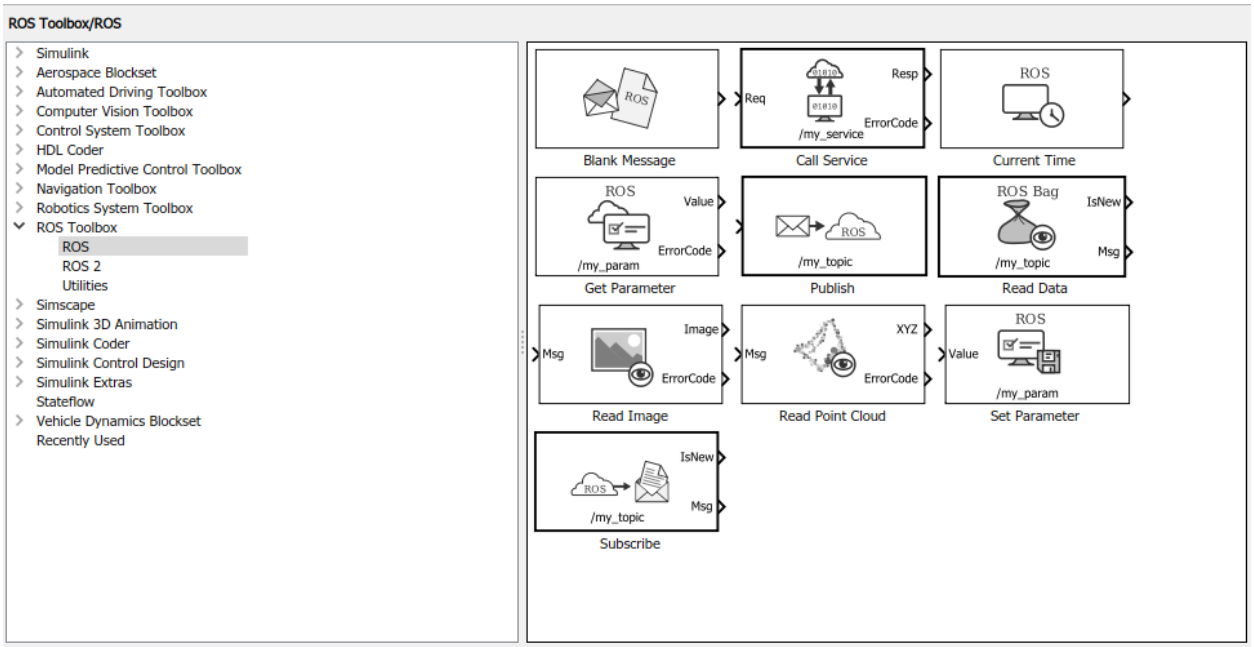


Figure 30: Librairie ROS Toolbox (Matlab 2020a)

Cette librairie a permis de faciliter le développement de ce projet puisque nous n'avons pas eu à faire tout le code nécessaire pour la gestion des interactions avec ROS. Pour l'instant, nous travaillons surtout avec les blocs « Subscribe » et « Publish » pour recevoir envoyer des messages sur le réseau ROS. Nous allons également faire l'intégration de paramètres ROS pour assurer la modularité du contrôle sur différentes plateformes dans une prochaine phase du projet.

8.4.1 *Subscriber*

Voici un exemple de « subscriber » que nous avons fait pour recevoir un message.

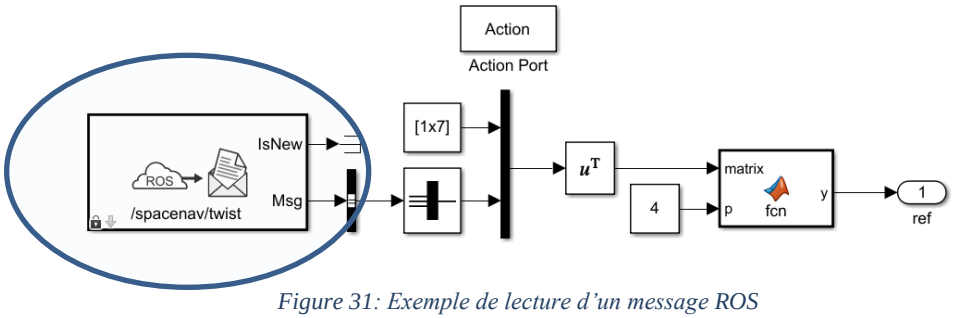


Figure 31: Exemple de lecture d'un message ROS

L'exemple ci-haut montre le schéma Simulink d'un sous-système qui permet de lire la commande d'une SpaceMouse de 3DConnection. Cette souris nous permet de faire le contrôle manuel du sous-marin. Dans cet exemple, nous recevons un message (Topic) nommé « /spacenav/twist » de type « geometry_msgs/Twist » que nous interprétons ensuite pour donner en référence au contrôle. Nous allons discuter plus en détail du contrôle en mode manuel dans la section sur les fonctionnalités du contrôleur. Voici à quoi ressemblent les configurations du bloc « Subscribe ».

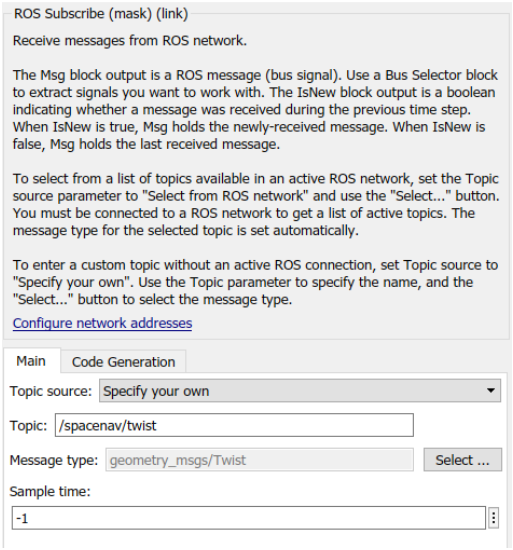


Figure 32: Configurations d'un bloc « Subscribe » (Matlab 2020a)

Tout d'abord, tel que spécifié plus haut, ce bloc permet de faire la lecture de messages sur le réseau ROS. Pour configurer ce bloc, il suffit d'écrire le nom du Topic ainsi que de spécifier le type du message. Si un « ROS Core » est présentement démarré sur votre PC de travail, il est également possible de sélectionner un Topic déjà existant sur le réseau.

Publisher

Voici un exemple de « publisher » que nous avons mis en place dans notre contrôleur afin d'écrire un message sur le réseau ROS.

8.4.2

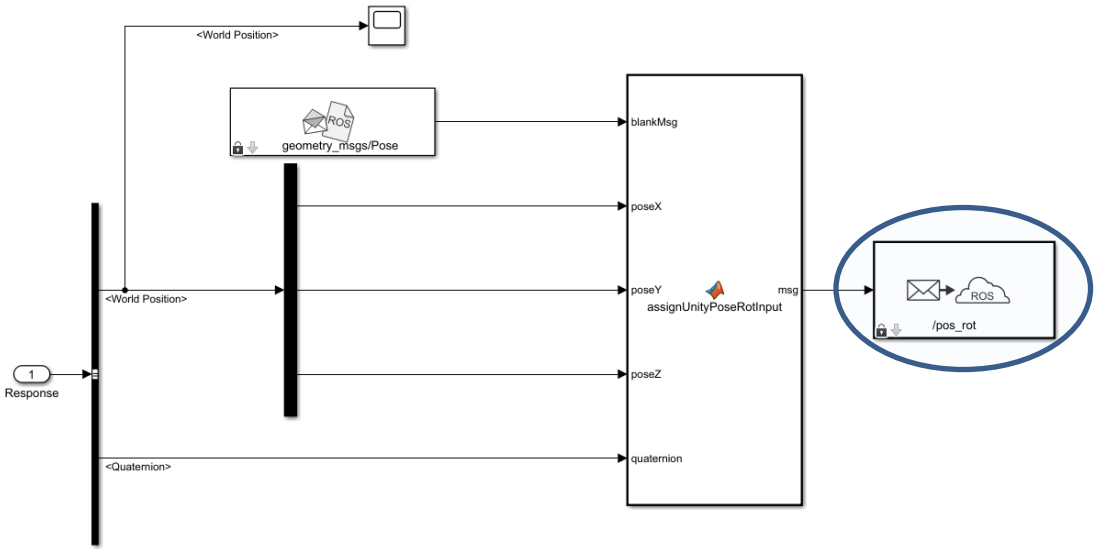


Figure 33: Exemple d'écriture d'un message ROS

L'exemple ci-haut présente le schéma Simulink qui permet de faire l'envoi de la position linéaire ainsi que l'orientation du sous-marin au simulateur. On fait l'envoi d'un message de type « geometry_msgs/Pose » au topic « /pos_rot » pour faire bouger le sous-marin en simulation.

Voici à quoi ressemblent les configurations du bloc « publish »

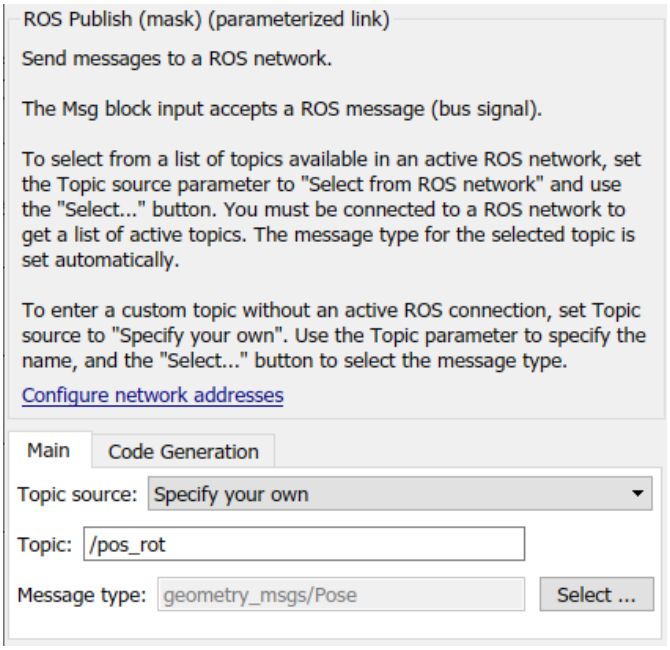


Figure 34: Configurations du bloc « Publish » (Matlab 2020a)

Ce bloc permet de faire l'écriture de messages sur le réseau ROS. Pour configurer ce bloc, il suffit d'écrire le nom du Topic ainsi que de spécifier le type du message. Si un « ROS Core » est présentement démarré sur votre PC de travail, il est également possible de sélectionner un Topic déjà existant sur le réseau pour l'attribuer au bloc.

Pour plus d'informations sur l'utilisation de ROS avec Simulink, voici la page de support de la « ROS Toolbox » : https://www.mathworks.com/hardware-support/robot-operating-system.html

Messages personnalisés

La plupart des messages que nous avons utilisés dans le cadre de ce projet sont des messages standards provenant de ROS. Matlab met à disposition un outil nécessaire pour permettre la création de messages personnalisés formés de types différents. L'utilisation de ce genre de message est une pratique courante pour développer et utiliser des messages qui ne sont pas natifs à ROS. Dans ce projet, nous avons justement à faire à l'utilisation de messages personnaliser pour un type bien particulier dont nous avons besoin. Nous avons créé un message de type « /proc_control_matlab/AddPose » qui permet de faire l'envoi d'une nouvelle coordonnée à la génération de trajectoire. Il n'existe présentement aucun message natif de ROS pour contenir l'information nécessaire pour notre message. Nous allons parler plus en détail de l'utilisation de ce message dans la section sur les fonctionnalités du contrôleur.

8.5

Avant de débuter la création d'un message personnalisé, assurez-vous d'installer le « Add-On » pour le support des messages personnalisés. Ce « Add-On » est présentement fonctionnel pour les versions de Matlab 2017a à 2020a et il ne sera plus nécessaire d'utiliser ce dernier avec la version 2020b. Voici la page du « Add-On » : https://www.mathworks.com/matlabcentral/fileexchange/49810-ros-toolbox-interface-for-ros-custom-messages.

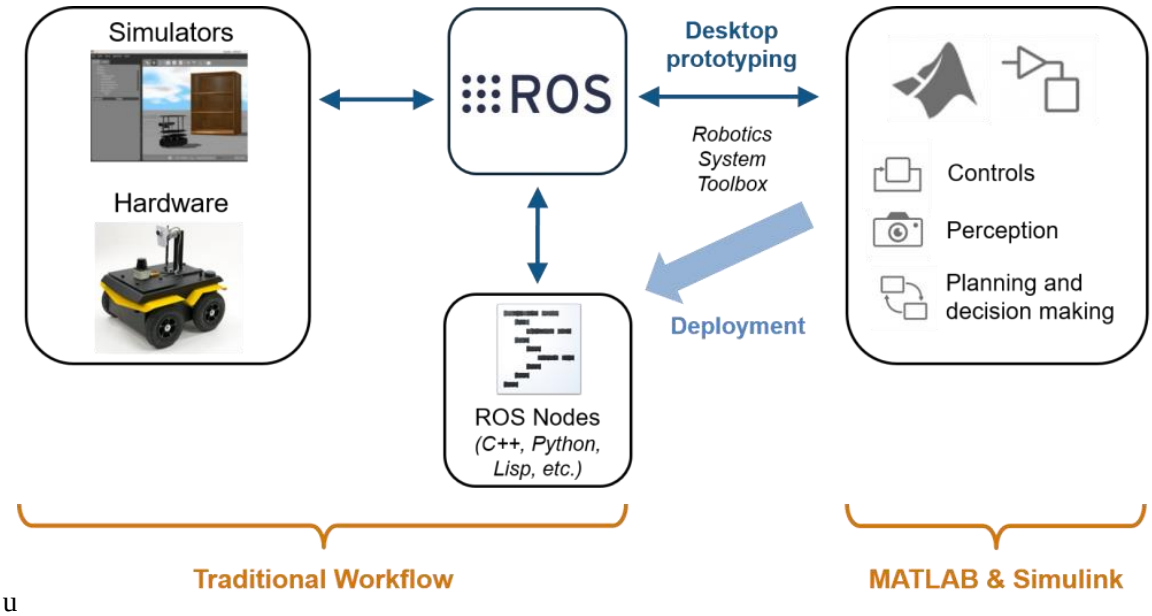
1. Premièrement, vous pouvez créer le fichier « .msg » ou « .srv » nécessaire pour faire la déclaration du message ou du service. Pour plus d'informations, vous pouvez consulter la documentation suivante sur la création de messages et de services : https://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv
2. Deuxièmement, compilez le « Catkin Workspace » pour générer les nouveaux messages pour ROS.
3. Troisièmement, suivez la procédure documentée dans la documentation de Matlab suivante pour générer les messages pour Matlab : https://www.mathworks.com/help/ros/ug/ros-custom-message-support.html.

Nous recommandons le plus possible de faire usage de messages standards lors du développement de nœuds ROS dans Simulink, mais sachez qu'il est tout de même possible de créer des messages supplémentaires lorsque nécessaire.

Déploiement sur le sous-marin

Dans cette section, nous discuterons de deux options qui s'offrent à nous pour faire le déploiement du logiciel sur le sous-marin. Après avoir fait le développement des différents nœuds ROS dans Simulink, il est maintenant temps de faire le déploiement du projet sur le sous-marin ou localement sur nos ordinateurs
8.6 de travail pour effectuer des tests. Le déploiement est une partie très importante dans ce projet puisqu'elle constitue l'étape finale de l'implémentation logicielle. C'est l'étape qui nous permet d'obtenir le code C/C++ qui sera compilé et exécuté sur le sous-marin. Pour réaliser cela, vous devrez avoir installé les « add-ons » suivants : « Embedded Coder », « Simulink Coder » et « Matlab Coder ».

Voici un résumé imagé des deux méthodes que nous allons décrire dans cette section.



u

Figure 35: Diagramme des méthodes de déploiement [13]

[13] Castro, tirée du blog de MathWorks (2017)

Desktop Prototyping

Nous voulons que le déploiement soit simple et rapide pour être en mesure d'effectuer des changements dans notre Simulink lors de tests en piscine pour ensuite tester directement ces changements. La première méthode que nous présentons nous apporte justement ces critères. Simulink, à l'aide de la

8.6.1 librairie « ROS Toolbox » nous permet de nous connecter à notre sous-marin via une connexion SSH (Secure Shell Protocol) pour déployer directement le code généré et l'exécuter ensuite. Cette fonctionnalité permet également de voir en temps réel les informations dans les « Scope » que nous avons placés dans Simulink pour connaître de l'information provenant des capteurs. Cette fonctionnalité est appelée « Monitor and Tune ».

Utilisation

Afin de pouvoir utiliser le « Monitor and Tune », il est nécessaire de s'assurer d'être en mesure de

8.6.2 se connecter avec une connexion SSH sur le robot. Pour se faire, il faut s'assurer que le robot ait un serveur SSH fonctionnel et que le client sur lequel Simulink est installé puisse se connecter au robot. Comme nous l'avons mentionné plus haut dans cette section sur l'implémentation logicielle, les nœuds ROS que nous développons doivent fonctionner à l'intérieur de « containers » Docker. Pour cette raison, nous avons dû faire la création d'un conteneur spécifique pour le « Monitor and Tune » qui contient le serveur SSH pour se connecter et déployer le code afin d'effectuer des tests. Voici la méthode pour se connecter au sous-marin. Assurez-vous d'avoir démarré l'ensemble des conteneurs Docker sur le sous-marin pour être en mesure de faire le déploiement. Parmi ces conteneurs se trouve « simulink-monitor » qui permet la connexion au sous-marin via Simulink.

Figure 36: Barre d'outils « Robot » (Matlab 2020a)

Dans la barre d'outils de l'onglet « Robot », vous pouvez appuyer sur l'option « Connect to Robot ».

Une fenêtre s'affichera ensuite pour connaître les informations du robot ainsi que de ROS.

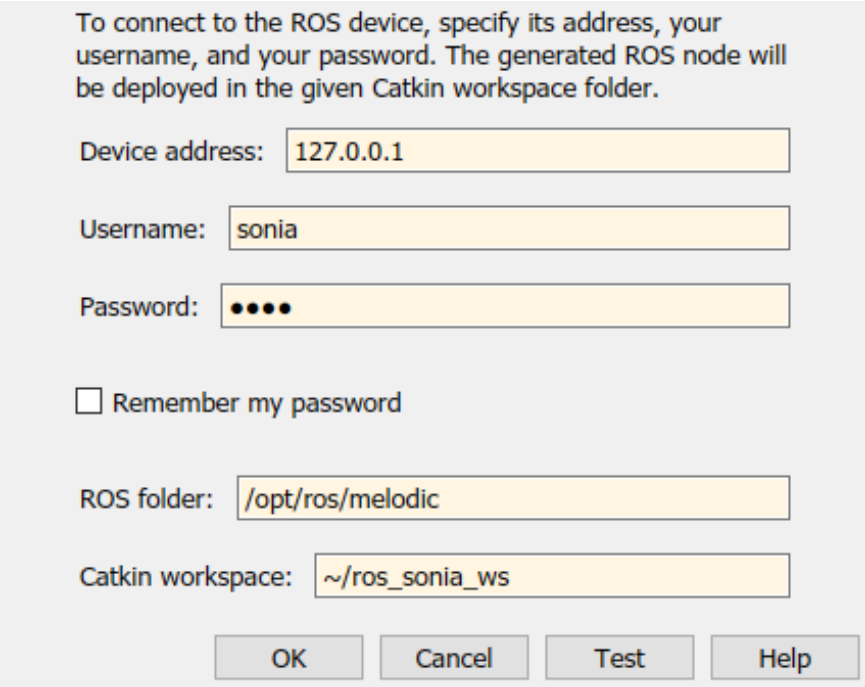


Figure 37 : Fenêtre de connexion au robot (Matlab 2020a)

Dans cette fenêtre, il suffit d'entrer l'adresse IP du sous-marin ou de votre ordinateur local, le nom d'utilisateur dans le conteneur, le mot de passe de ce dernier, le chemin vers les fichiers de ROS et le chemin vers le « Catkin Workspace » dans lequel le nœud ROS sera compilé sur le conteneur. Ensuite, vous pouvez effectuer un test de connexion pour valider les informations entrées. Veuillez noter que la connexion SSH doit se faire avec le port 22.

Une fois la connexion validée et fonctionnelle, il est maintenant possible d'utiliser « Monitor and Tune » pour déployer un nœud ROS que vous désirez tester sur le sous-marin. Pour ce faire, vous avez simplement à appuyer sur l'option « Monitor and Tune » dans la barre d'outils « Robot » pour déployer et démarrer le projet directement sur le sous-marin.

Il existe également une fonctionnalité supplémentaire provenant de « Monitor and Tune » qui permet de se reconnecter à un nœud opérationnel que nous avons précédemment déployé. Vous pouvez trouver l'option dans la barre d'outils sous le bouton « Monitor and Tune ».

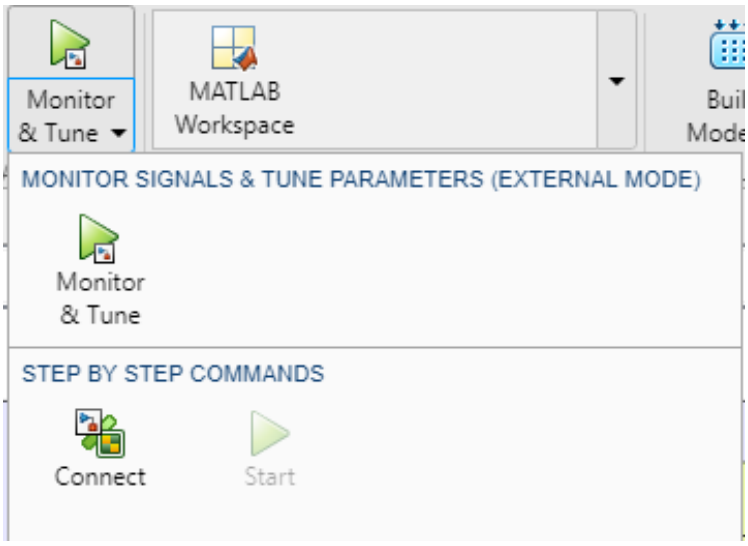


Figure 38 : Options supplémentaires « Monitor and Tune » (Matlab 2020a)

Déploiement pour mise en production

Lorsque nos tests seront effectués et que nous désirerons déployer les nœuds ROS sur le sous-marin, nous allons procéder à ce que nous appelons la mise en production. Nous allons particulièrement utiliser cette méthode pour générer du code testé à l'aide de « Monitor and Tune » et qui est fonctionnel.

8.6.3 Dans ce cas-ci, Simulink offre la possibilité de générer le code C/C++ que nous allons par la suite mettre dans un conteneur Docker prévu à cet effet. Une image de ce conteneur pourra par la suite être utilisée sur le sous-marin pour exécuter le contrôleur que nous avons développé à l'aide de Simulink.

Embedded Coder

Cette deuxième méthode requiert l'utilisation de l'extension « Embedded Coder » pour Simulink

8.6.4 afin de générer le nœud ROS en C/C++. Le code généré sera mis dans le conteneur prévu pour ensuite créer une image Docker utilisable sur le sous-marin. Voici comment nous pouvons déployer le code à l'aide de Simulink et de « Embedded Coder ».

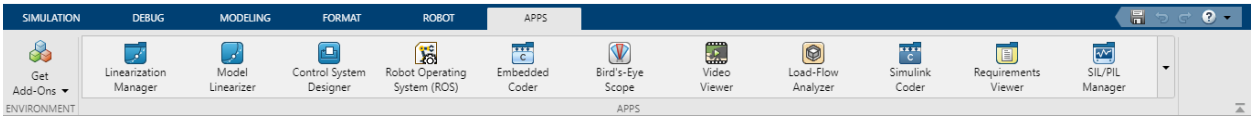


Figure 39 : Barre d'outils « Apps » (Matlab 2020a)

L'option « Embedded Coder » permet de faire la génération du code du schéma Simulink. L'onglet suivant devrait apparaître pour vous permettre de générer votre code.

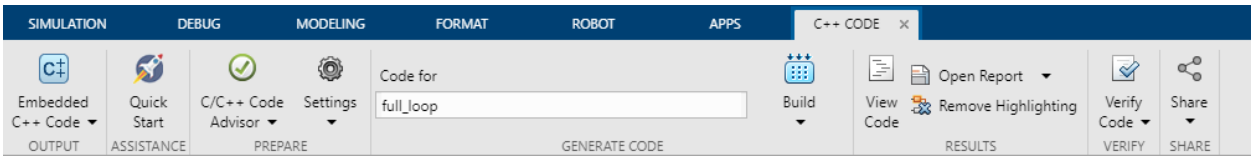


Figure 40 : Barre d'outils « C++ Code » (Matlab 2020a)

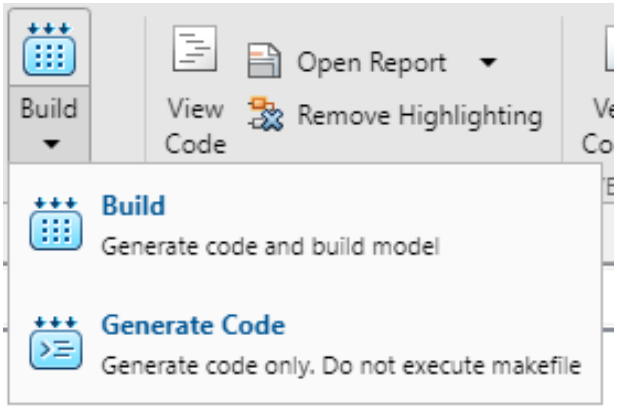


Figure 41 : Options de « build » (Matlab 2020a)

Pour générer le code, vous pouvez simplement appuyer sur « Build » pour générer et compiler le

code ou « Generate Code » pour simplement générer le code C++ sans effectuer de compilation dessus. Simulink produira une archive qui contient tout le code nécessaire pour la compilation. Des instructions à suivre pour terminer seront également affichées dans le « Diagnostic Viewer » de Simulink. Pour finir, il faut simplement prendre le code généré pour le déposer ensuite dans le conteneur Docker. Une image pourra alors être créée pour être déployée sur le sous-marin.

8.7 Ressources

Cette section sur l'implémentation logicielle parcourt l'essentiel de ce que nous avons fait comme travail pour développer le logiciel de notre contrôleur. Évidemment, il était impossible de décrire tous les concepts et outils en détail dans ce rapport. C'est pourquoi nous faisons une liste de quelques ressources qui pourraient améliorer votre compréhension lors de la lecture de ce rapport.

1. Documentation de S.O.N.I.A : https://wiki.S.O.N.I.A.etsmtl.ca/
2. Documentation de ROS : http://wiki.ros.org/Documentation
3. Documentation de Docker : https://docs.docker.com/
4. Documentation pour SSH : https://www.ssh.com/academy/ssh#the-ssh-protocol
5. Documentation pour ROS sur Simulink : https://www.mathworks.com/hardware-support/robot-operating-system.html

Pour conclure cette section sur l'implémentation logicielle, nous avons parcouru l'architecture logicielle de S.O.N.I.A, les choix technologiques que nous avons effectués, la génération du code pour notre modèle physique, l'utilisation de ROS dans Simulink ainsi que les différentes méthodes de déploiement du code sur le sous-marin. Nous tenons à mentionner que l'utilisation de Matlab et de Simulink a grandement facilité notre développement et les outils qu'ils ont mis en place améliorent également la qualité de notre développement. Nous avons été capables de développer un logiciel qui fonctionne bien avec l'architecture de S.O.N.I.A et les technologies utilisées par l'équipe.

9 Fonctionnalité du contrôleur

Dans cette section, nous allons passer en revue les différentes fonctionnalités que nous avons développées pour et autour de notre contrôleur. Ces fonctionnalités permettent d'améliorer grandement le contrôleur. Nous avons fait la création de deux modes de contrôle et nous allons discuter de modes que nous aimerions dans le futur. Pour terminer, nous allons discuter de l'implémentation de la connexion à deux simulateurs que nous utilisons.

Modes du contrôleur

Afin de faciliter l'utilisation du contrôleur, nous avons fait la création de deux modes qui permettent
9.1 de donner une référence au contrôleur. Le premier mode permet de faire la génération d'une trajectoire en donnant les différents points par lesquels nous désirons que cette dernière passe. Le deuxième mode consiste plutôt à contrôler le sous-marin en lui donnant des vitesses à l'aide d'une SpaceMouse de 3DConnection. Pour faire le changement entre les modes, nous avons créé un message ROS. Il suffit d'envoyer ce message au contrôle afin que ce dernier s'occupe de changer le mode entre la génération de trajectoire ou le contrôle manuel. Voici les informations importantes à connaître pour ce message.

Tableau 8 : Message ROS pour le changement de mode du contrôleur

Topic	Type	Définition	Valeurs
/proc_control/set_mode	std_msgs/UInt8	data (uint8)	1 : Mode manuel (SpaceNav)
			2 : Mode trajectoire

9.1.1 À tout moment, il est possible d'envoyer ce message pour faire le changement de mode.

Génération de trajectoire

Les trajectoires sont une excellente façon de donner une référence à notre contrôleur pour se rendre à une position ou des positions données et c'est pourquoi nous avons opté pour un système de génération de trajectoires pour assurer un suivi de trajectoire désirée. Nous voulions permettre à quiconque qui utilise le contrôle de pouvoir donner une liste de points afin de générer une trajectoire passant par ces points, mais aussi de donner des paramètres à respecter pour chacun de ceux-ci. Nous allons donc séparer cette section en quelques parties pour expliquer le fonctionnement de la génération de trajectoire. Dans la première partie, nous allons discuter des messages ROS à utiliser pour faire fonctionner la génération de trajectoire. Dans la seconde, nous allons présenter la fonction qui permet de traiter les points envoyés. La troisième partie traitera de la génération de trajectoire et la quatrième et dernière partie portera sur l'envoi des points en référence au contrôleur ainsi que le calcul d'obtention de la position voulue.

Messages ROS

Afin d'interagir avec le système de génération de trajectoires, nous avons mis en place une série de messages ROS que nous allons expliquer en détail ci-bas.

Le premier message est l'un des plus importants pour la génération de trajectoires puisque c'est le message qui permet d'envoyer des points à la trajectoire pour avant de la générer. Ce message est justement un exemple de message personnalisé que nous avons créé afin d'avoir un type de message qui n'existait pas parmi les messages standards de ROS. Voici les informations pour ce message.

Tableau 9 : Message ROS pour l'envoi de points

Topic	Type	Définition	Valeurs
/proc_control_matlab/add_pose	S.O.N.I.A_common/AddPose	position (geometry_msgs/Point)	Point (x, y, z)
		orientation (geometry_msgs/Vector3)	Vector3 (x, y, z)
		frame (uint8)	0 : Position absolue et angle absolu
			1 : Position relative et angle relatif
			2 : Position relative et angle absolu
			3 : Position absolue et angle relatif
		speed (uint8)	Vitesse
		fine (float64)	Précision du passage par le point

Le tableau ci-haut mentionne chacune des valeurs nécessaires pour faire l'ajout d'un point. À titre informatif, le « fine » ainsi que le « speed » ne sont pas encore implémentés dans notre génération de trajectoire pour l'instant, mais le seront certainement dans une version future.

Le prochain message permet de lancer la génération de trajectoire avec tous les points ajoutés pour créer la trajectoire envoyée au contrôleur. Dans ce cas-ci, nous avons utilisé un message standard de ROS.

Tableau 10 : Message ROS pour générer la trajectoire

Topic	Type	Définition	Valeurs
/proc_control_matlab/compute_trajectory	std_msgs/Bool	data (bool)	true : Génération
			false : Non utilisé

Le message suivant, quant à lui, permet de vider la liste des points en attente d'être utilisés pour générer la trajectoire. Lorsque l'on envoie un message pour générer la trajectoire, la liste des points est automatiquement vidée également. Voici les informations de ce message.

Tableau 11 : Message ROS pour vider la liste de points

Topic	Type	Définition	Valeurs
/proc_control_matlab/clear_waypoints	std_msgs/Bool	data (bool)	true : Vider la liste
			false : Non utilisé

Afin de savoir si le sous-marin à atteint la position finale demandée, nous avons également mis en place un message. Voici le message envoyé par le générateur de trajectoires qui permet de connaître si la cible est atteinte ou non.

Tableau 12 : Message ROS pour connaître l'arrivée à la cible

Topic	Type	Définition	Valeurs
/proc_control_matlab/target_reached	std_msgs/Bool	data (bool)	true : Arrivé
			false : Non arrivé

Traitement des points

La logique pour l'ajout des points a été programmée dans une classe héritant de « matlab.System » que l'on a appelé « AddPose ». Cette classe regroupe toutes les méthodes nécessaires à l'ajout d'un point. Le bloc d'ajout de position prend 6 entrées : l'information pour savoir si on a généré la trajectoire, l'information pour savoir si on veut vider la liste de points en attente, l'information pour savoir si on a reçu un nouveau point, le point lui-même, les conditions initiales et un reset. Les 2 sorties du bloc sont le vecteur de points ainsi que le nombre de points à l'intérieur de ce dernier.

Chaque fois qu'un nouveau point est ajouté par un utilisateur ou une mission, par exemple, le point doit être modifié pour être envoyé à la génération de trajectoire. Tout d'abord, on doit convertir l'orientation du point en quaternion. Ensuite, on doit transformer le point en fonction du « frame » spécifié dans le message envoyé.

Si la trajectoire a été générée et/ou le message pour vider la liste de points a été envoyé, le bloc « AddPose » videra la liste des points en attente pour être envoyé à la génération de trajectoire.

Voici le schéma du bloc « AddPose » que nous avons conçu.

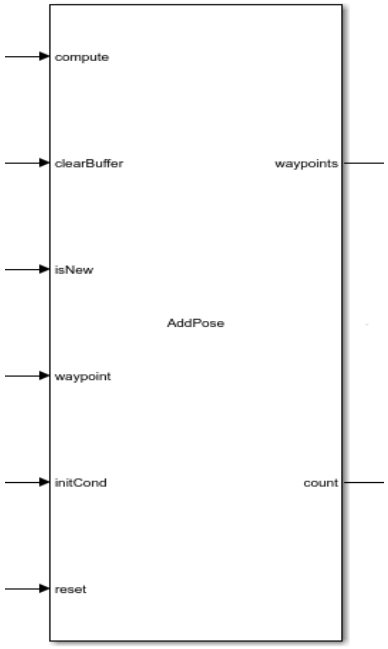


Figure 42 : Bloc « AddPose »

Génération de trajectoire

La logique pour la génération de trajectoires a également été programmée dans une classe héritant de « matlab.System » que l'on a appelé « TrajectoryGenerator ». Ce bloc permet de faire la génération d'une trajectoire à partir de la liste de point qui a été envoyés et ensuite traités par « AddPose » lorsque le message pour générer la trajectoire est envoyé. Le bloc prend 2 entrées : le vecteur de points provenant de « AddPose » ainsi que le nombre de points. Aussi, les 2 sorties du bloc sont le vecteur des positions générées ainsi que le nombre de positions.

À l'aide du vecteur de points, nous sommes en mesure d'estimer des temps d'arrivés pour chacun des points. Ces temps sont nécessaires pour faire la génération de la trajectoire. Pour ce faire, nous avons fait le calcul du temps entre deux points en faisant une approximation linéaire de la distance entre ces points. Nous calculons la distance linéaire à parcourir pour chacun des axes, calculons ensuite le temps et prenons le plus lent pour estimer le temps d'arriver à ce point.

Une fois le vecteur des temps d'arrivées déterminé, nous avons fait usage d'une fonction Matlab pour générer la trajectoire. La fonction que nous avons utilisée se nomme « waypointTrajectory ». Cette fonction prend le vecteur de points linéaires, le vecteur de temps d'arrivées, le temps d'échantillonnage et le vecteur des orientations en quaternion. Pour plus d'informations sur la fonction utilisée, voici la page de documentation de celle-ci : https://www.mathworks.com/help/nav/ref/waypointtrajectory-system-object.html. Cette fonction va générer la trajectoire en fonction de tous ses paramètres que nous allons ensuite envoyer en référence au contrôleur via le bloc « TrajectoryManager ». La fonction va extraire les positions linéaires, les quaternions les vitesses linéaires, les accélérations linéaires ainsi que les vitesses angulaires à chacun des points de la trajectoire. Seules les accélérations ne seront pas envoyées en référence au contrôleur.

Voici le schéma bloc de « TrajectoryGenerator ».

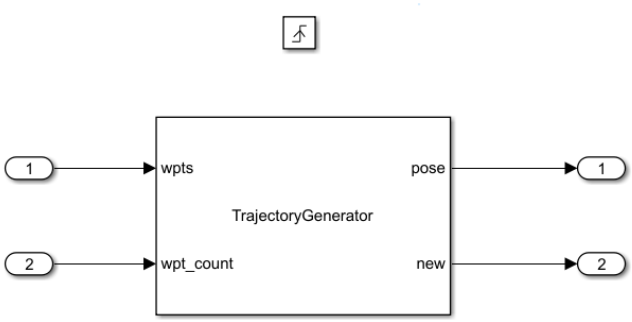


Figure 43 : Bloc « TrajectoryGenerator »

Nous avons mis ce bloc à l'intérieur d'un sous-système de type « trigger », car nous voulions que ce dernier soit exécuté seulement lorsque nous voulons générer la trajectoire.

Envoi de la référence

La dernière partie de notre générateur de trajectoire consiste à faire l'envoi des données provenant de « TrajectoryGenerator ». Nous avons également programmé la logique de ce bloc dans une classe héritant de « matlab.System » que l'on a appelé « TrajectoryManager ». Ce bloc prend en paramètre la description de la position et de l'orientation du sous-marin (pose), les données provenant de la génération de trajectoire, de l'information sur les nouvelles données générées reçues et un reset. Les sorties sont un vecteur de poses, ainsi que l'information pour savoir si nous avons atteint notre position finale.

Lorsque ce bloc reçoit des nouvelles données de la génération de trajectoire, il les ajoute à l'intérieur d'un grand vecteur qui regroupe l'ensemble des références à envoyer au contrôleur. À chaque instant, le bloc envoie un vecteur de poses de la dimension du nombre de prédictions que nous désirons envoyer au contrôleur.

Lorsque la trajectoire est terminée, nous vérifions également que le sous-marin est arrivé à la pose finale voulue. Nous comparons la pose donnée par les capteurs avec la dernière pose restante dans le vecteur à la fin de la trajectoire. Pour ce faire, nous avons choisi de faire une vérification à l'aide d'une sphère et d'un cône pour évaluer respectivement la position linéaire ainsi que le quaternion. Si le sous-marin reste durant un certain temps à l'intérieur des zones de convergence, nous considérons que ce dernier a bel et bien atteint sa cible et qu'il est arrivé au point final de la trajectoire désirée.

Voici le schéma du bloc « TrajectoryManager ».

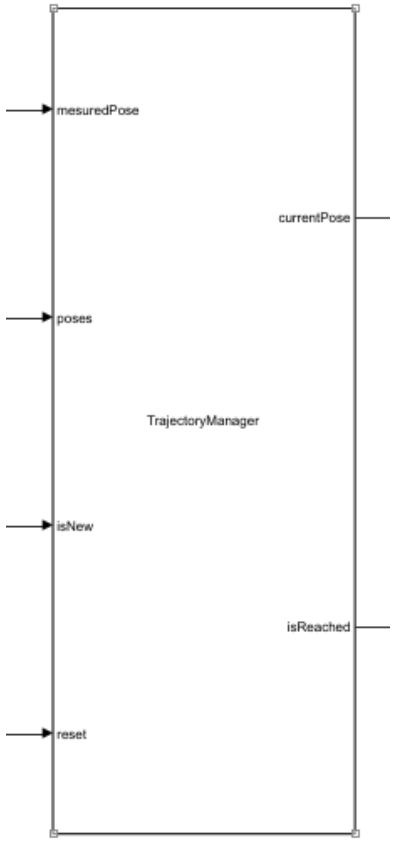


Figure 44 : Bloc « TrajectoryManager »

Tous ces blocs nous ont permis d'obtenir un générateur de trajectoires fonctionnel qui permet de faire l'envoi de points par lesquels nous voulons que le sous-marin passe pour atteindre la cible voulue. Nous avons pu effectuer beaucoup de tests en donnant plusieurs trajectoires consécutives afin de tester l'ensemble des blocs. Nous avons créé des « scripts » en Python pour tester le fonctionnement de nos messages ROS et ainsi interagir avec le générateur de trajectoire. Nous avons obtenu de très bons résultats que nous présenterons plus en détail dans la section sur l'analyse et les résultats dans ce présent rapport. Juste avant de discuter du prochain mode de contrôle, voici l'ensemble du schéma pour la génération de trajectoire.

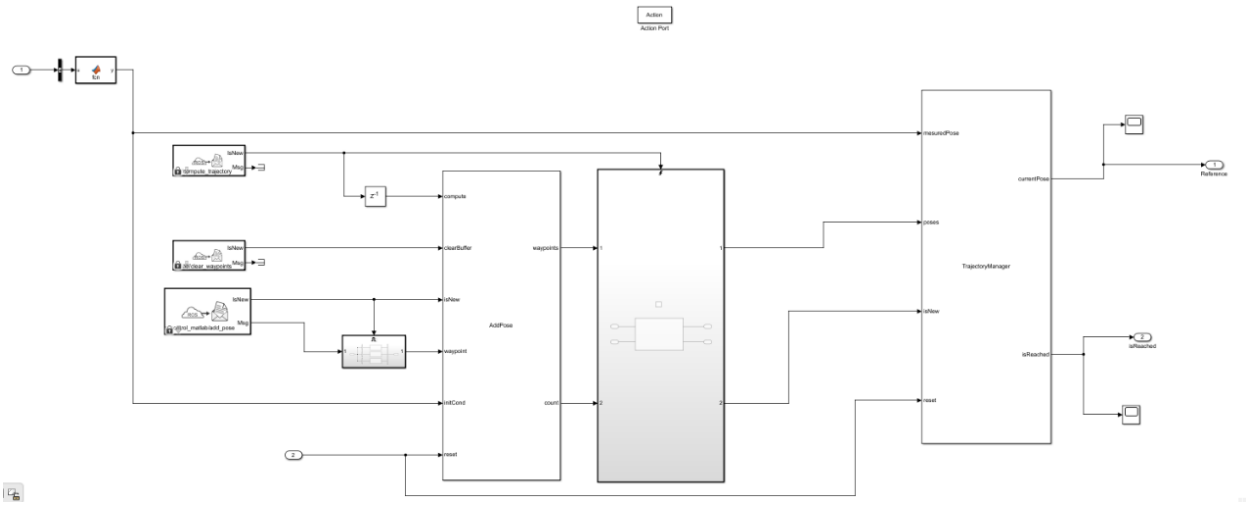


Figure 45 : Schéma global de la génération de trajectoire

Ce schéma regroupe tous les blocs que nous passés en revu dans cette section. Vous pouvez remarquer que ce sous-système a été mis dans un « Action Subsystem » afin qu'il soit exécuté seulement si le mode pour la génération de trajectoires est actif.

9.1.2 *Contrôle avec SpaceNav*

Le mode de contrôle manuel est un peu plus simple que la génération de trajectoire, mais également très pertinent pour le contrôle du sous-marin. En effet, ce mode nous permettra de bouger le sous-marin dans l'eau à l'aide d'une SpaceMouse de 3DConnexion. Ce mode pourra nous être très utile lors de tests piscines pour faire l'acquisition de données visuelles nécessaires pour l'entraînement de modèles pour la détection d'images. Ce mode peut aussi simplement nous permettre de nous déplacer à un endroit précis sans avoir recours aux plongeurs pour changer la position du sous-marin dans l'eau. Les périodes en test piscine seront les seuls moments où nous allons utiliser ce mode de contrôle, car nous rappelons que le sous-marin se veut être un sous-marin autonome.

SpaceMouse

La SpaceMouse de 3DConnexion est généralement utilisée pour se déplacer dans les six degrés de liberté à l'intérieur d'un environnement 3D tel que les logiciels de CAO (conception assistée par ordinateur) par exemple. Il existe également 2 boutons que l'on peut programmer pour différentes fonctionnalités. Voici à quoi ressemble une SpaceMouse de 3DConnexion.

Figure 46 : SpaceMouse de 3DConnexion [14]

Nous avons choisi d'utiliser une SpaceMouse pour sa facilité d'utilisation et ainsi que la rapidité à intégrer la souris avec notre contrôleur. ROS offre déjà un nœud qu'il est possible d'installer pour pouvoir utiliser la SpaceMouse directement sans avoir à programmer notre propre interface. Voici la documentation qui permet d'en connaître davantage sur l'installation et l'utilisation du nœud ROS : http://wiki.ros.org/spacenav_node.

[14] 3DConnexion, Tirée du site de 3DConnexion

Messages ROS

ROS à déjà implémenter les messages utilisables pour connaître les informations de la souris tels que la position des 6 degrés de liberté et l'état des boutons. Voici le message que nous avons utilisé dans notre cas pour créer le mode de contrôle manuel.

Tableau 13 : Message ROS pour le contrôle manuel

Topic	Type	Définition	Valeurs
/spacenav/twist	geometry_msgs/Twist	linear (geometry_msgs/Vector3)	Vector3 (x, y, z)
		angular (geometry_msgs/Vector3)	Vector3 (x, y, z)

Ce message nous donne de l'information sur les composantes linéaires et angulaires de la position du joystick de la souris. La valeur envoyée se situe entre -1 et 1 pour chacun des axes. Nous allons donc utiliser cette valeur pour représenter la vitesse du sous-marin dans chacun des axes pour ainsi contrôler le sous-marin avec une référence en vitesse.

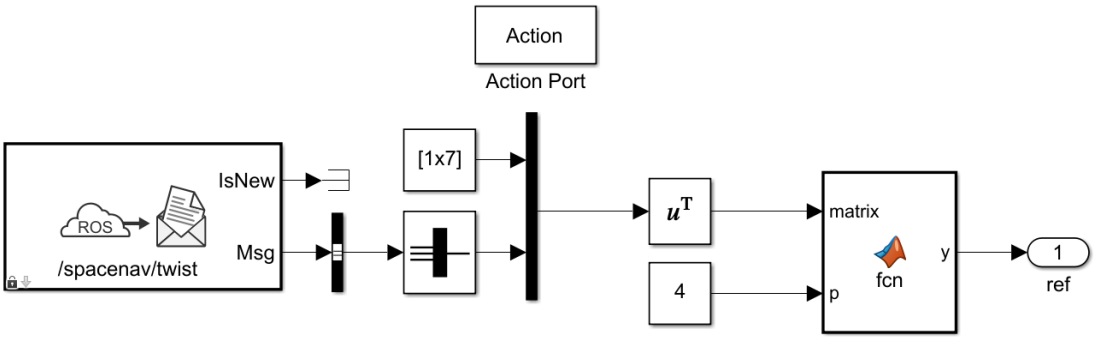


Figure 47: Contrôle manuel

Notez bien que la fonction Matlab utilisée dans le schéma ci-haut permet d'envoyer un vecteur de pose de la dimension du nombre de prédictions en référence au contrôleur. L'utilisation de deux modes de contrôle nous permet d'être plus flexibles quant à notre façon de contrôler le sous-marin.

9.2 Modes futurs

Après avoir fait le développement d'un mode pour la génération de trajectoire et d'un mode manuel, il existe également deux modes que nous aimerions développer dans les futures versions de notre contrôleur. Ces modes permettraient d'améliorer énormément notre contrôleur et aider à effectuer des tâches de la compétition. Pour chacun de ces modes futurs, nous allons décrire en quoi il consiste ainsi que leurs avantages.

9.2.1 *Mode vision*

Le mode vision est un mode qui permettrait de faire l'alignement du sous-marin à l'aide de données visuelles qu'il obtient des caméras. Nous sommes en mesure, à l'aide d'un outil appelé « proc_image_processing » (développé par S.O.N.I.A) de déterminer la position (dans l'image) de l'objet que nous détectons. Du travail est également en cours afin d'estimer la distance qui nous sépare de l'objet en sachant préalablement sa dimension réelle. Ce mode nous permettrait d'intégrer toute la logique pour l'alignement avec des données visuelles à l'intérieur même du contrôleur. Cela éviterait de faire ce travail à l'intérieur des missions qui demandent de l'alignement en vision et le processus de conception des missions serait alors réduit et simplifié.

9.2.2 *Mode hydrophone*

Le mode hydrophone est un mode qui nous permettrait de faire l'alignement du sous-marin à l'aide des données acoustiques obtenues à l'aide des hydrophones. Tel que mentionné dans la section sur les tâches d'alignement, les hydrophones nous permettent de déterminer les angles de direction et d'élévation de la source acoustique. Ces données pourraient être utilisées pour aligner le sous-marin vers la source afin de la rejoindre pour réussir des missions qui demandent de faire de l'alignement avec les hydrophones. Du travail est présentement en cours au sein de S.O.N.I.A pour la revue de l'implémentation des hydrophones sur les sous-marins (AUV7 et AUV8). Ce mode permettrait, tout comme le mode vision, d'intégrer la logique de l'alignement avec hydrophones pour éviter de le faire directement à l'intérieur des missions.

Connexion à un simulateur

Cette dernière fonctionnalité du contrôleur nous a été grandement utile et sera grandement utile pour l'ensemble de l'équipe de S.O.N.I.A. En effet, les simulateurs sont extrêmement importants pour les projets d'asservissement afin de visualiser des résultats. Les simulateurs prennent également une grande
9.3 place dans le domaine de la robotique puisqu'ils peuvent également permettre aux équipes de développement de poursuivre leur travail sans avoir accès directement au matériel. S.O.N.I.A développe actuellement une simulation qui va nous permettre de tester les missions pour la compétition dans un environnement virtuel qui reflète le plus possible le TRANSDEC. Durant ce projet, Simulink a permis d'effectuer énormément de simulations pour tester certaines parties de notre contrôle. Afin de faciliter le plus possible la visualisation des données provenant du contrôleur, nous avons choisi de faire l'utilisation de deux simulateurs conçus avec Gazebo et Unity.

Unity

9.3.1 Le nouveau simulateur sur lequel l'équipe logicielle travaille est fait à l'aide de Unity puisqu'il permet d'avoir un environnement virtuel beaucoup plus réaliste que Gazebo. Nous visons présentement faire usage de ce simulateur pour faire l'entraînement d'un modèle pour la reconnaissance d'images. Unity offre une interface graphique relativement simple pour travailler et permet de développer le simulateur sur différents systèmes d'exploitation.

Évidemment, afin d'être capables de communiquer avec le simulateur conçu sur Unity, nous avons ajouté une partie à notre contrôleur. Nous avons fait usage d'un message ROS pour envoyer au simulateur afin de commander la position et l'orientation du sous-marin. Voici le message utilisé pour cela.

Tableau 14 : Message ROS pour envoi à Unity

Topic	Type	Définition	Valeurs
/pos_rot	geometry_msgs/Pose	position (geometry_msgs/Point)	Point(x, y, z)
		orientation (geometry_msgs/Quaternion)	Quaternion(x, y, z, w)

Voici le schéma Simulink qui permet de faire l'envoi de la position ainsi que l'orientation au simulateur de Unity.

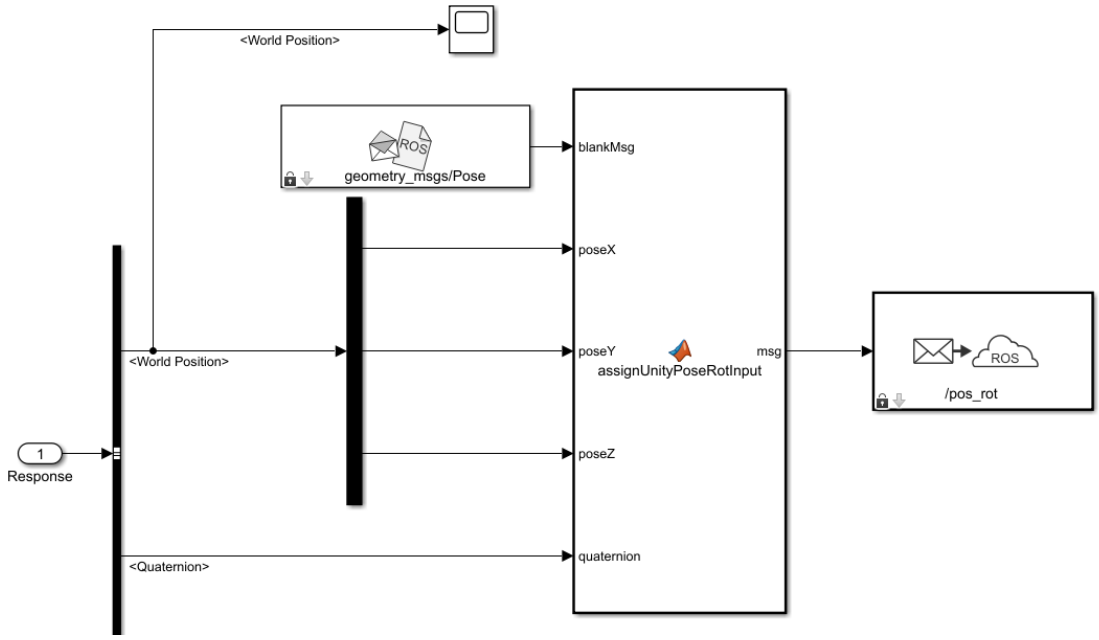


Figure 48 : Schéma bloc pour la connexion à Unity

La fonction Matlab que nous retrouvons sur le schéma ci-haut permet d'attribuer les données provenant du contrôleur à un message ROS pour être envoyé grâce au « publisher ».

Un autre message ROS important a été utilisé pour la connexion avec Unity pour recevoir la condition initiale du simulateur pour informer le contrôleur. Ce message sera utilisé pour permettre à l'utilisateur du simulateur de déplacer sous-marin à l'intérieur de l'environnement tout en donnant l'information au contrôleur pour assurer le bon fonctionnement de ce dernier. Le message est envoyé au démarrage du simulateur.

Tableau 15 : Message ROS reçu de Unity

Topic	Type	Définition	Valeurs
/initial_condition	geometry_msgs/Pose	position (geometry_msgs/Point)	Point(x, y, z)
		orientation (geometry_msgs/Quaternion)	Quaternion(x, y, z, w)

Gazebo

Étant donné que le développement du simulateur sur Unity est toujours en cours, nous avons utilisé Gazebo comme simulateur pour effectuer certains de nos tests. Gazebo est très utilisé dans le domaine de la robotique pour sa facilité d'utilisation avec ROS. Gazebo offre déjà une série de messages disponibles pour interagir avec ROS. Vous pouvez trouver de la documentation sur les messages et services offerts pour Gazebo au lien suivant : http://docs.ros.org/en/jade/api/gazebo_msgs/html/index-msg.html. Voici le message que nous utilisons pour commander la position et l'orientation du sous-marin.

9.3.2

Tableau 16 : Message ROS pour envoi à Gazebo

Topic	Type	Définition	Valeurs
/gazebo/set_model_state	gazebo_msgs/ModelState	model_name (string)	« auv8 »
		pose (geometry_msgs/Pose)	Point(x, y, z) et Vector3(x, y, z)
		Twist (geometry_msgs/Twist)	Non utilisé
		reference_frame (string)	« world »

Voici le schéma Simulink qui permet de faire l'envoi de la position ainsi que l'orientation au simulateur de Gazebo.

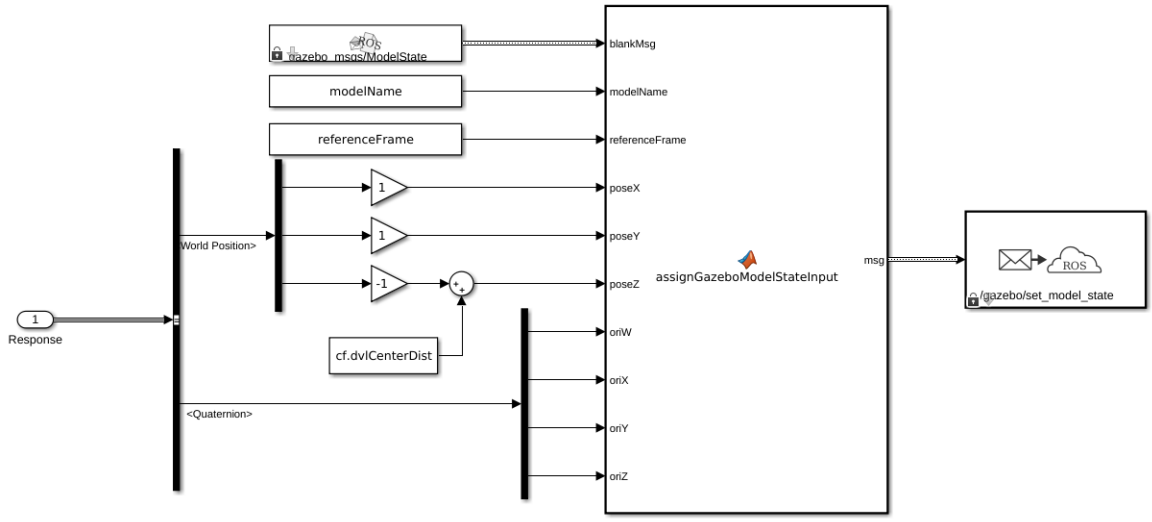


Figure 49: Schéma bloc pour la connexion à Gazebo

Encore une fois, la fonction Matlab que nous retrouvons sur le schéma ci-haut permet d'attribuer les données provenant du contrôleur à un message ROS pour être envoyé grâce au « publisher ».

10 Analyse et résultats

Dans cette section, nous allons présenter les résultats de simulations que nous avons effectuées à l'aide de Simulink pour finalement en faire l'analyse. Nous présentons alors deux simulations que nous avons effectuées.

Simulation 1

Trajectoire

Pour la première simulation, nous avons donné une trajectoire spécifique au sous-marin. Celle-ci représente bien ce que le sous-marin pourrait être amené à faire en compétition. On le fera descendre de 1 mètre, tourner de 90 degrés et avancer de 2 mètres vers l'avant. En compétition, il est interdit d'opérer le sous-marin à la surface de l'eau. C'est pourquoi nous devons toujours submerger complètement le sous-marin dans l'eau au départ. Voici les graphiques des positions linéaires et du quaternion de la trajectoire que nous avons générée.

10.1

10.1.1

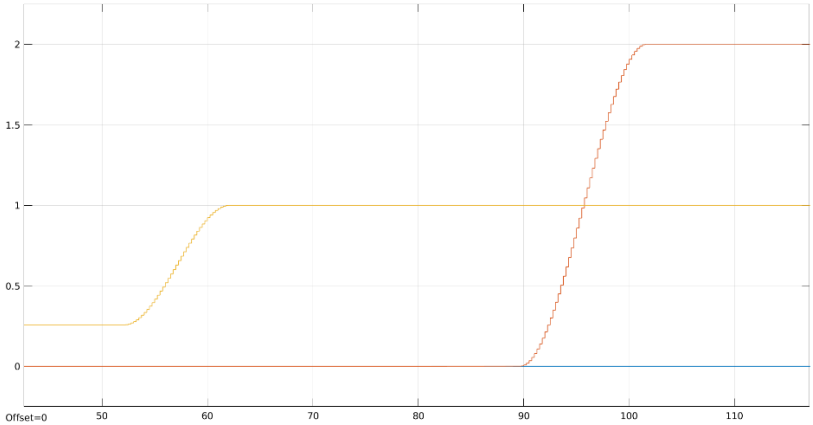


Figure 50 : Trajectoire générée – Positions linéaires (Simulation 1)

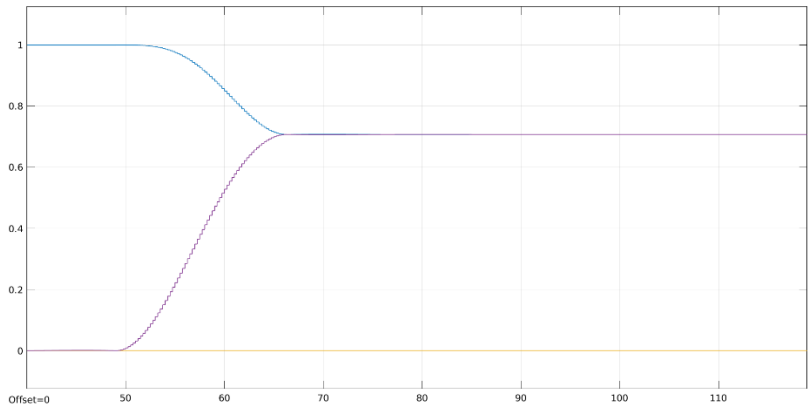


Figure 51: Trajectoire générée – Quaternion (Simulation 1)

Résultats

Cette section présente les résultats de la simulation effectués à l'aide Simulink. Nous affichons les graphes qui comparent les sorties (les états dans notre cas) à leurs références respectives pour ensuite faire une analyse récapitulative. Voici les résultats.

10.1.2

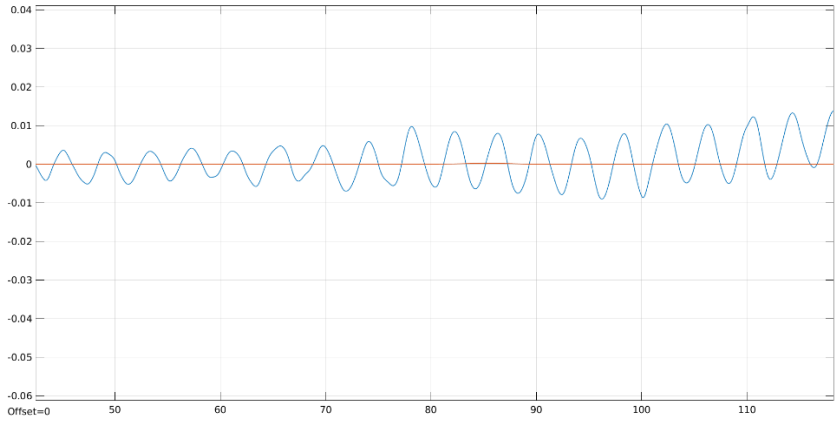


Figure 52 : Réponse – Axe X (Simulation 1)

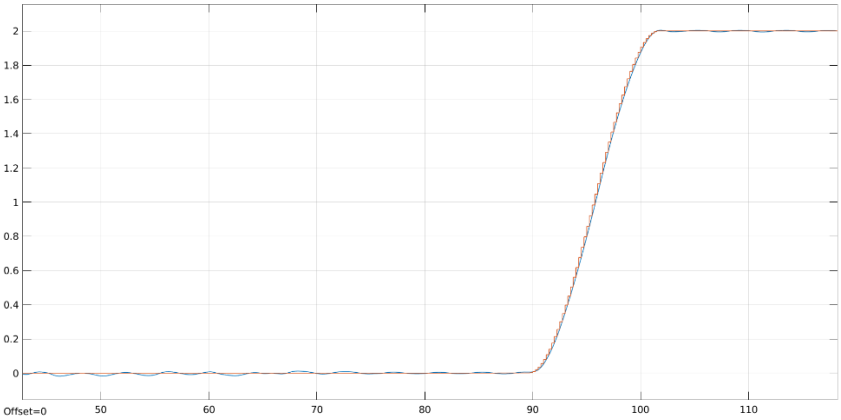


Figure 53 : Réponse – Axe Y (Simulation 1)

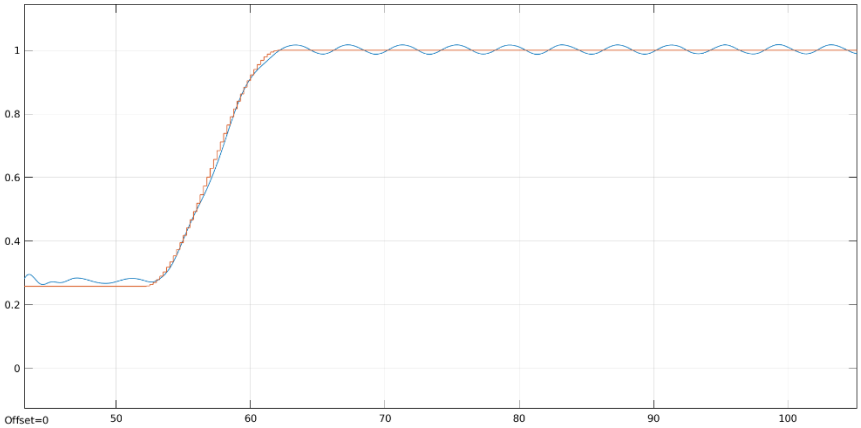


Figure 54 : Réponse axe Z (Simulation 1)

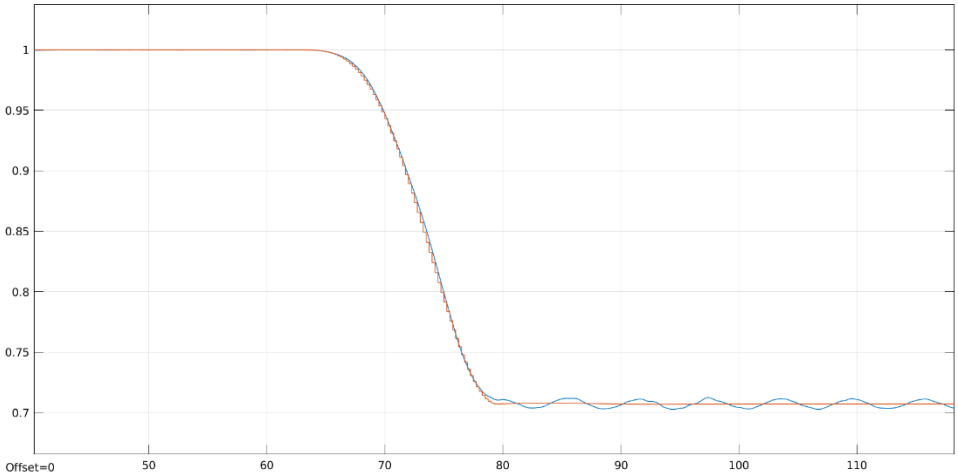


Figure 55 : Réponse η (Simulation 1)

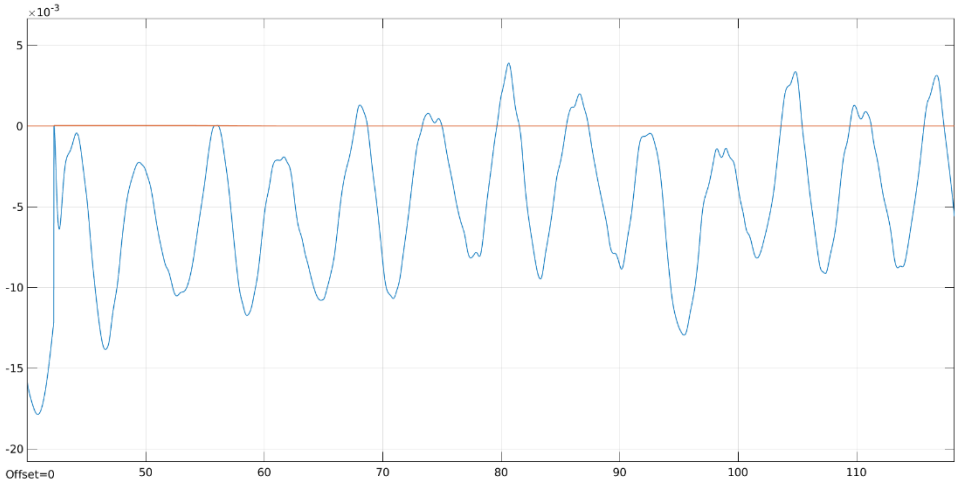


Figure 56 : Réponse ε_1 (Simulation 1)

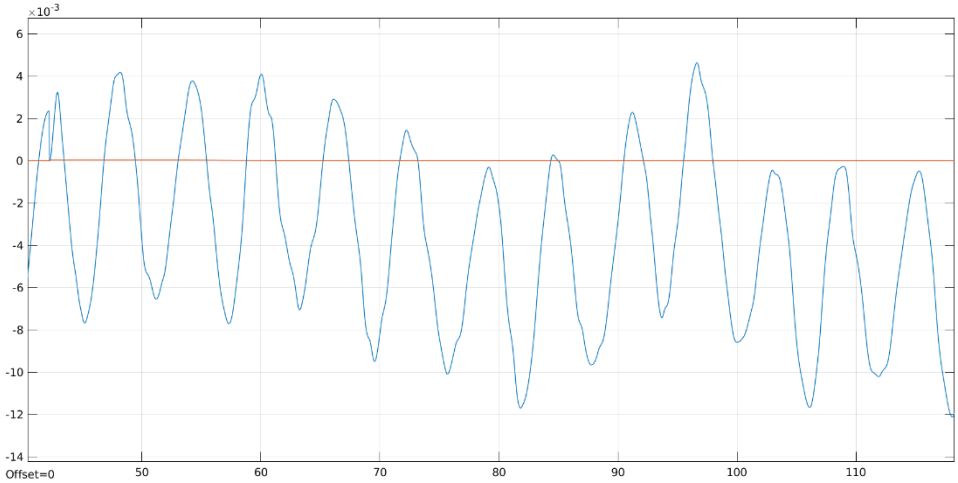


Figure 57: Réponse ε_2 (Simulation 1)

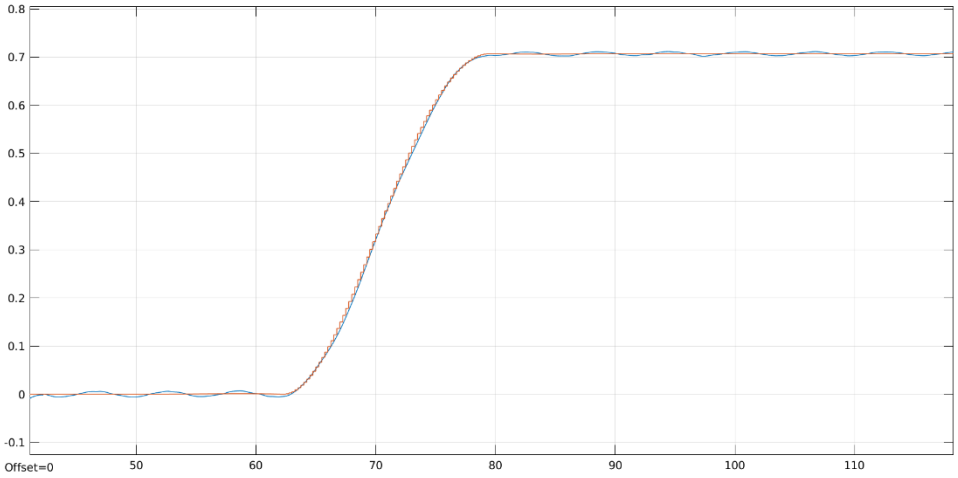


Figure 58 : Réponse ε_3 (Simulation 1)

Finalement, voici la commande des moteurs provenant du MPC.

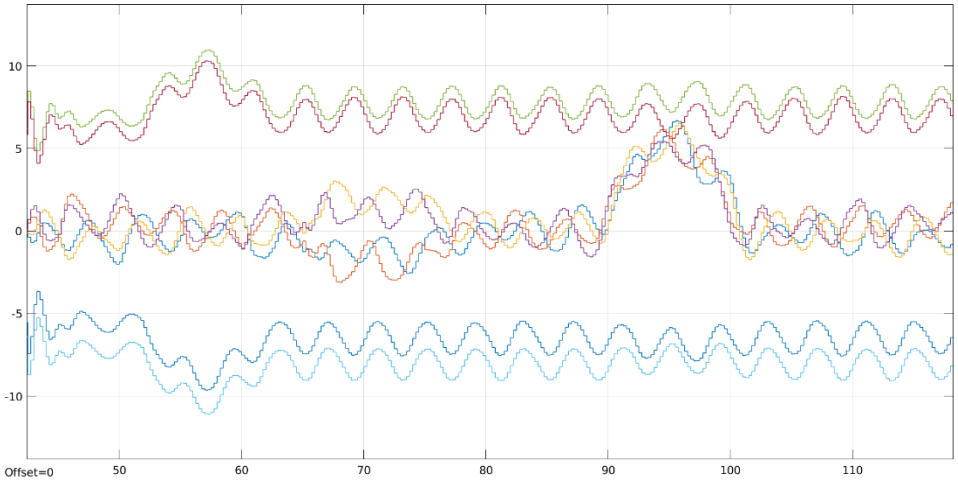


Figure 59 : Commande du MPC (Simulation 1)

Simulation 2 (Préqualification)

Cette simulation est une des plus importantes que nous avons faites puisqu'elle représente la préqualification de la compétition. Toutes les équipes qui désirent participer à la compétition doivent absolument réussi cette mission pour pouvoir participer à la compétition.

10.2 *Trajectoire*

Afin de créer la trajectoire utilisée pour cette simulation, nous avons procédé en plusieurs étapes. Tout d'abord, nous avons tracé cette trajectoire dans l'outil de modélisation SolidWorks. Voici à quoi 10.2.1 ressemble la trajectoire modélisée. Comme vous pouvez le voir sur la figure suivante, nous avons choisi de faire monter le sous-marin tout en tournant. Nous voulions générer une trajectoire ou le sous-marin se déplace sur 4 degrés de liberté simultanément.

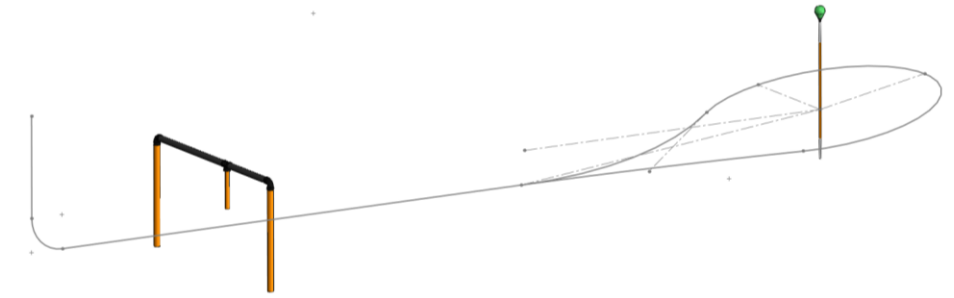


Figure 60 : Trajectoire modélisée dans SolidWorks (Simulation 2)

Par la suite, nous avons choisi quelques points sur la trajectoire afin de les donner à notre générateur de trajectoire à l'aide d'un script Python. Voici la trajectoire qui a été générée par notre générateur de trajectoire. Sur la figure, nous pouvons observer les points générés ainsi que les orientations pour chacun d'eux dans une perspective 3D. Ce sont ces points et orientations générés qui seront envoyés en référence au MPC.

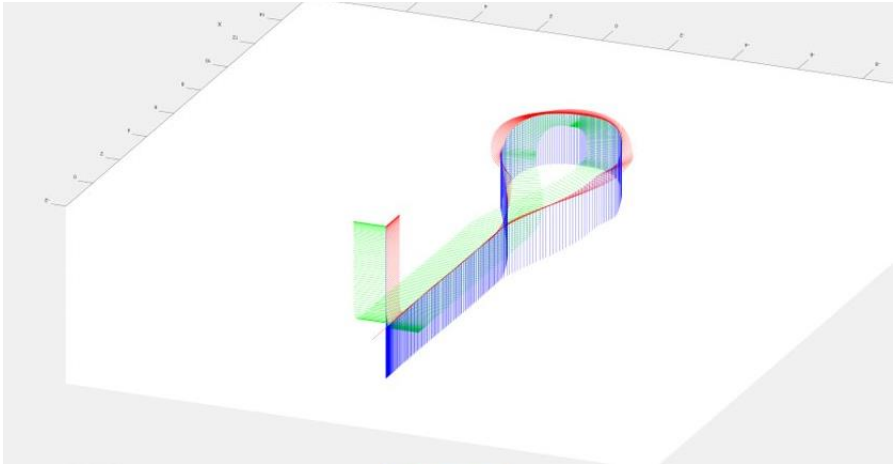


Figure 61 : Trajectoire générée - Matlab (Simulation 2)

Voici les graphiques des positions linéaires et du quaternion de la trajectoire que nous avons générée.

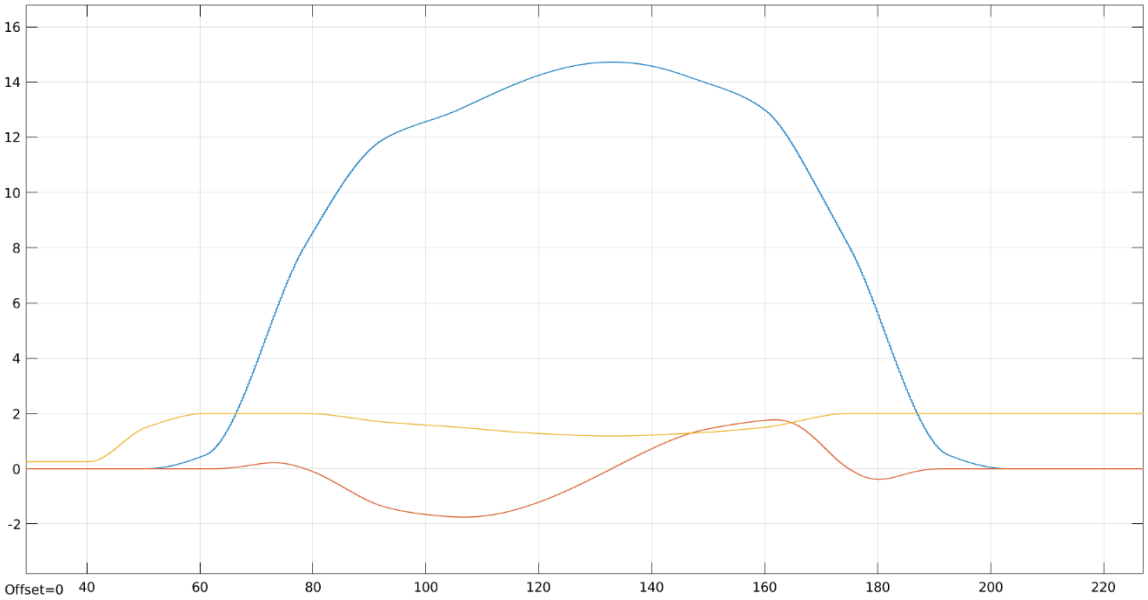


Figure 62 : Trajectoire générée – Positions linéaires (Simulation 2)

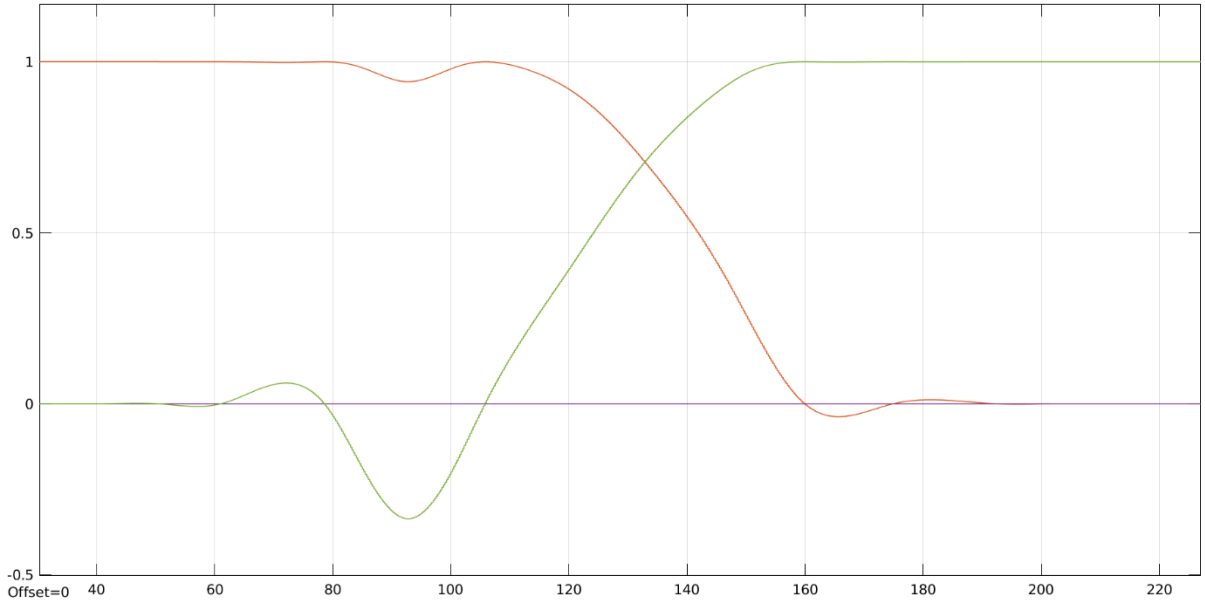


Figure 63 : Trajectoire générée – Quaternion (Simulation 2)

Résultats

Cette section présente les résultats de la simulation effectués à l'aide Simulink. Nous affichons les graphes qui comparent les sorties (les états dans notre cas) à leurs références respectives. Ensuite, nous allons faire une analyse récapitulative des résultats.

10.2.2

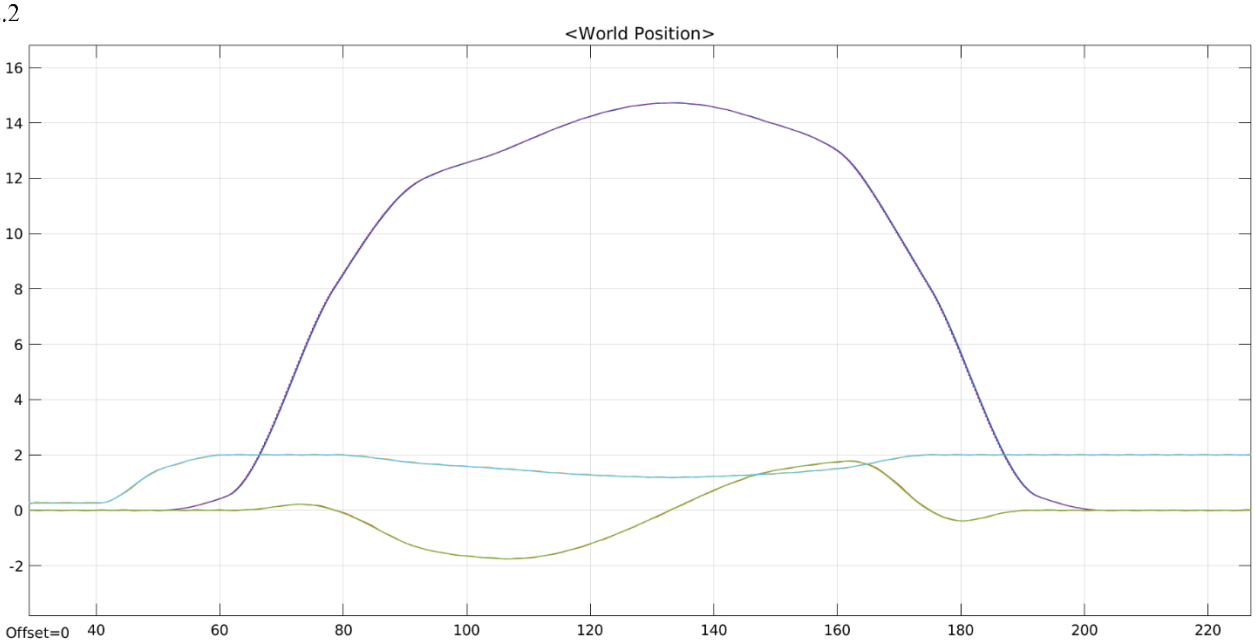


Figure 64 : Réponse – Position linéaire (Simulation 2)

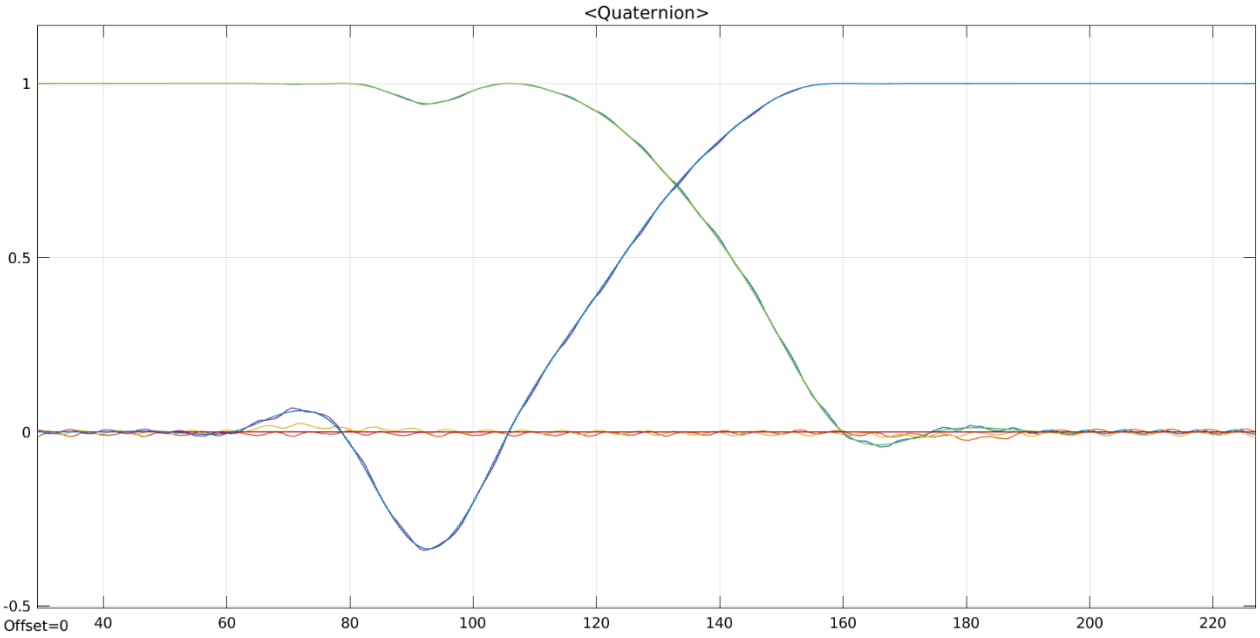


Figure 65 : Réponse – Quaternion (Simulation 2)

Finalement, voici la commande des moteurs provenant du MPC.

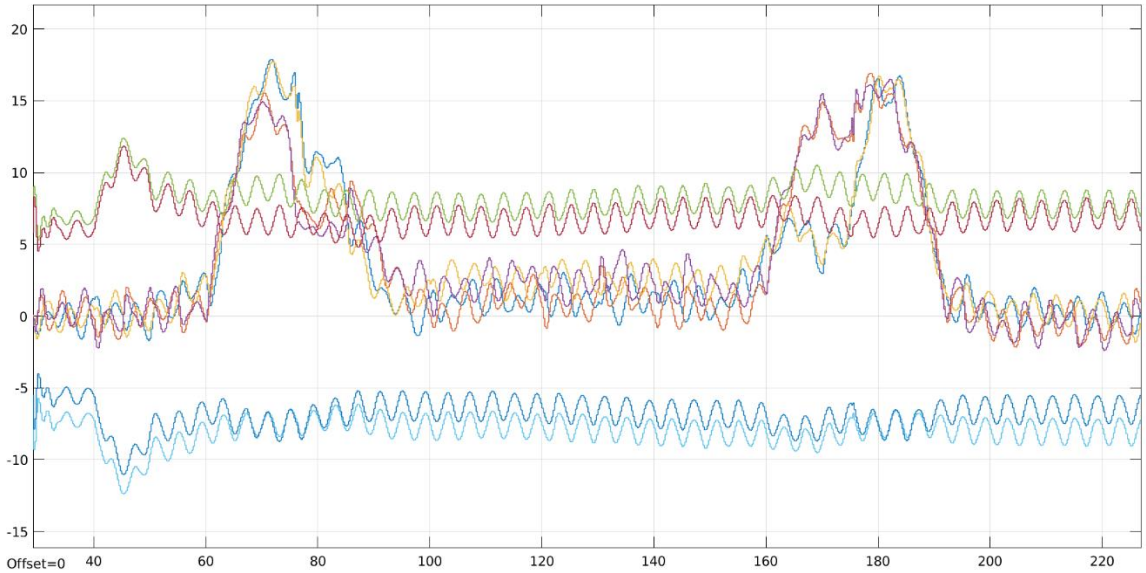


Figure 66 : Commande du MPC (Simulation 2)

10.3 Analyse

Nous pouvons remarquer que le contrôleur est en mesure de suivre la trajectoire qu'on lui a imposée en tenant compte des contraintes que nous avons imposées. On remarque également que nous avons une erreur nulle en régime permanent et que si un dépassement est observé, il est négligeable. On peut également remarquer que les réponses pour les positions linéaires sont meilleures que celles de l'orientation. Il ne s'agit pas là d'un hasard, mais plutôt d'un résultat souhaité que nous avions prévu en mettant les poids pour l'orientation plus faibles que ceux de la position. Nous voulions avoir une meilleure position au détriment de l'orientation et c'est ce que nous avons obtenu en simulation. Pour la commande, nous pouvons également observer que cette dernière respecte les contraintes de forces que nous avions données au MPC. Nous pouvons remarquer que la commande oscille afin de contrer les perturbations externes. De plus, les gains MVR ont bien été ajusté, car nous pouvons voir que la commande n'est pas agressive. Le moteur devrait être en mesure de suivre la commande du contrôleur. Finalement, les points que nous avons envoyés pour créer la trajectoire ont permis d'obtenir une trajectoire telle que la voulions. La trajectoire générée n'est pas complètement lisse et comporte des imperfections, mais elle sera améliorée dans le futur.

11 Conclusion

Pour conclure, nous avons fait la conception d'un asservissement pour un sous-marin autonome dans le cadre d'un projet spécial pour le club technologique et scientifique S.O.N.I.A. Pour réaliser ce projet, nous sommes passés par les étapes suivantes : l'identification du problème, la recherche de solutions, l'analyse de ces solutions, l'élaboration d'un plan de travail pour mener à terme les objectifs établis, le développement de la solution, la documentation et la présentation du projet. Tout au long du projet, nous avons travaillé en étroite collaboration avec le reste de l'équipe de S.O.N.I.A pour développer un contrôle innovant qui se lie directement avec la philosophie du club. Nous avons fait la documentation du projet ainsi que des nombreuses réflexions que nous avons eues. Ces réflexions nous ont alors permis d'effectuer des choix pour faire avancer ce projet. Nous sommes très heureux de pouvoir laisser aux futurs membres les connaissances que nous avons acquises durant ce projet.

Bien que nous ayons eu l'occasion d'effectuer des simulations avec Simulink ainsi que dans nos simulateurs, nous aurions aimé pouvoir effectuer des tests en piscine. Dès que la situation sanitaire nous le permettra, nous allons pouvoir effectuer des tests plus réalistes en piscine afin de voir notre travail en action dans l'eau et effectuer le travail nécessaire pour que notre contrôle puisse être déployé sur le sous-marin.

Dans les prochaines itérations du projet, nous allons continuer de faire des améliorations sur notre contrôle. Nous pensons déjà à quelques améliorations logicielles que nous pourrions apporter pour augmenter la modularité du logiciel afin que ce dernier soit portable sur l'un ou l'autre des sous-marins. Nous aimerions aussi concevoir de nouveaux modes de contrôle et peut-être même implémenter d'autre algorithme de contrôle. Un autre élément que nous aimerions serait de faire l'estimation des constantes du modèle qui nous sont encore manquantes et qui n'étaient pas dans le cadre de ce projet. Enfin, nous désirons poursuivre notre travail afin d'améliorer encore et encore le contrôle du sous-marin.

Notre développement va certainement permettre à S.O.N.I.A de continuer d'innover, de développer et implémenter de nouvelles technologies sur leurs prototypes. Le club pourra se baser sur un contrôle fiable qui permettra au sous-marin d'accomplir des tâches complexes et ce, de manière complètement autonome. Le partage de notre travail va peut-être même pouvoir aider des équipes de la compétition ou des gens passionnés par les AUV et c'est exactement ce que nous souhaitons en tant que membres d'un club technologique et scientifique.

BIBLIOGRAPHIE

Images

RoboSub, (2019). « **22nd Annual International RoboSub Competition - Mission and Scoring V1.0** ». Repéré à <https://robonationforum.vbulletin.net/filedata/fetch?id=2196>

Tesei, A., Fioravanti, S., Gandi, V., Guerrini, P., Maguer, A. (2010). « **Localization of small surface vessels through acoustic data fusion of two tetrahedral arrays of hydrophones** » Repéré à <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.879.5677&rep=rep1&type=pdf>

BlueRobotics, (2019). « **T200 Thruster** ». Repéré à <https://bluerobotics.com/store/thrusters/t100-t200-thrusters/t200-thruster-r2-rp/>

ArduSub. « **Building a Vehicle Frame** ». Repéré à <https://www.ardusub.com/quick-start/vehicle-frame.html>

De A. Fernandes, D., J. Sørensen A., C. Donha, D., (2015). « **Path Generation for High-Performance Motion of ROVs Based on a Reference Model** » Repéré à <https://www.researchgate.net/publication/277013046_Path_Generation_for_High-Performance_Motion_of_ROVs_Based_on_a_Reference_Model>

Scheidler, P., (2018). « **Quaternions for Orientation** ». Repéré à < https://blog.endaq.com/quaternions-for-orientation >

Liu, F., Shen, Y., He, B., Wan, J., Wang, D., Yin, Q., Qin, P., (2019). « **3DOF Adaptive Line-Of-Sight Based Proportional Guidance Law for Path Following of AUV in the Presence of Ocean Currents** ». Repéré à < https://www.mdpi.com/2076-3417/9/17/3518/htm#B31-applsci-09-03518 >

G.U.N.T. « **TM 605 Coriolis Force** ». Repéré à < https://www.gunt.de/en/products/engineering-mechanics-and-engineering-design/dynamics/kinetics-dynamics-of-rotation/coriolis-force/040.60500/tm605/glct-1:pa-148:ca-18:pr-1409>

Wikipédia, (Dernière modification en 2021). « **Model Predictive Control** ». Repéré à <https://en.wikipedia.org/wiki/Model_predictive_control>

Castro, S., (2017). « **Getting Started with Matlab, Simulink, and ROS** ». Repéré à <https://blogs.mathworks.com/racing-lounge/2017/11/08/matlab-simulink-ros/>

3DConnexion, « **SpaceMouse Compact** ». Repéré à <https://3dconnexion.com/uk/product/spacemouse-compact/>

Documentation

Fossen, Thor I., (1994). « **Guidance and Control of Ocean Vehicles** ». Chichester: John Wiley & Sons Ltd., 494 p.

Fossen, Thor I., (2005). « **A Nonlinear Unified State-Space Model for Ship Maneuvering and Control in a Seaway** ». Norwegian University of Science and Technology: Department of Engineering Cybernetics, 28 p

Fernando Sá Reibeiro de Faria, P., (2016). « **Quaternion-based Dynamic Control of a 6-DOF Stewart Platform for Periodic Disturbance Rejection** ». Pontifical Catholic University of Rio Grande do Sul: Engineering Faculty, 96 p.

Islam, M., Okasha, M., Sulaeman, E., (2019). « **A Model Predictive Control (MPC) Approach on Unit Quaternion Orientation Based Quadrotor for Trajectory Tracking** ». International Journal of Control, Automation and Systems: KIEE and Springer, 14 p.

Vervoort, J.H.A.M., (2018). « **Modeling and Control of an Unmanned Underwater Vehicle** ». University of Canterbury: Department of Mechanical Engineering, Mechatronics, 119 p.

Fjellstad, O., Fossen, Thor I., (1994). « **Position and Attitude Tracking of AUVs: A Quaternion Feedback Approach** ». Norwegian University of Science and Technology: Department of Engineering Cybernetics, 15 p

Lavoie, O., Chennouf, R., (2018). « **Contrôle pour Sous-Marin Autonome** ». Montréal : École de technologie Supérieure, Université du Québec pour le cours SYS802, 53 p.

Aubin-Perron, F., Lemay, L., (2019). « **Étude Inertielle et Hydrodynamique d'un Sous-Marin Autonome pour l'intégration d'un Modèle de Contrôle** ». Montréal : École de technologie Supérieure, Université du Québec pour le cours MEC791 (Projet Spécial), 53 p.