Le génie pour l'industrie

RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COURS LOG791 PROJET SPÉCIAUX

TÉLÉMÉTRIE S.O.N.I.A.
INTERFACE UTILISATEUR DE CONTRÔLE DU SOUS-MARIN S.O.N.I.A.

KEVIN COOMBS
COOK25039304

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur-superviseur

Alain April

MONTRÉAL, 13 AVRIL 2017
HIVER 2017

TÉLÉMÉTRIE S.O.N.I.A.
INTERFACE UTILISATEUR POUR LE CONTRÔLE DU SOUS-MARIN

KEVIN COOMBS
COOK25039304

RÉSUMÉ

Ce rapport décrit l'entièreté du travail effectué dans le cadre du cours LOG791-PROJET SPÉCIAUX. Le projet est de développer une interface utilisateur basée sur RQT, une application fourni par la suite d'application et de librairie de ROS (Robot Operating System). Cette télémétrie prend place dans le contexte où le club S.O.N.I.A de l'École de Technologie Supérieur à fait de gros changement à sa plateforme logicielle, passant de Java à C++ et Python. En effectuant ce changement, le club a perdu plusieurs outils, dont une télémétrie permettant d'afficher le statut des différents périphériques du sous-marin en plus de permettre le contrôler. Suite à ce projet, le club S.O.N.I.A. est maintenant outiller pour partir en compétition au mois de Juillet 2017.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

DVL	Doppler Velocity Logger
ETS	École de Technologie Supérieur
IMU	Inertial Mesurement Unit
LOC	Line of code
ROS	Robot operating system
SONIA	Système d'Opérations Nautique Intelligent et Autonome

INTRODUCTION

Ce rapport se présente dans le cadre du cours LOG791, Projet spéciaux. Il présente le travail effectué lors du développement de la télémétrie pour le sous-marin du club S.O.N.I.A. Le travail sera présenté tout d'abord par une mise en contexte par rapport au club étudiant. Par la suite, il décrit l'analyse des besoins du club qui a été effectué.

Afin de bien présenter les résultats obtenus, une brève description de l'outil utilisée est présenté afin d'aider le lecteur à mieux comprendre le travail qui a été effectué. Ensuite, les différents modules développés au cours du projet sont démontré suivi d'un chapitre sur les différents tests réalisés pour prouver l'efficacité de la solution développé. Finalement, une section sur la méthodologie utilisée ainsi que les apprentissages acquis et les difficultés rencontrées sont présentés afin de mener à une conclusion du projet et quelques recommandations.

CHAPITRE 1

MISE EN CONTEXTE

1.1 Le club S.O.N.I.A

Fondée en 1999 à l'ETS, le club S.O.N.I.A a pour objectif de participer chaque année à la compétition Robosub à San Diego en Californie aux États-Unis. Cette compétition consiste à réaliser un parcours à obstacles à l'aide d'un sous-marin autonome. Le sous-marin étant autonome, il n'est aucunement contrôlé par un humain durant la compétition, réalisant le parcours à l'aide d'algorithmes et d'intelligence artificielle qui sont développé tout au long de l'année par les étudiants de l'école. Les obstacles de la compétition ont pour but de reproduire des tâches réelles d'applications des sous-marins autonomes dans les océans.

1.2 Virage 2017

Le sous-marin actuel utilisé en compétition a été fabriqué en 2011 par une équipe extraordinaire. Depuis 2011, les membres du club améliore et maintienne le sous-marin afin de l'amélioré à chaque année. Toutefois, les autres équipes de la compétition ont fait de nouveau sous-marin depuis 2011 et le sous-marin du club SONIA est devenu désuet. Il était donc temps pour l'équipe de se lancer le défi de fabriquer un nouveau sous-marin avec les nouvelles technologies disponibles. Pour ce faire, le club a fait la conception d'un sous-marin à moitié en aluminium et à moitié en fibre de carbone. De plus, la plateforme logicielle a été revue en entier. Anciennement en Java, avec des modules tous propriétaires, les membres de l'équipe ont choisi de refaire la plateforme à l'aide de ROS, une suite d'application et de librairie Open-Source et qui est maintenu par une communauté, spécifiquement pour les robots. Afin d'utiliser ROS, il a fallu abandonner le Java et refaire le code en C++ et en python, les deux seuls langages supporté par la communauté. En faisant ce changement, le club S.O.N.I.A. a perdu plusieurs années de travail à développer des outils en Java. C'est donc dans ce contexte que s'inscrit ce projet.

1.3 Problématique du club

Avec la perte de nombreux outils, le club S.O.N.I.A se retrouve à être beaucoup moins performant lors des tests en piscines. A titre d'exemples, les derniers tests piscines ont eu lieu au mois de novembre et le sous-marin était contrôler par ligne de commande dans un terminal sur Ubuntu. Il est devenu évident qu'il y avait un problème d'outils graphique que le club avait perdu lors de la migration vers ROS. La liste des outils perdus a été établi et les principaux outils perdu sont la télémétrie et l'éditeur de mission qui sont essentiel au bon déroulement des tests qui eux permettent d'être efficace lors de l'arrivé de l'équipe en compétition.

1.4 Objectif du projet

L'objectif du projet est de développer une télémétrie contenant, au minimum, les fonctionnalités perdu lors de la migration vers ROS par rapport à l'ancienne télémétrie.
Cette télémétrie doit permettre d'afficher le statut des périphériques du sous-marin en plus de permettre le contrôle du sous-marin.

CHAPITRE 2

ANALYSE DES BESOINS

2.1 Rencontre avec les membres logiciels du club S.O.N.I.A

Afin d'être en mesure de définir les besoins du club étudiant, un brainstorming a été tenu afin de faire sortir les différentes idées pour la nouvelle télémétrie. Dans cette rencontre, les points principaux qui sont ressorti sont les suivants :

1	Il doit être facile d'ajouter des nouveaux modules
2	La télémétrie doit contenir l'éditeur de mission
3	Visionnement 3D de la position du sous-marin
4	Affichage intuitif des données critiques
5	Interface modulaire
6	Permettre d'avoir plusieurs utilisateurs en même temps
7	Avoir un client de vision dans la télémétrie
8	Avoir les modules critiques de l'ancienne télémétrie

Table 2.1.1 Résultats brainstorming

Suite à cette rencontre, il a été simple de constater que certains membres voulaient quelques modules innovants, tel que le client de vision et l'affichage en 3D. Tandis que d'autre trouvait que l'ancienne télémétrie faisait un bon travail et été fiable et donc qu'il serait content d'avoir un logiciel le plus similaire possible. Une analyse des anciens outils a donc été effectuée.

2.2 Analyse de l'ancienne télémétrie

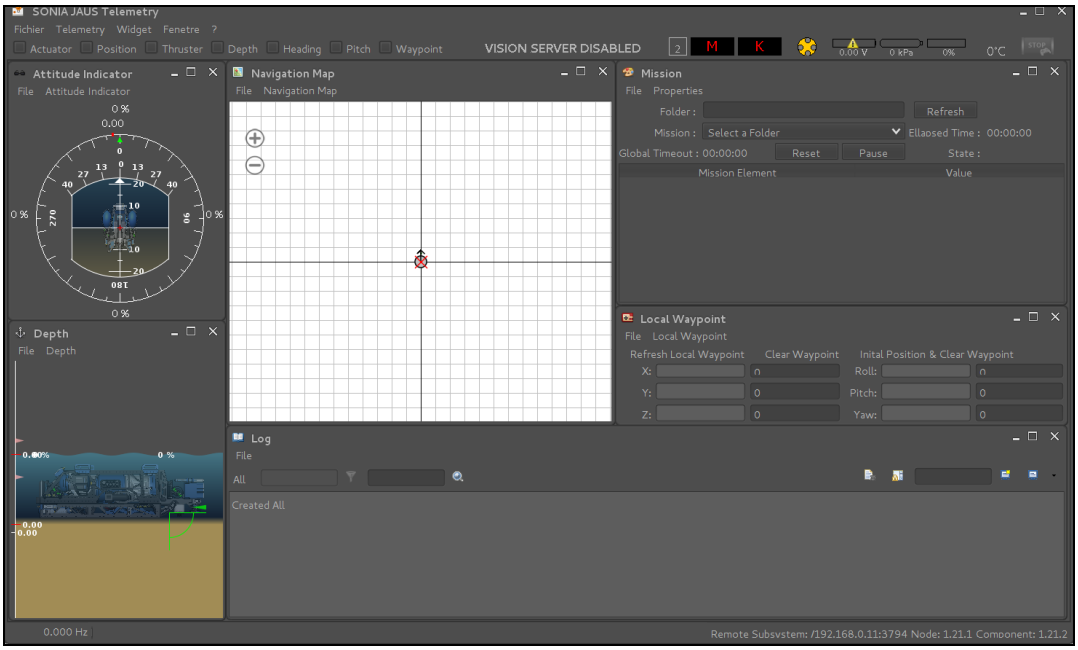

Figure 2.2.1 Ancienne télémétrie

L'analyse des modules présents dans l'ancienne télémétrie à mener à la production de cette liste :

Liste des logs	Indicateur de profondeur
Compass	Indications de forces motrices effectuées
Carte de navigation 2D	Position brute du sous-marin sur les 6 axes
Action sur la lumières	Gestion des LEDs
Activation / Désactivation des périphériques	Choix des missions
Lancement d'une mission	Gestion de la piscine
Sélectionner un point sur la carte comme destination	Ajustement des PIDs
Gestion des torpilles	Gestion du Sonar

Table 2.2.1 Listes des anciennes fonctionnalités de la télémétrie

Cette liste a été prise en compte lors de l'élaboration des spécifications

2.3 Analyse de l'ancien éditeur de mission

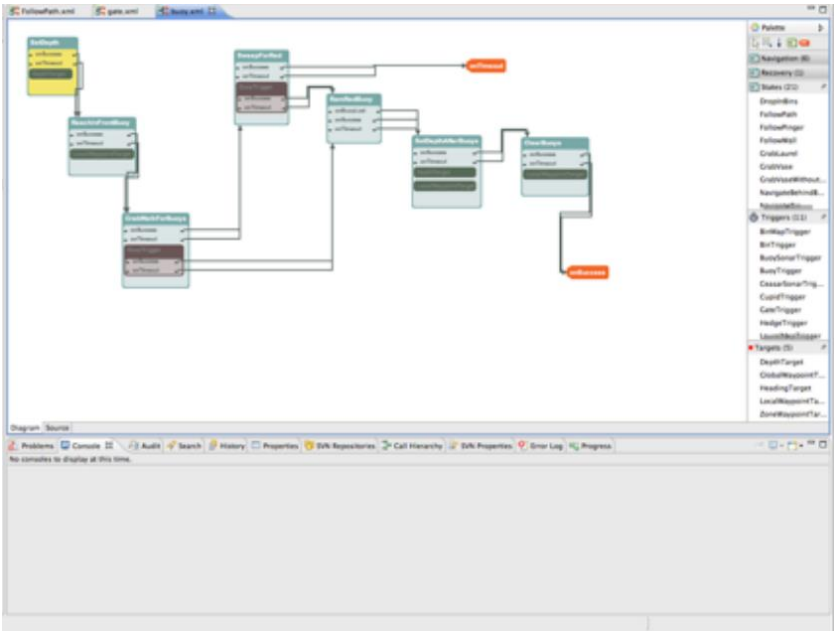

Figure 2.3.1 Ancien Éditeur de mission

Il faut savoir le que l'éditeur de mission permet de configurer la machine à état utiliser dans le sous-marin pour savoir les différentes étapes de la compétition. L'analyse de l'éditeur de mission a permis d'établir cette liste de fonctionnalités :

Ajout d'état machine	Ajout de lien entre les états
Catégorisation des états	Paramétrage des états
Ajout dynamique des états à la liste basée sur les fichiers de codes	Ajout de sous-missions dans une mission
Sauvegarde d'une mission	Chargement d'une mission
Affichage de problème de validité dans la mission (par exemple : pas d'état de début)	Déplacement des états dans un environnement libre (Style Paint)

Table 2.3.1 Listes des anciennes fonctionnalités de l'éditeur de mission

Différentes fonctionnalités n'ont présente ont été énumérer par l'équipe comme étant des « Nice to Have » :

- Édition du code à partir de l'éditeur de mission;

- Téléversement de la mission directement sur le sous-marin;
- Ouvrir une sous-mission dans un onglet.

2.4 Spécifications

Suite à cette analyse, les modules suivants ont été déterminants comme étant les prioritaires pour être efficace le plus rapidement possible pour les tests en piscine.

1. Console
2. Indicateur de profondeur
3. Carte de navigation 2D/3D
4. Affichage de la position brute sur les 6 axes
5. Affichages des efforts des moteurs
6. Affichage d'un flux de vision
7. Une barre d'outils affichant les informations de batteries et permettant de désactiver des axes de contrôles.
8. Éditeur de mission.
9. Affichage des données de puissances
10. Affichage des données de sonar
11. Activation des torpilles
12. Activation des marqueurs
13. Activation de la pince mécanique

Afin d'être le plus efficace possible, le développement se fait en mode agile sur des sprints de 4 semaines. Le premier sprint inclura les points 1 à 7. Le sprint 2 sera de faire l'éditeur de mission et finalement, le sprint 3 sera le restant des points sachant qu'il s'agit de modules secondaires qui pourront être développé par la suite en cas de manques de temps.

2.5 Identification des risques

Une identification des risques a été effectué. Voici un tableau présentant les résultats de l'analyse de risques.

Légende :
Impact : 1(très faible) à 5 (très important)
Probabilité : 1 (très faible) à 5 (très forte)

Risque	Impact	Probabilité	Mitigation / atténuation
Problème/Bug du côté du système de contrôle.	Ralentissement de la connexion de l'interface utilisateur avec le système de contrôle. (2)	Puisque le code est nouveau, cela risque d'arriver assez fréquemment (4)	Se tenir informé du statut d'un module du système de contrôle avant de faire un module de la télémétrie qui sera connecté sur celui-ci. [Garder une bonne communication avec l'équipe]
Manque d'expérience avec ROS et RQT	Ralentissement au début de l'implémentation (4)	(4)	Commencer à lire immédiatement sur ROS et RQT.
Manque de disponibilité de l'équipe SONIA pour donner une rétroaction de l'avancement	Avoir à refaire certaine section après avoir commencé le développement d'un module.(3)	L'équipe est souvent disponible (2)	Envoyé les mock-up par courriel au lieu d'attendre de croisé l'équipe au local.
Manque de temps	Ralentissement totale de l'avancement du projet(5)	Facilité à mettre 9h par semaine sur le projet, peut-être plus dur d'en mettre plus. (2)	Bien planifier l'agenda pour réussir à bien organiser son temps.

Manque de connaissance en C++ et/ou Python	Le développement sera plus lent à démarrer. (3)	(4)	Demander des conseils aux membres de l'équipe SONIA et faire plus de recherche.

CHAPITRE 3

PLATEFORME UTILISÉE

3.1 ROS – Robot Operating System

La suite d'outils fournis par la communauté de ROS est très complète. Malgré un petit manque de documentation sur les différents outils fournis dans la suite, il est facile de se rendre compte que les outils sont très utiles lors du développement d'un robot. La base de ROS est certainement son protocole de communication. Dans ROS, chaque module est considéré comme un nœud, ayant des communications entrantes et sortantes. La suite d'application permet de rendre très abstrait la communication et de pouvoir se concentrer sur le contenu des messages. Si un nœud veut communiquer à tous, alors il diffuse un message sur ce qu'il appelle un Topic. De même, il peut écouter sur n'importe quel Topic. Si le nœud cherche à avoir une communication plus synchrone avec un autre nœud, alors il utilise plutôt un Service qui permet d'envoyer un message et d'obtenir une communication en retour. Sous cette belle structure de message, ROS s'occupe de gérer les adresses IP entre les nœuds, de gérer le TCP/IP ou la retransmission de message en cas de message non rendu.

À partir de ce protocole de communication, qui est un standard dans ROS, plusieurs outils de bases ont été développés par la communauté pour permettre d'écouter le trafic réseau et voir afficher selon différentes manières les données. De plus, ROS fourni plusieurs standards de messages qui sont supporter dans plusieurs outils. À titre d'exemple, si on a u point en 3d, alors ROS suggère d'utiliser un message de type PointCloud qui est ensuite compatible avec plusieurs outils tel que RViz, qui permet de visualiser en 3D un environnement.

C'est donc basé sur ces standards que RQT voit le jour.

3.2 RQT

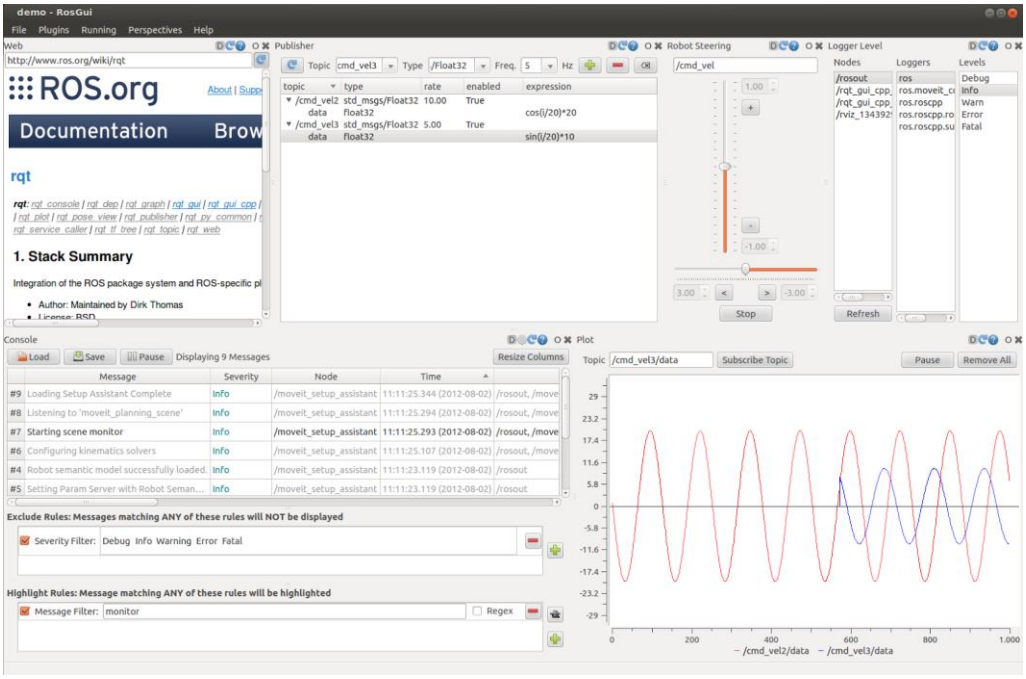

Figure 3.2.1 RQT

RQT est une application de ROS qui permet d'ouvrir des plugins développé avec QT. De base, l'application contient plusieurs plugins utiles tels qu'un fureteur web, différents plugins de modélisation de données et d'autres plugins qui permettent de faire des actions sur des messages standards dans ROS.

RQT permet donc de démarrer des plugins dans une interface modulaire et d'en sauvegarder des perspectives. Ce qui permet d'ouvrir plus d'un plugin à la fois, de placer ces plugins comme voulu dans l'écran et finalement de sauvegarder la perspective afin de la retrouver rapidement.

C'est ce logiciel qui est utilisée à la base pour faire la télémétrie du club S.O.N.I.A. Ainsi, le but du projet est de développer les différents plugins et ensuite simplement de faire une perspective qui sera facilement réutilisable. De plus, ceci offre la flexibilité que plusieurs utilisateurs du club pourraient avoir des perspectives différentes selon leurs rôles.

CHAPITRE 4

RÉSULTAT DU PROJET

Afin de démontrer les résultats du projet, voici le détail de chacun des modules développés avec l'objectif, les requis ainsi que les détails de réalisation de chacun. Le tout suivi d'une image du module.

4.1 Console

4.1.1 Objectif

L'objectif de ce panneau est de fournir l'affichage en temps réel de tous les messages de débuggage dans les différents modules du sous-marin.

4.1.2 Requis

- Afficher les logs en temps réel.

4.1.3 Réalisation

Avant de faire le développement de ce module, quelques recherches ont été effectués et le module rqt_console, fourni par la communauté de ROS, à été trouvé. Ce module rempli tous les requis que le club SONIA avait pour l'affichage des messages. Il permet de filtrer les messages, de faire des recherches, de sauvegarder les messages reçus, etc. Toutefois, il était un petit peu trop complexe. Le projet étant Open Source, une copie a été effectué, puis des modifications mineurs ont été effectuées afin d'ajouter un bouton « Hide Config » permettant de cacher certaines parties de l'interface initiale lors de l'appuie sur ce bouton.

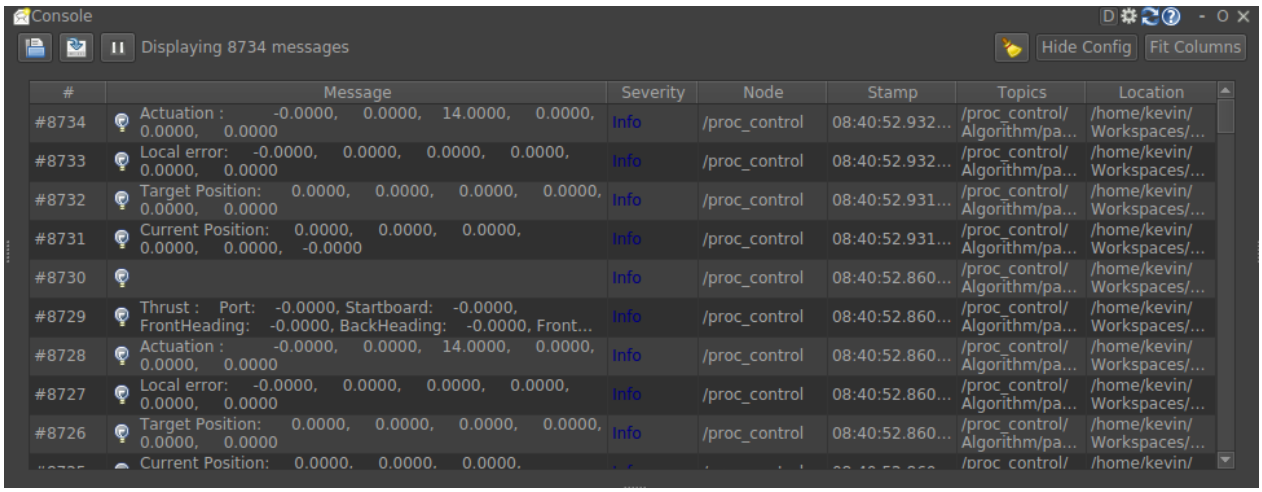

Figure 4.1.1 Module : Console

4.2 Indicateur de profondeur

4.2.1 Objectif

L'objectif de ce module est de fournir un indicatif clair et intuitif de la profondeur actuelle du sous-marin.

4.2.2 Requis

- Afficher la profondeur actuelle par une image, ou graphique ou autre.

4.2.3 Réalisation

Afin de réalisée un panneau simple et intuitif, un « Slider » vertical a été implémenter avec un dégradé du bleu pâle vers un bleu plus foncé comme couleur de fond. Ainsi, lorsque le sous-marin descend dans l'eau, le curseur descend vers la zone de plus en plus foncé. En liaison avec la réalité du club, le « Slider » descend linéairement entre 0m et 5m. Si le sous-marin descend sous les 5 mètres, le curseur reste accoté dans le bas et la valeur au descend continue d'indiquer la profondeur en temps réel. Toutefois, cette situation ne devrait jamais ce produire puisque physiquement, le sous-marin n'est pas fabriquer pour supporter des pressions plus grande que celle retrouver à 5m de profondeur.

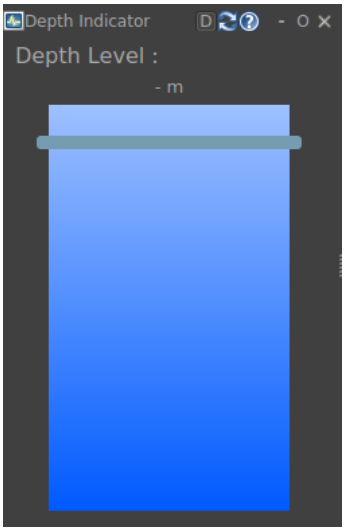

Figure 4.2.1 Module : Indicateur de

4.3 Navigation Map

4.3.1 Objectif

L'objectif de ce panneau est d'afficher sur une carte la position globale du sous-marin en temps réel.

4.3.2 Requis

- Afficher la position du sous-marin sur une carte;
- Permettre d'envoyer une destination au sous-marin par un clic droit;
- Permettre d'activer/désactiver les différentes couches de la carte (Sous-marin, grille, etc.);
- Permettre de redémarrer à zéro l'historique de navigation.
- Permettre de « lock » la position de la carte sur le centre du sous-marin et ainsi avoir la carte qui bouge en même temps que le sous-marin.(Style GPS)

4.3.3 Réalisation

Avant de démarrer le développement, des recherches ont été effectué dans les projets offerts par la communauté de ROS, et le projet rqt_pose_view, qui permet de représenter la position

d'un objet en 3D, semblait se rapprocher du besoin du club SONIA. Il a donc été utiliser comme base de projet. Toutefois, après avoir démarrer le développement, il est rapidement devenu évident que le résultat serait très différent du projet original.

Le résultat répond parfaitement au besoin club. Ce module permet d'afficher la position du sous-marin en 3D, ainsi qu'en 2D. Un clic droit sur la carte permet de définir une « target » au sous-marin, d'activer/désactivé chacun des « Layer » ainsi que de redémarrer à zéro l'historique de navigation.

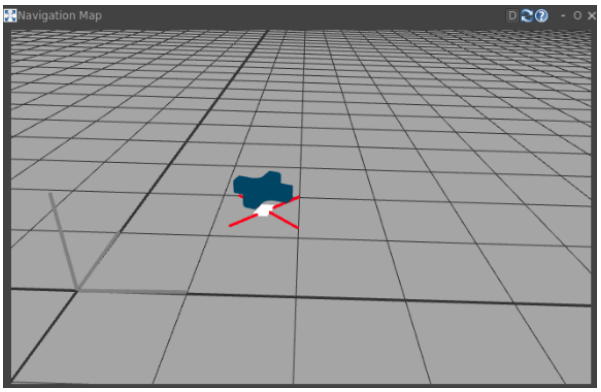

Figure 4.3.1 Module : Carte de navigation 3D

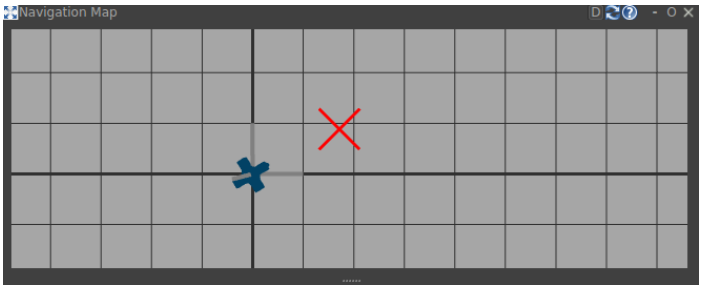

Figure 4.3.2 Module : Carte de navigation 2D

4.4 Local waypoint

4.4.1 Objectif

L'objectif de ce module est d'afficher les coordonnées du sous-marin, aux centimètres près, en temps réel, en plus d'afficher et de permettre la saisie d'une nouvelle destination du sous-marin.

4.4.2 Requis

- Afficher les coordonnées du sous-marin en temps réel;
- Permettre la saisie d'une destination
- Permettre de réinitialisée la position initial. (Menu → Waypoint)

4.4.3 Réalisation

Afin de réaliser ce module, un simple panneau à été développer avec des champs de saisies désactivée pour l'affichage de la position courante et des champs de saisies modifiable pour la destination. Sur l'appuie de la touche « Entrée », la destination est envoyé au sous-marin. De cette façon, il est possible de modifier plusieurs champs avant que la destination ne soit envoyé.

Figure 4.4.1 Module : Position brute

4.5 Thruster Effort

4.5.1 Objectif

L'objectif de ce module est d'affiché, en temps réel, les efforts effectuées par les moteurs, de manière instinctive et rapide, afin de rapidement détectée un comportement anormal.

4.5.2 Requis

-Afficher en temps réel les efforts de moteurs par une image, un graphique ou autre.

4.5.3 Réalisation

Afin de réaliser ce module, le projet de DepthIndicator à été utilisé comme base. Le dégradé de fond à été modifié, et le « Slider » à été dupliquer pour correspondre à nos différents moteurs.

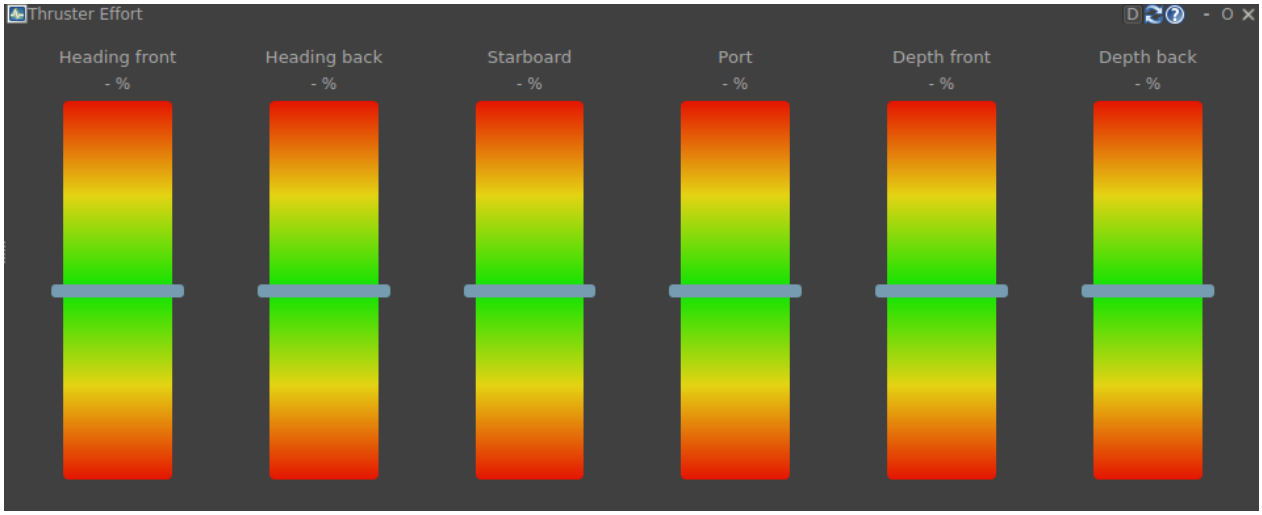

Figure 4.5.1 Module : Effort des moteurs

4.6 Thruster Control

4.6.1 Objectif

L'objectif de ce module est de permettre d'envoyer manuellement des commandes aux différents moteurs. Il doit également permettre d'effectuer un « Test à sec[1] »

4.6.2 Requis

- Permettre d'envoyer des commandes moteurs manuellement;
- Permettre de faire un test à sec(Menu → Test à sec)

1

Routine consistant à faire tourner tous les moteurs, un à la fois, afin de vérifier rapidement que tous les moteurs sont opérationnels.

4.6.3 Réalisation

Le panneau développer à été inspirer très fortement du panneau dans l'ancienne télémétrie détenu par le club S.O.N.I.A. Ainsi, il y a un « Slider » permettant de définir la valeur de la commande moteur à envoyer, et des boutons sont présents pour envoyer les commandes les plus utilisées soit 0%, ±50% et ±100%.

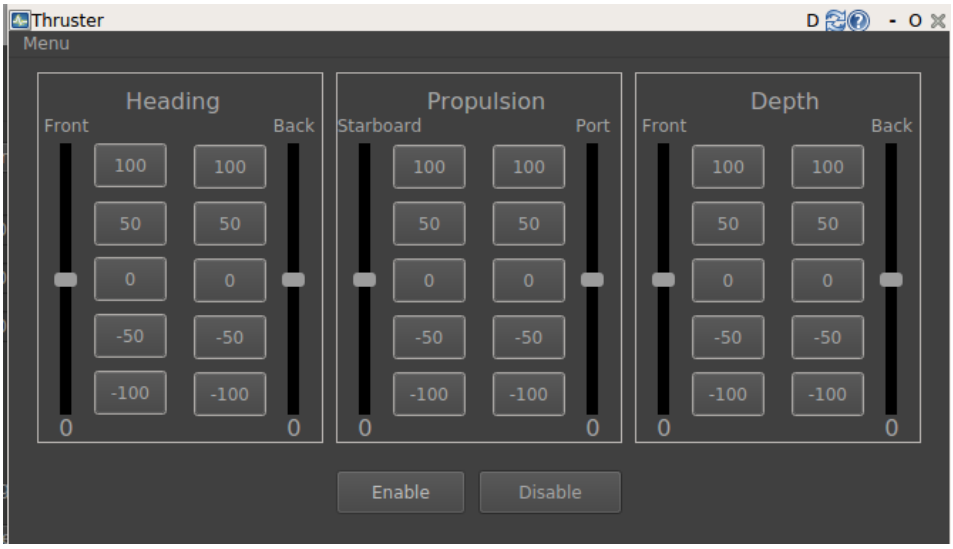

Figure 4.6.1 Module : Contrôle des moteurs

4.7 Vision Client

4.7.1 Objectif

L'objectif de ce module est d'intégrer une visualisation des flux vidéos présentement utilisées par le sous-marin à la télémétrie.

4.7.2 Requis

- Permettre d'afficher un flux vidéo;
- Permettre de choisir son flux vidéo;
- Permettre de rafraichir la liste des flux vidéos;
- Permettre de créer un nouveau flux vidéo, et de lui associé une chaîne de filtre;

- Permettre de créer une nouvelle chaîne de filtre;
- Permettre de configurer la chaine de filtre d'un flux vidéo.

4.7.3 Réalisation

L'idée originale de ce module était de prendre le projet existant au club S.O.N.I.A GUI_VISION_CLIENT, qui est une application « standalone » en C++, et de l'intégrer sur RQT dans la télémétrie. Toutefois, certains membres de l'équipe était retissant à vouloir l'intégration de ce logiciel parce qu'il était trop complexe et plein de « bug ». Le logiciel GUI_VISION_CLIENT a été développer par des nouveaux membres voulant se prouver un peu, donc l'architecture du logiciel est beaucoup trop complexe, contenant près de 40 fichiers en comptant les .h et .cpp. Après une analyse rapide des besoins, il a tout de même été décidé d'essayer de faire l'intégration sous prétexte qu'il était sûrement plus simple de faire l'intégration et ensuite de régler les « bugs » existants, que de refaire le tout. Après quelques heures d'essai, la décision a été prise de le refaire au complet parce que l'intégration n'était pas simple. Tous les « layouts » de l'application étaient à refaire, et les bugs étaient assez désagréables. Le client de vision a donc été refait en quelques heures, en python. Il remplit tous les requis que remplissait le dernier client, et fait un maximum de 1000 LOC, séparer sur 3 panneaux.

Le nouveau client est donc beaucoup plus facile à maintenir, plus léger en *processing* également, ce qui permet de lancer plusieurs instances en même temps.

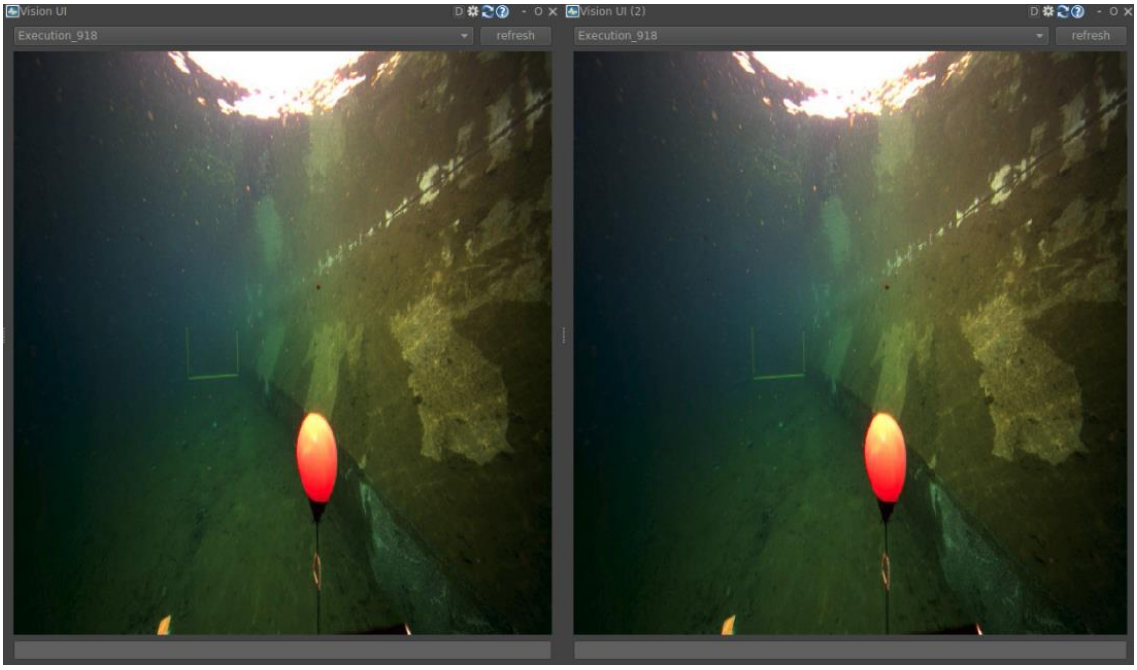

Figure 4.7.1 Module: Client de vision

4.8 Toolbar

4.8.1 Objectif

L'objectif de ce module est d'afficher le statut de la batterie, de désactiver les axes de contrôle du sous-marin ainsi que d'afficher le statut de deux périphériques surnommés Mission Switch et Kill Switch.

4.8.2 Requis

- Afficher le statut de la batterie;
- Afficher le statut de la Mission Switch;
- Afficher le statut de la Kill Switch;
- Permettre d'activer/désactiver les axes de contrôle.

4.8.3 Réalisation

La barre d'outils est séparé en trois parties. La première partie permet d'activer/désactiver les différents axes du sous-marin. Des « PushButtons » ont été utilisées afin d'avoir deux états.

Ensuite, un style à été défini pour être vert lorsque le bouton est appuyé, et rouge lorsqu'il n'est pas appuyé.

La deuxième partie est l'affichage du statut de la Mission Switch et Kill Switch. Lorsque ces périphiques sont activés, les lettres deviennent de couleurs. Rouge pour désactivé, Vert pour activer.

La dernière partie est l'affichage du statut de la batterie. Le dégradé est liée au niveau de la batterie. Le « Slider » est donc configurer pour bouger linérairement en 29V et 25V. La valeur précise de la batterie est également affiché en texte à droite du « Slider »

Figure 4.8.1 Module : Toolbar à gauche

Figure 4.8.2 Module : Toolbar à droite

4.9 Éditeur de mission

4.9.1 Objectif

L'objectif de ce module est de générer un fichier .yaml qui sera chargé par l'exécuteur de mission sur le sous-marin.

4.9.2 Requis

- Lister les états disponibles depuis les fichiers de codes du projet de l'exécuteur de mission;
- Permettre d'ajouter des états à la mission;
- Permettre de faire des liens entre les états;
- Permettre de définir un état initial;
- Permettre d'ajouter des sous-missions;
- Permettre d'ouvrir une sous-missions dans un autre onglet;

- Permettre de se rendre dans le code de l'état sur un clic droit.

4.9.3 Réalisation

La réalisation de ce module a été longue et ardu. Tout d'abord, une recherche d'un module pour dessiner les états machine avec leur liaison a été effectué et aucun module simple dans QT n'a été trouver alors les états ainsi que leur liaisons ont été dessiné « à la main » à partir des fonctions onPaint des QWidgets. Cette partie à été longue car il s'agissait de faire les calculs de pixels pour les différentes formes et de faire plusieurs tests. Par exemple, faire une flèche vers un état rectangulaire était très difficile, alors un cercle a été choisi ainsi, peu importe l'orientation de la flèche, elle est toujours à la même distance du centre. Ensuite, listes les états disponibles était une partie de plaisir mais ensuite, il fallait déterminer quels étaient les paramètres d'un état alors un « *parsing* » du fichier a été effectué basée sur la syntaxe du python ainsi que quelque standards qui ont été défini afin de facilité la détection des paramètres dans le code comme par exemple que le nom des paramètres doit débuter par « param_ ».

La réalisation de ce module a prise beaucoup plus de temps que prévu. Dû à ce délai, plusieurs autres modules ont été reportés à plus tard.

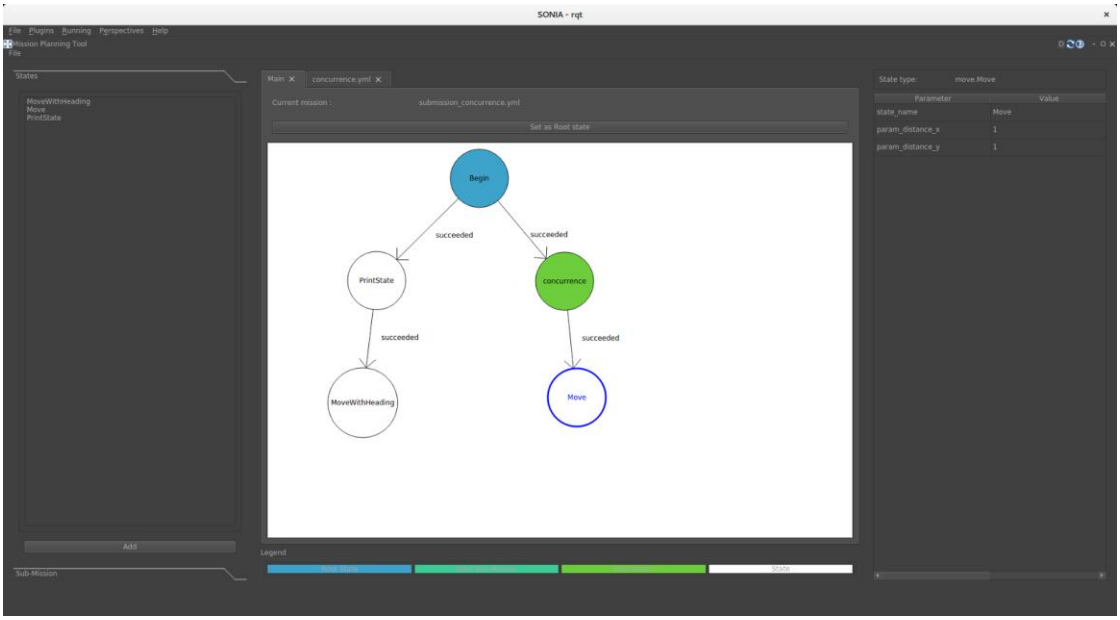

Figure 4.9.1 Éditeur de mission

4.10 Résultat final :

Voici le résultat de la télémétrie lorsque chaque module est ouvert en même temps.

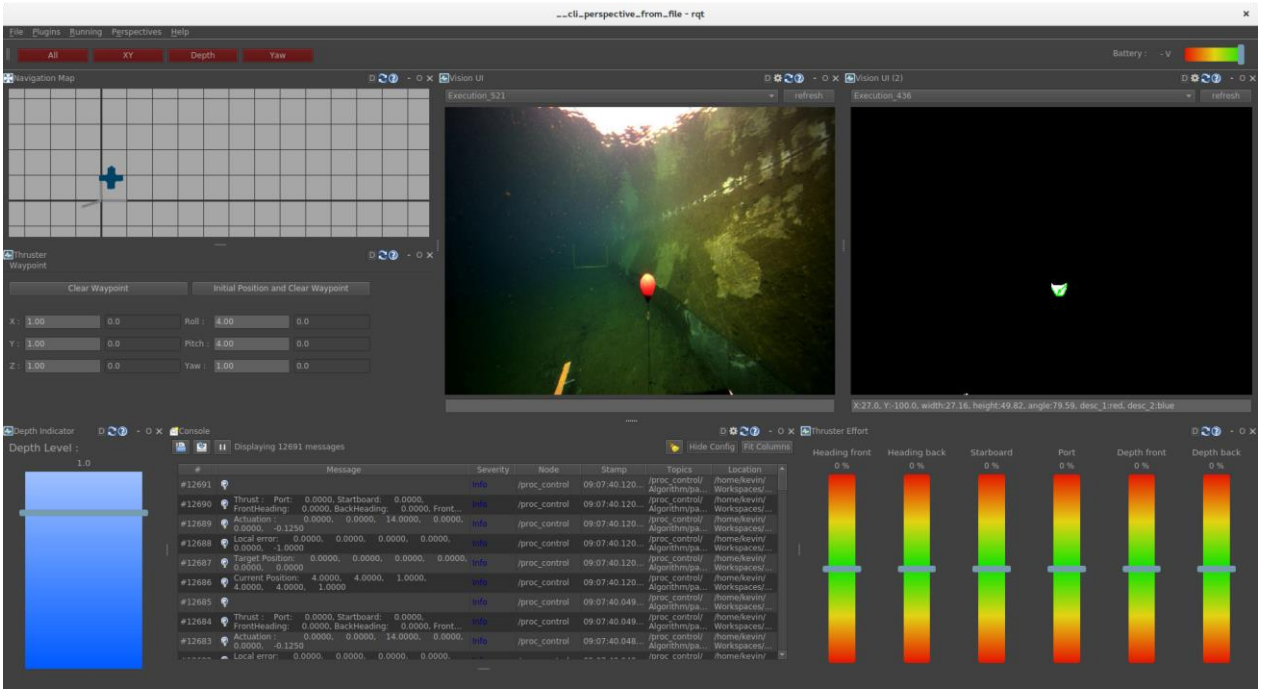

Figure 4.10.1 Télémétrie finale

CHAPITRE 5

TESTS RÉALISÉS

5.1 Pylint

Au cours du projet, il est rapidement évident que l'utilisation du python dans la télémétrie avait un gros désavantage. Lors de l'évolution du code, il allait devenir de plus en plus difficile de trouver les références des classes dans les autres et donc de faire des tests efficaces sur les interfaces utilisateurs au niveau des dépendances.

L'intégration de Pylint au niveau du développement a donc été nécessaire. Pylint est un analyseur statique du code qui fait quelques vérifications utiles. Par exemple, il réalise les imports des modules pour s'assurer que les dépendances existent. Ainsi, si jamais une dépendance change de nom ou autre, il y a un problème. En faisant l'intégration de Pylint sur la compilation de notre solution C++, il y a maintenant une validation du code qui est lancé à chaque compilation du code C++. Ceci est très pratique pour le club puisque souvent il y a de petites refactorisation ici et là et ceci aurait pu causer de grave problème lors de modification rapide en compétition.

L'integration de Pylint a donc apporter une validation supplémentaire au code python et viens assurer la cohérence du code. De plus, Pylint vient assurer un certain standard dans le nom des variables, classes et autre composants de python.

5.2 Simulateur

Afin de faire des tests plus efficaces, un simulateur a été développé simulant une vitesse du sous-marin au niveau des modules du DVL et IMU, les modules responsables de la vitesse et l'orientation du sous-marin, à partir des forces des moteurs demandée par le système de contrôle. Ceci permet donc de faire de belle démonstration d'avancement et de faire beaucoup de test boite noir sur le système.

CHAPITRE 6

MÉTHODOLOGIE

6.1 Agile

La méthodologie agile a été utile dans ce projet. Ceci a permis de faire des démonstrations assez régulièrement aux membres du club S.O.N.I.A. Leur commentaires ont permis de faire quelques ajustements mineurs en cours de route et ainsi améliorer la solution finale.
L'utilisation de la méthodologie agile dans ce projet a été une réussite.

6.2 Gestion du code

La gestion du code a été effectué avec GitHub sur les *repositories* du club S.O.N.I.A.
Les modules sont se trouve tous dans le répertoire rqt_sonia_plugins.
L'utilisation de git a bien été durant ce projet, sans aucun accrochage.
https://github.com/sonia-auv

6.3 Gestion des tâches

En début de projet, l'application YouTrack de JetBrains a été utilisée pour faire la gestion des tâches. Toutefois, il a été rapidement évident que l'application était un peu trop complexe pour un projet de petite envergure. YouTrack se compare un peu à Jira de Atlassian. Il y a tout pour faire des projets de 20 ingénieurs logiciels, et gérer plusieurs projets en même temps. L'utilisation d'un tel logiciel dans un projet seul est un peu trop complexe. C'est pour cette raison que l'application Trello a été utilisées pour la suite du projet.

CHAPITRE 7

APPRENTISSAGE

7.1 Python

Avant le début du projet, le manque de connaissance de python était un gros risque. Toutefois, python est un langage très simple à apprendre et très puissant. Il s'agit d'une corde de plus à mon arc que je suis fier d'avoir acquis.

7.2 Qt

Qt était également un risque en début de projet dû au manque d'expérience avec celui-ci. Toutefois, Qt ressemble à Swing de Java et il a donc été facile de faire des liens entre les deux.

7.3 ROS

ROS a été la marche la plus haute à surmonter dans le début du projet. Afin de l'apprendre plus rapidement, beaucoup de lecture a été faite sur le site de ROS et les tutoriels ont été très pratiques, tout au long du projet.

7.4 Pylint

Pylint est une belle trouvaille. Malgré que la configuration de ce dernier puisse quelque fois être ardue, il est intéressant d'utiliser un tel outil afin de s'assurer de la qualité du travail effectué. Sans Pylint, je vois mal comment un projet peut oser espérer croitre en python. Il s'agit d'un langage de *Scripting* à la base et sa plus grande force est également son plus grand défaut, le fait qu'il ne soit pas compilé.

CHAPITRE 8

DIFFICULTÉS RENCONTRÉES

8.1 Communication avec les membres du club S.O.N.I.A.

Une des difficultés rencontrées au cours de ce projet a été que certains membres importants de l'équipe étaient en stage à la session d'hiver ce qui rendait les communications difficiles. Afin de passer au travers cette difficulté, l'utilisation de Slack a été préconiser mais il en demeure que la communication n'était pas évidente.

8.2 Back-end pas prêt.

Une réalité des clubs scientifiques est que le travail est fait bénévolement, quand les gens ont du temps. Or, assez rapidement dans le projet, il est arrivé que l'interface fût prête mais le module derrière n'était pas encore compléter. Afin de pouvoir tester les modules, des efforts supplémentaires ont donc été fourni afin d'aider les membres à compléter les modules sur lesquels la télémétrie dépendait.

CONCLUSION

Pour conclure ce rapport, voici deux rétrospectives des membres du club S.O.N.I.A :

“On a hâte de l’utiliser en piscine, au mois de Mai !” - Francis Massé, Capitaine

“C’est exactement ce que nous avions en tête.” - Jérémie St-Jules, Responsable Logiciel

---

Suite à une démonstration du résultat à l’équipe, la réponse a été unanime. Le projet est une réussite. Bien entendu, il reste du travail à faire, mais le gros du travail est fait et il est certain que la télémétrie sera utilisée en compétition au mois de juillet cette année.

L’objectif initial du projet de redévelopper un outil permettant l’affichage du statut du sous-marin ainsi que son contrôle est une réussite. Les différents modules de la télémétrie remplissent des besoins très précis, mais ce qui est encore plus intéressant est la facilité de développement de ces modules. Avec l’expérience acquise, près de 500 heures de développement, il est facile de dire qu’un module simple prend environ une journée à développer ce qui est très acceptable et enviable dans le contexte d’un club scientifique. La majorité des membres du club n’aimant pas particulièrement faire des interfaces utilisateurs, ils sont heureux de savoir que le développement est rapide et facile à tester.

En conclusion, l’analyse, les méthodologies utilisées, ainsi que les tests réalisés ont été efficaces et ont permis un développement très rapidement d’un logiciel complet et utile dès maintenant.

LISTE DE RÉFÉRENCES

ROS – Robot Operating System
www.ros.org

GitHub SONIA-AUV
https://github.com/sonia-auv

BIBLIOGRAPHIE

Python 2.7 - https://docs.python.org/2/

QT - http://doc.qt.io/qt-4.8

Pylint - https://www.pylint.org/

Ubuntu - https://www.ubuntu.com/