

Abstract

This document deals about achievements conducted for SII –the “Company for Industrial Information Technology”– as a part of an internship in the engineering school INSA Centre Val-de-Loire. It completes the Security and Information Technologies school programme and lies under the aegis of a partnership between SII and INSA Centre Val-de-Loire. My work in the company is placed under *SII Research* organization. This entity intends to build innovating offers as well as communicating around technologically advanced topics through proof of concept projects. Also called demonstrators, these achievements are financed with SII own resources and must intrinsically address a major group of clients. For their parts, my studies and tasks are directed to Aero-Space Defense sector through the following problem.

In order to carry out specific tasks, robotic systems must rely on consistent environment perception which can be distinguished into different types. On one hand we consider external environment like spatial features or obstacles and, on the second hand, the location of the mobile platform itself in this external environment. These complementary aspects are better known as SLAM, for Simultaneous Localization And Mapping. SLAM algorithms implementation typically requires an external measurement system such as a LIDAR, for Light Detection And Ranging, and internal wheels encoders able to reconstitute odometry records. This report addresses these issues through the specification, the conception and the implementation of software components that intend to control a mobile robot and compute his environment perception in real time. The project described here covers the whole system, from physical acquisition devices to results presentation through a human-machine interface. It relies on the mobile robot *Wifibot Lab* equipped with the *RPLidar A2* LIDAR for spatial distances acquisition. Computations are centralized on a user workstation, where part of processing is handled by a ROS (Robot Operating System) network. Presentation and control interface, for its part, is written in C++ with the Qt Library.

Mots-clés

Cartographie et Localisation Simultanées

Développement logiciel

Temps-réel

Système embarqué

Robot Operating System

Remerciements

En prélude à ce rapport, je tiens à remercier M. David Daumand, ingénieur logiciel au sein de la société SII, qui a défini, tutoré et défendu ce projet. Merci à lui pour le suivi et l'implication sans faille sur lesquels j'ai pu compter durant ces six mois, et ce malgré les efforts organisationnels qui lui incombaient. J'ai tout particulièrement apprécié la passion pour les sujets de robotique et de développement qu'il a su transmettre avec patience et surtout, avec un enthousiasme constant. Ce stage se concluant par une embauche, je ne pourrais que souligner la part importante qu'il a jouée dans le cheminement menant à cette situation et les conseils expérimentés qu'il a pu prodiguer à cet effet.

Je salue sincèrement M. Adel Hafiane, enseignant-chercheur et responsable du laboratoire de vision par ordinateur de l'INSA Centre Val-de-Loire, qui m'a assistée en qualité d'enseignant-référent. Je le remercie d'avoir veillé assidûment sur ce projet, tant dans les conditions matérielles de son déroulement, que dans son orientation technique et pédagogique au regard de ma formation. J'exprime toute ma gratitude à son égard pour nous avoir, Alban Chazot et moi, orientés vers SII pour nos stages de fin d'étude avec la confiance qui était la sienne.

Enfin je remercie l'ensemble de l'équipe qui m'a accueillie avec bienveillance, sympathie et humanité au sein de l'agence de SII Bourges. Merci aux consultants, stagiaires, Responsable de site et collaborateurs qui ont jalonné mon quotidien et enrichi mon expérience professionnelle nouvelle dans ses aspects techniques, organisationnels et bien sûr humains.

Table des matières

Introduction	1
1 L'environnement du stage, la Société pour l'Informatique Industrielle	3
1.1 Principaux repères et évolution de la structure SII	3
1.1.1 Signalétique générale : du groupe à l'agence parisienne	3
1.1.2 Évolution de SII en France et à Bourges	4
1.2 Une mission R&D pour bâtir de nouvelles offres	5
1.2.1 SII Research dans l'agence berruyère	5
1.2.2 Fruits du partenariat avec l'INSA Centre Val-de-Loire	6
2 Définition et mise en oeuvre d'un système de SLAM en temps-réel	7
2.1 De l'expression du besoin à une spécification complète	7
2.1.1 Vision et cas d'utilisation	7
2.1.2 Formalisation d'exigences	8
2.1.3 Matériel et Interfaces	10
2.2 Théorie et choix techniques	12
2.2.1 Cartographie et Localisation Simultanées	12
2.2.2 Le méta-système d'exploitation ROS	13
2.2.3 Hector SLAM	15
2.3 Réalisations et architecture logicielles	17
2.3.1 Le réseau ROS complet	17
2.3.2 Interface Homme-Machine avec Qt Creator	22
2.3.3 Éléments de modularité mis en place	25
3 Bilan et perspectives	29
3.1 Organisation du travail	29
3.1.1 Application du référentiel qualité interne	29
3.1.2 Vers une conduite AGILE adaptée	31
3.2 Résultats en vue d'un prolongement	32
3.2.1 Un module de SLAM en adéquation avec les attentes du projet	32
3.2.2 Pourquoi et comment envisager la continuité de SRT2M ? . . .	34
3.3 Retour d'expérience	36
3.3.1 Apports et difficultés du projet	36
3.3.2 Évolution personnelle au sein de la structure	37
Conclusion	38

Annexes	48
Annexe 1	48
1 Définition du message nav_msgs/LaserScan.msg	48
2 Définition du message std_msgs/Header.msg	49
Annexe 2	50
1 package.xml	50
2 CMakeLists.txt	51
Annexe 3	52
1 Message OCCUPANCY_GRID	52
1.1 Header	52
1.2 Info	53
1.3 Origin	53
1.4 Position	53
1.5 Orientation	53
1.6 Data	53
Annexe 4	54
1 Planning hebdomadaire	54

Table des figures

1.1	BU actives en IDF	4
1.2	Répartition sectorielle de SII en IDF[2]	4
1.3	Wifibot embarquant un RPLidarA2 et une caméra Theta S	6
2.1	Diagramme de cas d'utilisation du système de cartographie et de localisation d'un robot mobile	8
2.2	Exigences de Spécification Technique du Besoin Logiciel	9
2.3	RPLidar A2 utilisé durant le projet, développé par SLAMTEC	10
2.4	Résultats de l'application constructeur du RPLidar A2	10
2.5	Architecture physique du projet	11
2.6	Processus de SLAM classique	12
2.7	Logo du méta-système d'exploitation ROS (issu du kit de presse ROS[21])	13
2.8	Noeuds et topics d'intérêt et actifs lors de l'exécution d'Hector SLAM	16
2.9	Visualisation avec RViz d'un jeu de données pré-enregistrées	17
2.10	Réseau ROS intégré au système	17
2.11	Systèmes de coordonnées et transformations au sein du système	18
2.12	Définition du paquet générique de transmission de données à l'IHM .	19
2.13	Machine à état de l'application Qt	22
2.14	Modélisation logicielle de l'Interface Homme-Machine	23
2.15	Rendu visuel de l'IHM en mode Réception	24
2.16	Rendu visuel de l'IHM en mode rejeu	25
2.17	Boîte de dialogue de configuration du réseau ROS depuis l'IHM	27
2.18	Contrôle du réseau ROS créé dynamiquement sur l'interface graphique	27
3.1	Modèle de définition des modules logiciels et relations entre leurs fonctions	29
3.2	Entrées de la fonction <i>Transmettre : Formater un message</i>	30
3.3	Sorties de la fonction <i>Transmettre : Formater un message</i>	30
3.4	Politique de gestion de branches Git adoptée	31
3.5	À gauche cartographie des locaux avec le logiciel, à droite plan d'évacuation correspondant	33
3.6	Illustration : déploiement et approche furtive de la flotte	35
3.7	Illustration : réception de données unifiées	35
3.8	Contenu du message Occupancy_Grid	52
3.9	Contenu du message Header	52
3.10	Contenu du message Info	53
3.11	Contenu du message Origin	53
3.12	Contenu du message Position	53
3.13	Contenu du message Orientation	53
3.14	Aperçu du planning complet, jalons et dead-lines clés, sans la dénomination des tâches	55

3.15 Détail du planning, partie 1	56
3.16 Détail du planning, partie 2	57
3.17 Détail du planning, partie 3	58

Introduction

Issu d'un partenariat entre SII et l'INSA Centre Val-de-Loire, ce stage vise à la réalisation d'une application logicielle au sein de la société SII, sur le site de Bourges. Il s'inscrit dans le domaine du développement informatique appliqué à la robotique. Les briques logicielles implémentées devront permettre de contrôler un robot mobile, de cartographier son environnement, ainsi que de le localiser en temps réel au travers d'une IHM interactive. Les versants de cartographie et de localisation sont également connus sous le terme de SLAM pour cartographie et localisation simultanées. Ils s'appuient en priorité sur des mesures spatiales, fournies par un LIDAR embarqué sur le robot et –dans une moindre mesure– sur les relevés d'encodeurs présents au niveau des roues du robot. Au travers de sa mise en œuvre, le SLAM constitue une base indispensable à tout système de navigation autonome. Bien que ce point ne fasse pas l'objet de ce rapport, il constitue en grande partie l'essence applicative du présent projet.

Les réalisations attendues s'inscrivent en effet dans un périmètre plus large qu'est la mise en œuvre d'un Système Robotique Tactique Multi-Missions pour la surveillance et l'aide à la prise de décisions dans les milieux à risques (SRT2M). Adressé au secteur de la Défense, ce système en devenir permet à un opérateur humain d'effectuer des missions de contrôle, de surveillance et de recherche en terrain dangereux ou potentiellement dangereux directement depuis un “shelter”, à savoir une zone abritée. L'opérateur dispose à cet effet d'une flotte de robots terrestres et aériens qu'il contrôle à distance. L'exploration du milieu permet d'une part la visualisation de données cartographiques et la localisation des véhicules au sein de la carte, et d'autre part, la réception de flux vidéo émanant de caméras à 360 ° embarquées. Ces flux vidéo sont soumis à des traitements qui permettent de détecter des objets d'intérêt et de les classifier de manière automatique. C'est Alban Chazot, également étudiant ingénieur à l'INSA Centre Val-de-Loire, qui a assuré la réalisation de cette dernière fonctionnalité. Les classifications sont ensuite incrustées sur la carte générée, de manière à ce que l'utilisateur puisse visionner les résultats de SLAM, la classe, la position et la taille des entités détectées en une seule et même zone de rendu.

Le premier chapitre vise à apporter au lecteur une compréhension suffisante du contexte du stage pour en saisir les enjeux. Il s'attache d'abord à décrire la structure d'accueil, à savoir le groupe SII, au sein duquel s'articulent les agences françaises et en particulier, l'agence Île-de-France dont dépend le site de Bourges. À ce titre, sa portée et son organisation seront particulièrement détaillées.

Un deuxième chapitre donnera les étapes les plus conséquentes de la réalisation. Cette restitution consistera d'abord à définir clairement le livrable attendu à la fin du stage notamment au travers de l'analyse fonctionnelle du produit. Une partie viendra étayer les choix techniques ou théoriques qui constituent la base de l'implémentation, comme les principes fondamentaux du SLAM ou l'utilisation de ROS. Puis nous

discuterons la façon dont ces principes ont été intégrés au projet, au travers d'éléments qui relèvent de la mise en œuvre architecturale et du fonctionnement général de l'application résultante.

Ce document sera complété d'un chapitre destiné à exposer les stratégies organisationnelles mises en place durant les différentes phases d'avancement du stage. Nous décrirons également les résultats obtenus et les suites envisagées pour le projet, ainsi que les perspectives individuelles au sein de la structure d'accueil.

Enfin, ce rapport s'accompagne d'un glossaire visant à détailler les acronymes, termes techniques ou peu communs que le lecteur est susceptible de rencontrer.

Chapitre 1

L'environnement du stage, la Société pour l'Informatique Industrielle

1.1 Principaux repères et évolution de la structure SII

1.1.1 Signalétique générale : du groupe à l'agence parisienne

Couramment nommée SII, la Société pour l'Informatique Industrielle est une SA à directoire et conseil de surveillance, aujourd'hui implantée dans dix-huit pays, sur quatre continents. Sur l'exercice 2015/2016, le groupe SII enregistre un chiffre d'affaires consolidé de 360,1M€[1], pour un résultat net de 13.13M€[1], et compte sur un effectif moyen de 5 226 collaborateurs[1].

Depuis plus de 30 ans[1][2], SII œuvre pour s'inscrire en tant que partenaire technologique de choix auprès d'une clientèle professionnelle diversifiée. Le groupe –qui a consolidé son expérience dans quatorze secteurs d'activités distincts– propose des offres liées aux savoir-faire suivants :

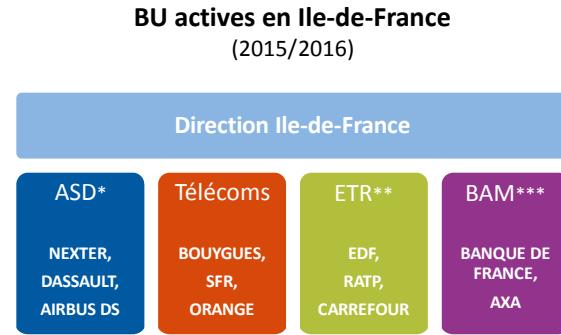
- **l'informatique embarquée** incluant le développement de logiciels embarqués, de contrôle commande ou encore de bancs de tests
- **l'ingénierie scientifique** qui balaye les champs de l'électronique, du traitement du signal et de la mécanique
- **les NTIC**, englobant les problématiques de téléphonie, de web, de mobilité ou d'infrastructures diverses
- **les Systèmes d'Informations** qui recouvrent l'informatique décisionnelle, financière ou la sécurité du SI

Les secteurs d'activités s'organisent quant à eux en BU. Chacune de ces unités s'adresse à un groupe de clients majeurs et détient un organigramme interne, placé sous la responsabilité du Directeur de BU.

L'agence Île-de-France est l'une des neuf agences françaises de la société. Elle est gérée par son Directeur, M. Didier Bonnet. Son administration est abritée sur le site

de Kennedy¹ – principalement dédié aux opérations de gestions – et recouvre deux sites supplémentaires : celui du Dynasteur² et celui de Bourges³ au sein duquel s'est déroulé ce stage.

Les activités de réalisations se concentrent pour cette agence autour de quatre secteurs d'activités qui sont présentés figure 1.1.



*Aero-Space Defense, **Energie Transport Retail, ***Banque Assurance Mutuelle

FIGURE 1.1 – BU actives en IDF

Pour ces secteurs, un portefeuille de dix clients est détenteur de 80% du chiffre d'affaires annuel de l'agence. La figure 1.1 donne la nomenclature des BU définies par l'entreprise, et illustre la clientèle française qui leur est associée.

1.1.2 Évolution de SII en France et à Bourges

SII a été créée à Paris en 1979 par un ingénieur, Bernard Huvé, qui en est aujourd'hui le Président du Conseil de Surveillance. S'en suivra la création de multiples agences provinciales, jusqu'en 1999 où, forte de 400 collaborateurs et d'une croissance soutenue, la société s'introduit en bourse. On peut citer l'internationalisation de SII, qui intervient en 2006 avec l'ouverture d'une filiale en Pologne, pour atteindre aujourd'hui un total de 17 filiales à l'étranger[1].

Dans ce qui va suivre, nous nous concentrerons uniquement sur l'implantation française de SII, qui se retrouve à présent au travers 22 sites sur le territoire.

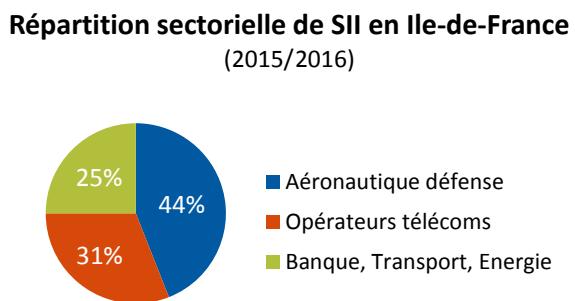


FIGURE 1.2 – Répartition sectorielle de SII en IDF[2]

Aujourd'hui, les secteurs d'activités clés de l'entreprise peuvent être classés selon deux catégories[2], que la figure 1.2 permet de justifier :

1. 104 Avenue du Président Kennedy, 75016 Paris
2. 6, 12, 10 Rue Andras Beck, 92360 Meudon
3. 14, allée Charles Pathé, 18000 Bourges

- **les secteurs stratégiques**, que sont ASD et Télécoms – étant des générateurs forts de valeur, ils concernent des marchés mûrs où SII a su établir des relations pérennes avec ses clients grand-compte
- **les secteurs de conquête**, inclus dans les BU ETR et BAM, amenés à se digitaliser massivement

Dirigée par M. Fabrice Bosch, l'agence berruyère compte quant à elle une cinquantaine de collaborateurs répartis sur les secteurs d'activités ETR et ASD. Elle héberge également des activités de QHSE sous la responsabilité de M. Jean-Sébastien Salis.

Les activités ETR se concentrent sur le développement de solutions billetiques pour des acteurs majeurs du transport de voyageurs. En parallèle, les activités ASD sont tournées vers un ensemble d'acteurs du secteur de l'armement et de la Défense historiquement implantés à Bourges et, aujourd'hui encore, essentiels à l'économie locale[4]. Parmi ces clients on peut citer la filiale missilière du groupe Airbus MBDA, ou encore Nexter Munitions dont le site de Bourges accueille les ingénieurs et cadres affiliés aux missions de R&D . Enfin, et au regard du caractère critique des missions relevant de la défense nationale ou européenne, on notera que le site de Bourges dispose d'une habilitation à mener au sein de son infrastructure des projets classés Confidentiel Defense.

1.2 Une mission R&D pour bâtir de nouvelles offres

1.2.1 SII Research dans l'agence berruyère

La mission qui m'a été confiée durant ce stage est placée sous l'égide de l'entité SII Research. Sous la responsabilité du Pôle Innovation, cette structure permet de mener à bien des projets transverses aux différents secteurs d'activités, financés sur les fonds propres de l'entreprise.

Typiquement, ce type de projet sera qualifié de POC ou démonstrateur. Pour l'entreprise d'accueil, les réalisations sont menées avec les objectifs suivants :

- valoriser l'expertise de l'entreprise dans des domaines techniques précis et actuels
- élargir les connaissances internes en communiquant autour des savoir-faire sollicités
- bâtir de nouvelles offres à partir de tout ou partie du POC

En outre, ces réalisations permettent de communiquer techniquement avec des acteurs internes et des parties prenantes externes –clients, prospects ou étudiants par le biais de médias sociaux– sans compromettre la confidentialité de projets clients.

Ce stage s'inscrit quant à lui dans la mise en œuvre d'un “Système Robotique Tactique Multi Missions pour le Surveillance et l'Aide à la Prise de Décisions dans les Milieux à Risques”, adressé au secteur de la Défense et, plus particulièrement, aux clients berruyers de la BU ASD. Dans cette optique, le projet s'est d'emblée accompagné de l'objectif ambitieux d'être montré à des représentants de l'entreprise Nexter Systems, à son terme ou durant des stages ultérieurs. Ainsi, plusieurs jalons ont pu être posés avec succès dans le respect des procédures internes de validations

hiérarchiques. Il a notamment pu être présenté à MM. Giraud-Sauveur et Boucher respectivement commercial / chargé d'affaires sur le périmètre MBDA / Nexter et Directeur de projets Nexter. Accompagné d'une démonstration, cet entretien s'est soldé positivement et a permis au POC d'être relayé par la suite auprès des instances susceptibles d'assurer son pilotage futur.

1.2.2 Fruits du partenariat avec l'INSA Centre Val-de-Loire

Ce stage –comme celui d'Alban Chazot– s'effectue dans le cadre d'un partenariat entre SII et l'INSA Centre Val-de-Loire. La relation établie entre ces deux entités découle de la convention "Carré des partenaires" acceptée par les deux parties en 2015. Quentin Ménart, aujourd'hui ingénieur logiciel au sein de l'agence SII Bourges nous a précédés en expérimentant l'année dernière un stage issu du même partenariat.



FIGURE 1.3 – Wifibot embarquant un RPLidarA2 et une caméra Theta S

Cette relation privilégiée conduit à retrouver SII en qualité d'acteur ou de sponsor de certains évènements majeurs de la vie étudiante : Nuit de l'Info, Forum des entreprises, conférences techniques. Elle conduit aussi à favoriser l'accueil de stagiaires et d'alternants en provenance de l'institut.

En ce qui nous concerne, cette collaboration a impacté nos stages de manière visible et fructueuse. En effet, elle a permis le prêt d'un robot mobile par M. Benali Abderraouf, enseignant-chercheur en robotique à l'INSA Centre Val-de-Loire. Ce robot qui a constitué la base du projet est présenté sur la figure 1.3. Dans ce cadre, l'INSA Centre Val-de-Loire a également financé pour moitié le matériel nécessaire au projet, à savoir une caméra Theta S à 360 ° (cf. figure 1.3).

Le partenariat a aussi donné lieu à des réunions régulières de co-pilotage de projet réunissant MM. Hafiane et Bosch. Elles nous ont permis de réfléchir ensemble au cap à donner, au matériel à acquérir ou encore aux techniques à employer. Enfin, nous avons pu évoluer sereinement en qualité de stagiaires, en comptant sur le suivi fréquent de M. Adel Hafiane, tant dans la mise en place du projet, que durant les phases plus tardives de développement.

Chapitre 2

Définition et mise en oeuvre d'un système de SLAM en temps-réel

2.1 De l'expression du besoin à une spécification complète

2.1.1 Vision et cas d'utilisation

Les réalisations effectuées ont été guidées par une phase d'analyse du besoin, menée de concert avec les membres de l'équipe projet. En qualité de tuteur de stage, M. David Daumand a guidé ce projet afin que le système final implique des perspectives commerciales concrètes. Dans cette optique, il a apporté son expertise sur ce que seraient les besoins de clients potentiels, en orientant la démarche vers une application militaire. Il a ainsi défini la dénomination du système SRT2M, permettant à elle seule d'exprimer la fonction, le contexte et le secteur visé par le produit.

La preuve de concept réalisée durant ce stage est un sous-système de SRT2M. Afin d'en exposer clairement le périmètre, la figure 2.1 définit sa nomenclature et en donne les principaux cas d'utilisation.

Ainsi, ce *sous-système* doit permettre d'effectuer une cartographie de l'environnement d'un robot mobile, tout en donnant sa localisation. L'acteur primaire du système pourra visualiser une carte, soit à partir de données préalablement sauvegardées au sein de ce même système, soit grâce à des données acquises en temps réel par des périphériques dédiés. Cette visualisation implique la représentation d'obstacles statiques dans l'espace exploré, mais aussi la position et la trajectoire du robot en tout temps. Le téléguidage du robot par l'utilisateur doit également être assuré. Un opérateur –typiquement un contributeur du système– pourra jouer sur les paramètres de présentation des résultats, de la mise en réseau des périphériques ou de la définition de ces périphériques.

Divers périphériques sont à ce stade réunis en un seul acteur “Périphérique matériel”. Il inclut le LIDAR et le robot qui permettent respectivement d'acquérir les mesures de distances aux obstacles et de déplacer la plateforme. On souligne aussi la présence d'un “Composant logiciel” qui interagit avec le système. Ce composant est vu comme une boîte noire qui fournit en sortie les résultats de détection et de classification d'objets d'intérêt rencontrés par le système. Dans les faits, ces résultats correspondent

LOCALISATION ET CARTOGRAPHIE DE L'ENVIRONNEMENT D'UN ROBOT MOBILE

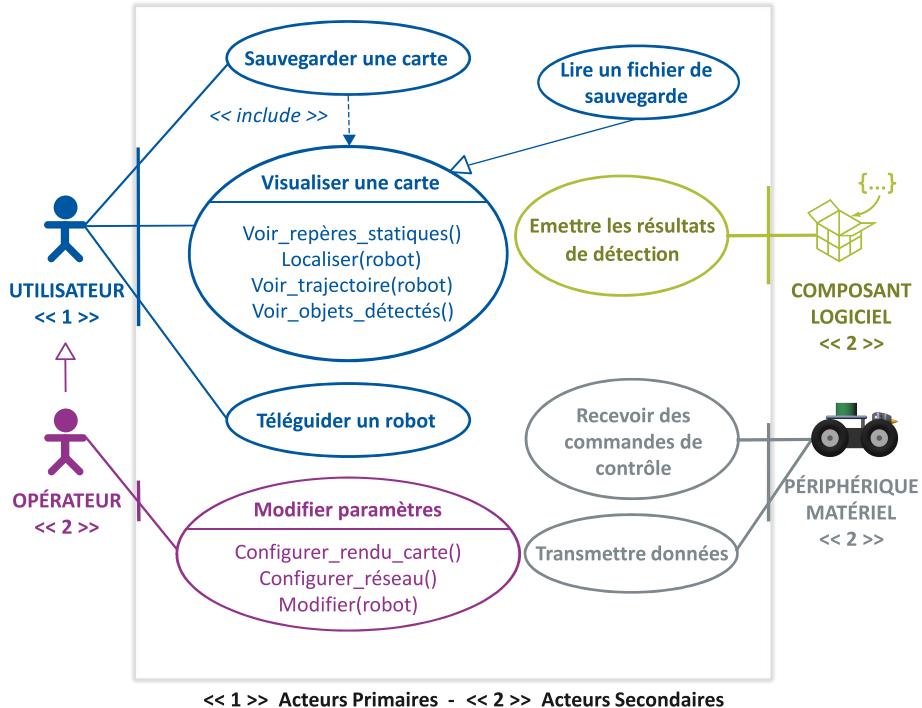


FIGURE 2.1 – Diagramme de cas d'utilisation du système de cartographie et de localisation d'un robot mobile

à la position, la taille et la classe de chaque objet, ce qui nous permettra de les représenter sur la carte générée (cf. fonction `Voir_objets_detectés()`).

2.1.2 Formalisation d'exigences

Les réalisations s'articulent autour d'exigences formalisées au sein d'un document de spécification du besoin logiciel (STBL) dont le but et la portée seront décrits dans la section Organisation du travail. La figure 2.2 reprend les déclarations de haut-niveau qui permettent de cerner les principaux axes de l'analyse fonctionnelle. Les exigences exposées s'appliquent à un ou plusieurs composants du système¹.

Chaque exigence est une entrée du tableau, que l'on associe à l'une des catégories suivantes :

- **fonctionnelle** réuni les besoins métiers qui justifient la réalisation du logiciel
- **performance** regroupe les contraintes spécifiques liées au temps d'exécution ou à l'utilisation des ressources
- **design et conception** donne les impératifs en termes de rendu visuel
- **opérationnelle** traite des interactions survenant pendant l'exploitation du système

On attribue une référence unique permettant le suivi unitaire des exigences lors des différentes phases du projet, une description textuelle non ambiguë ainsi que le niveau de criticité de l'exigence : "C" signifie critique, "I" important et "S" souhaitable.

1. Par opposition aux exigences unitaires –allouées à un et un seul composant– abordées au chapitre Bilan et perspectives

Fonctionnelles		
Référence	Description	Valeur
FNC001	Visualisation en 2D de l'environnement du robot (cartographie)	C
FNC002	Localisation du robot sur la carte	C
FNC003	Visualisation de la trajectoire du robot	C
FNC004	Commande des périphériques à distance	C
FNC005	Téléguidage du robot au clavier et joystick virtuel	I
FNC006	Journalisation selon plusieurs niveaux de criticité	I
FNC007	Sauvegarde des données de cartographie et de localisation	S
FNC008	Rejet de données sauvegardées	S
FNC009	Représentation des objets détectés et classifiés sur la carte, à l'échelle	S
FNC010	Centralisation du contrôle de l'ensemble des briques logicielles dans l'IHM	S
FNC011	Modularité des informations réseau pour les périphériques	S
Performance		
PRF001	Rendu visuel en quasi temps réel	C
PRF002	Navigation fluide dans le contexte graphique	C
Design et Conception de l'IHM		
DCC001	IHM ergonomique et intuitive	C
DCC002	Représentation indépendante des éléments cartographiques au regard du flux vidéo	C
DCC003	Présence d'un panneau de contrôle des modules	C
DCC004	Présence de témoins de statut des modules	I
DCC005	Présence d'une console de journalisation dans le panneau de contrôle	I
DCC006	Modification des modules (nom, exécutable, arguments) depuis l'IHM	S
DCC007	Rendus visuels aux 3 ^{ème} et 1 ^{ère} personnes	S
Opérationnelle		
OPE001	Communication sans fil entre le robot et le poste de travail	C
OPE002	Sécurisation des communications avec les éléments distants	S
OPE003	Stabilité des périphériques sur la plateforme mobile	S

FIGURE 2.2 – Exigences de Spécification Technique du Besoin Logiciel

À ce niveau, les exigences ne suggèrent pas d'implémentation ou de choix techniques. Par contre, elles guident l'ensemble de la réalisation, depuis la conception jusqu'aux tests, en passant par l'estimation de la satisfaction du client. Chacune d'entre-elles s'accompagne donc d'un paragraphe servant à la détailler, à en donner les pré-requis et les cas de validation. Par exemple, on précisera que la FNC001 nécessite la possession d'un matériel d'acquisition de mesures spatiales ou, à défaut, de données de test simulant cette acquisition.

Dans ce qui va suivre, nous nous concentrerons sur les exigences qui apportent la valeur intrinsèque de l'ouvrage, en mettant l'accent sur les niveaux de criticité élevés : à savoir celles qui permettent d'établir et de représenter une cartographie de l'espace et d'y localiser le robot en tout temps. La définition de ces exigences a permis d'entamer une phase de définition des architectures physiques et logicielles sereinement, sans perdre de vue l'ensemble des fonctionnalités à assurer et leur priorité respective.

2.1.3 Matériel et Interfaces

Les fonctionnalités principales de l'application dépendent de l'acquisition de mesures spatiales. Ainsi, nous devons nous doter d'une dispositif capable de prendre ces mesures et de les relayer en quasi temps-réel jusqu'à un module de traitement, encore à définir.



FIGURE 2.3 – RPLidar A2 utilisé durant le projet, développé par SLAMTEC

Le choix du matériel retenu a fait l'objet d'une veille technologique visant à comparer les différents LIDAR présents sur le marché. Ceux-ci ont pu être classés, outre selon leur prix, au regard des critères suivants :

- la technologie employée (1D, 2D, 3D)
- la documentation disponible (notamment en ce qui concerne l'utilisation de la SDK)
- la précision angulaire
- la vitesse de calcul et de transfert des mesures
- la pérennité du produit et/ou de la marque en gage de qualité
- la disponibilité du produit en France, selon des délais serrés

Le résultat de ce comparatif a débouché sur l'acquisition du RPLidar A2 visible sur la figure 2.3. Celui-ci est équipé d'un seul faisceau laser qui a la particularité de tourner sur 360° . Cet équipement est donc adapté à la mesure des distances dans un espace plan, tout en impliquant une dépense acceptable. La possibilité d'équiper le robot de

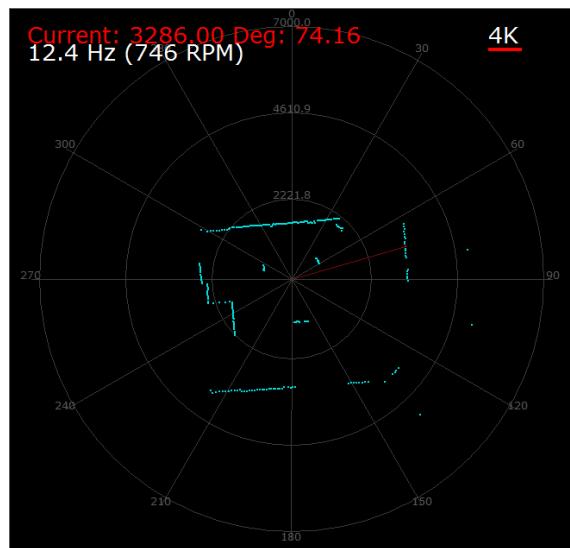


FIGURE 2.4 – Résultats de l'application constructeur du RPLidar A2

servo-moteurs et d'un LIDAR à faisceau unique fixe moins onéreux a également été explorée, mais n'a pas été retenue au regard de la complexité de mise en œuvre d'un tel dispositif, du coût cumulé du LIDAR et des servo-moteurs et enfin, des fortes imprécisions potentiellement engendrées lors du mouvement du robot. La figure 2.4

montre le nuage de points restitué par un logiciel intégré au kit de développement du RPLidar A2, utilisable uniquement sous Windows. Le robot mis à disposition du projet est un Wifibot Lab, muni d'une carte mère Windows Embedded. Enfin, un nano-ordinateur Raspberry Pi 3 modèle B est utilisé à des fins de communication entre les briques embarquées et une station de travail qui réunit les modules de calcul et l'IHM.

Nous obtenons ainsi l'archctecture physique schématisée sur la figure 2.5, au travers de laquelle sont définis quatre modules principaux, notés M_i .

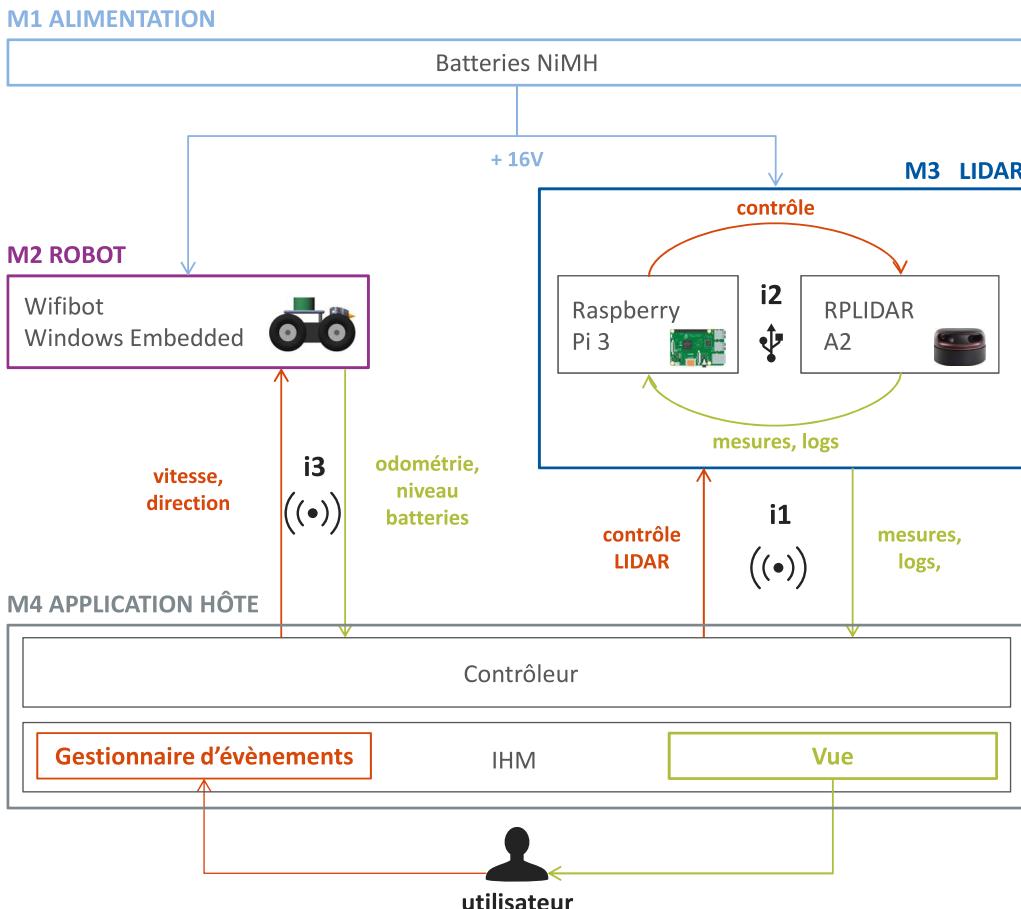


FIGURE 2.5 – Architecture physique du projet

M_1 est dédié à l'alimentation électrique du robot, du LIDAR et du Raspberry Pi. M_2 correspond au Wifibot, M_3 regroupe le LIDAR et le Raspberry Pi et M_4 symbolise l'application hôte sur une station de travail Linux².

Les interactions entre les briques matérielles sont représentées par des flèches d'entrées / sorties de couleur orange quand il s'agit de requêtes et de couleur verte pour les réponses afférentes. Des interfaces de communications inter-modules notées i_i apparaissentent en noir et définissent les canaux au travers desquels requêtes et réponses sont transmises.

L'application hôte est responsable de l'émission de commandes de contrôle du LIDAR au travers d'une interface WiFi³ i_1 . Ces commandes transitent par le Raspberry Pi qui les formattera et les transmettra au LIDAR via la liaison UART↔USB i_2 . Ce

2. Ce dernier module présente un intérêt seulement en termes de transmission de données avec les périphériques matériels

3. Le Raspberry Pi 3 est nativement doté d'une carte et d'un émetteur WiFi

cheminement est parcouru dans le sens inverse pour remonter les données acquises par le LIDAR, à savoir les mesures de distances spatiales communiquées tous les 360° contenant des couples de la forme (θ, d) , avec :

$$\begin{cases} \theta \in [0, 360[, \text{en degrés} \\ d \in [0.5, 6] \cup \infty , \text{en mètres} \end{cases}$$

Parallèlement, M_4 communique directement des ordres de vitesse et de direction au Wifibot à travers l'interface WiFi i_3 . Ces commandes de direction sont quasi-instantanément suivies de réponses contenant les relevés odométriques des encodeurs du Wifibot, son niveau de batterie et le relevé de capteurs infrarouges présents à l'avant du robot. Les réponses empruntent là encore le chemin inverse par le biais de la même interface.

2.2 Théorie et choix techniques

2.2.1 Cartographie et Localisation Simultannées

Le SLAM, pour Simultaneous Localization And Mapping, est un ensemble de méthodes permettant à un robot mobile d'établir une cartographie de son environnement et de se repérer de manière fiable dans cette même carte.

De manière générale, on peut dégager un schéma autour duquel s'articulent ces techniques, dont la figure 2.6 donne les grandes lignes.

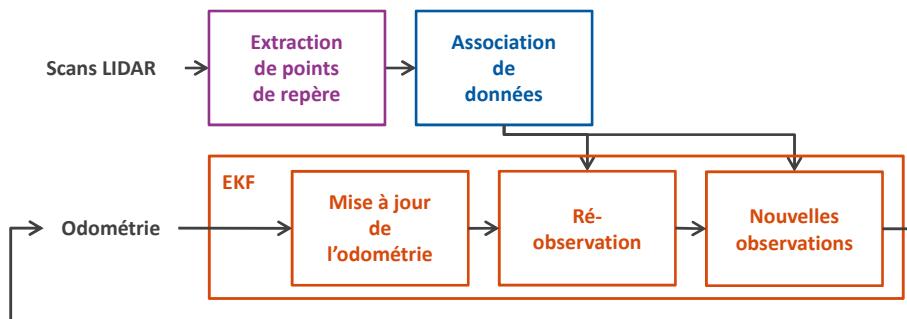


FIGURE 2.6 – Processus de SLAM classique

La première étape d'un processus de SLAM consiste à récupérer les données numériques issues d'un laser. Généralement, on peut décrire de telles données en termes de précision, de champs de capture, de longueur du faisceau ou encore de résolution verticale. Le RPLidar A2 dispose d'un champ de 360° sur l'axe horizontal, d'une fréquence de rotation typique autour de 600 rpm, d'une portée comprise entre 0.15 et 6m et pour finir, d'une résolution angulaire comprise entre 0.45° et 1.35° .

Les données odométriques –ou odométrie– permettent la prédiction de la position du robot dans le plan. Ce calcul se base typiquement sur le différentiel de vitesse entre les deux roues motrices, fourni par des encodeurs internes au robot. La difficulté réside lorsqu'il s'agit de corrélérer dans le temps et avec précision ces données avec les mesures en sortie du laser. Pour pallier ce problème, l'odométrie peut être extrapolée de l'instant précédent dans une démarche prédictive.

Une fois les données externes collectées, il convient de les traiter pour en extraire des points de repères spatiaux. Cette étape, représentée en violet sur la figure 2.6, consiste à caractériser les points de repères pertinents. Ils sont ensuite gardés en mémoire pour établir la cartographie de l'espace et également en vue d'être discriminés ou assimilés ultérieurement. Plusieurs algorithmes, tels que Spike ou RANSAC, proposent ces fonctionnalités avec des approches diverses qu'il convient d'adapter à une application donnée[5].

L'étape suivante, représentée en bleu sur la figure 2.6, vise à associer les données collectées, à savoir reconnaître les points de repères qui arrivent dans le champs d'observation du robot à plusieurs reprises. Le problème de l'association de données vise à prendre en compte du mieux possible les risques de confusion ou de non reconnaissance de points de repères au travers d'une politique de validation qui encadre la fusion de données. À cet effet, on peut appliquer l'algorithme KNN implémentant des mesures de distances diverses comme la distance Euclidienne ou la distance de Mahalanobis.

Enfin, un EKF est utilisé pour estimer un vecteur d'état du robot décrivant sa position et son orientation. Dès lors que les briques d'extraction de points de repères et d'association de données sont mises en place, un processus de SLAM peut être considéré en trois étapes, présentées en orange sur la figure 2.6. Le premier point consiste simplement à ajouter les relevés odométriques du robot à l'ancien vecteur d'état. Le deuxième point apporte une correction au vecteur d'état et à l'état du système de façon globale. En effet, au vu de l'état courant, on peut estimer la position des points de repères précédemment sélectionnés. Si un décalage spatial a lieu, on l'appellera innovation, c'est-à-dire la différence entre la position estimée du robot à l'étape 1 et sa position basée sur la perception de son environnement. De plus, cette étape nous permettra d'affiner la confiance accordée dans notre connaissance de l'environnement. Une innovation faible va entraîner une confiance accrue quant aux positions des points de repères connus et visibles, tandis qu'une innovation forte aura l'effet inverse. Le troisième point permet d'ajouter à notre système les points de repères détectés ayant été discriminés par la politique d'association mise en place.

La définition et l'implémentation d'un algorithme de SLAM est un processus fastidieux qui repose sur des fondements théoriques complexes. La diversité de ces algorithmes[6] dénote l'ampleur de la tâche qui consiste à les recenser, à en comprendre tous les rouages pour proposer des améliorations et finalement passer à l'implémentation. Ainsi il n'était pas envisageable de développer un algorithme de SLAM dédié au projet. Par contre, l'utilisation de tels algorithmes paraissait incontournable au regard des exigences du stage. Un état de l'art sur le sujet a donc été mené, permettant de choisir l'algorithme à intégrer, présenté à la section 2.2.3.

2.2.2 Le métasystème d'exploitation ROS



FIGURE 2.7 – Logo du métasystème d'exploitation ROS (issu du kit de presse ROS[21])

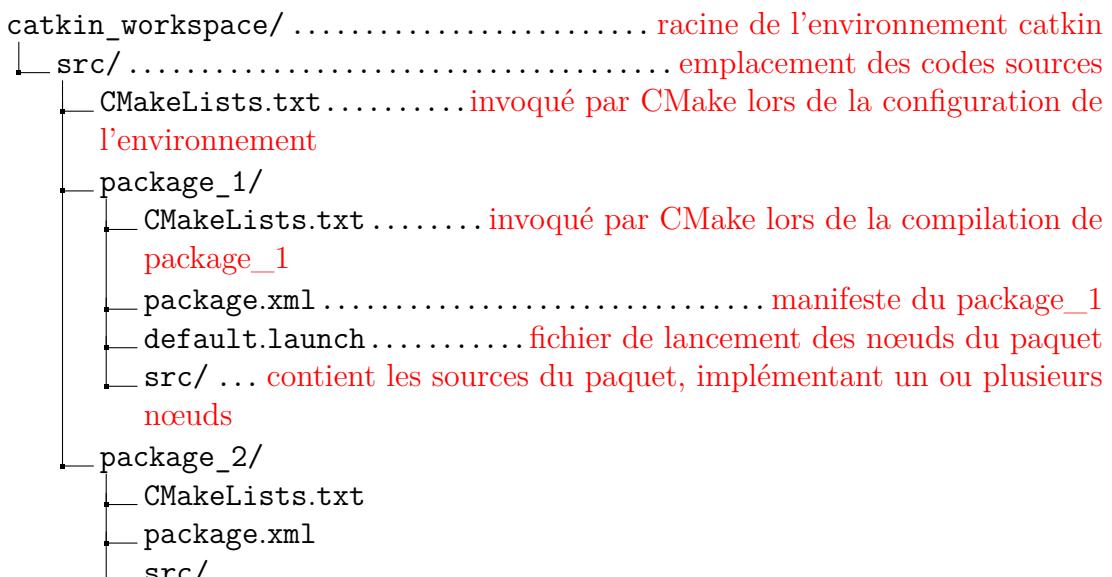
Robot Operating System, ou ROS, est un middleware pour systèmes Unix destiné à la réalisation de projets robotiques écrits en C++ ou en Python. Il offre des

fonctionnalités liées à la robotique grâce à des librairies dédiées mais aussi des moyens de communication et des outils facilitant le développement et le déploiement de tels systèmes. Il a ainsi été rapidement étudié puis adopté pour ce projet sous sa version Kinetic. Notre utilisation de ROS couvre les briques logicielles embarquées sur le Raspberry Pi et des briques présentes localement sur le poste de travail de l'utilisateur final. Son fonctionnement passe par la création d'un réseau de noeuds[8], c'est-à-dire des exécutables dans l'écosystème ROS, organisés autour d'un noeud central appelé ROS Master[7]. Cette entité enregistre les noeuds, leur fournit un nom et instancie un annuaire qui leur permet d'échanger des données. Il s'appuie en partie sur le protocole XML-RPC.

Les noeuds disposent de plusieurs moyens de communication, disponibles à travers une librairie cliente fournie par ROS : les topics[9] auxquels il est possible de s'abonner, les services[10] qui fonctionnent sur le modèle requête / réponse, et des valeurs stockées par un serveur de paramètres géré par le ROS Master et qui peuvent être récupérés par les noeuds durant leur exécution. On notera que, dans le cas des topics, le format des données transmises correspond généralement à des types de messages définis par ROS dans ses packages internes⁴. Par exemple, le type de données `sensor_msgs/LaserScan` dispose d'un fichier de définition du même nom, portant l'extension `.msg` et d'un fichier d'en-tête qui permet la publication et l'abonnement aux topics de ce type. À titre d'exemple, le fichier `sensor_msgs/LaserScan.msg` est présenté en Annexe 1.

Le framework s'accompagne de nombreux utilitaires en ligne de commande qui permettent à l'utilisateur de configurer le réseau, de le sonder et de créer des noeuds à la volée pour diverses utilisations.

ROS dispose également d'un outil de gestion du système de fichiers et de chaîne de compilation nommé Catkin. Ce dernier permet la mise en place d'un format d'arborescence facilitant la production et la gestion de packages. Ces packages sont définis comme des agglomérats logiques de noeuds et possèdent une structure spécifique au sein de l'environnement de travail Catkin, tel que représenté ci-dessous.



Le fichier `package.xml`[11] est le manifeste des paquets. Placé à la racine de chacun

4. Lors d'une installation standard, ces paquets se situent sous `/opt/ros/<ROS-version>/share/` pour les messages et `/opt/ros/<ROS-version>/include/` pour les fichiers d'en-tête

d'entre-eux, il en donne la définition (nom, version, auteur, license) ainsi que les dépendances. Les fichiers CMakeLists.txt[12] sont les entrées du système de build CMake, invoqués dans la chaîne de compilation de Catkin. C'est ici que l'on définit les noms et fichiers sources associés à chaque nœud du paquet, les sources étant généralement stockées sous le répertoire `catkin_workspace/src/<package_name>/src/`. L'Annexe 2 permet d'illustrer le formalisme des documents package.xml et CMakeLists.txt au travers d'exemples issus du code source du projet.

Les deux éléments précédents sont indispensables à la création de paquets dans l'environnement Catkin. Parallèlement, on relève la présence d'un fichier portant l'extension `.launch`, appelé *launchfile*. Ce fichier optionnel permet d'exécuter, via un utilitaire en ligne de commandes, plusieurs noeuds en une seule fois. Pour parser et exécuter le *launchfile* présent dans notre exemple on saisira simplement :

```
1 > rosrun package_1 default.launch
```

Les briques logicielles implémentées avec ROS présentent intrinsèquement un fort potentiel de modularité et de maintenabilité. En effet, chaque package et chaque nœud observent des cycles de vie indépendants et n'interagissent les uns avec les autres qu'en cas d'échange d'informations utiles. Cet aspect modulaire, sur lequel nous reviendrons, a fortement participé à l'adoption de ROS et de ses outils. De plus, ROS est entretenu par une large communauté de chercheurs et développeurs qui alimentent fréquemment l'écosystème par de nouveaux paquets, généralement mis à l'épreuve lors de compétitions robotiques internationales. Ainsi, la problématique de SLAM est largement couverte par ROS, tant et si bien que les librairies et outils publics dédiés au SLAM se retrouvent presque exclusivement sous la forme de paquets ROS.

2.2.3 Hector SLAM

Dans le cadre de ce projet un état de l'art a été entrepris, visant à recenser les principaux algorithmes de SLAM intégrables à notre système logiciel. Les algorithmes majeurs du domaine ont donc été étudiés et qualifiés notamment au travers des applications qu'ils visent, de leur principe de fonctionnement, des possibles implementations qu'elles connaissent et de leurs limites. Suite à cette étude, le projet Hector SLAM a été retenu. Notons qu'il profite d'une notoriété acquise depuis plusieurs années, lui permettant d'être aujourd'hui encore largement actif et reconnu[14][15].

Hector SLAM est un métapackage⁵ ROS[16] qui inclut nativement trois packages principaux : `hector_mapping`, `hector_geotiff` et `hector_trajectory_server`. Nous ne reviendrons pas sur `hector_geotiff` responsable de l'enregistrement des données au format GeoTIFF qui a été écarté du système final. `hector_mapping` est le moteur de SLAM intégré par Hector. Il a la particularité de ne pas requérir de données odométriques pour fonctionner. Cela sous-entend que l'on peut disposer de résultats de SLAM en tenant simplement le LIDAR dans ses mains. Ce point a été déterminant dans la sélection d'Hector SLAM puisque d'une part l'acquisition du robot s'est faite tardivement, et d'autre part, ce dernier a été fourni sans documentation permettant de s'assurer de la précision des encodeurs. Nous avons donc préféré un système de SLAM qui favorise les données issues du LIDAR fraîchement acquis. Par ailleurs,

5. Un métapackage est simplement pourvu d'un manifeste, il permet de référencer d'autres packages regroupés logiquement mais faiblement couplés d'un point de vue fonctionnel

Hector SLAM s'adapte à une utilisation sur des drônes par le biais de données IMU. Dans l'optique d'assurer une continuité au projet en l'orientant vers le secteur de la défense, la possibilité d'adapter le système à des dispositifs aériens semblait particulièrement intéressante. Enfin, le noeud `/hector_trajectory_server` est utilisé pour traiter la succession de positions de la plateforme mobile depuis le début de l'acquisition.

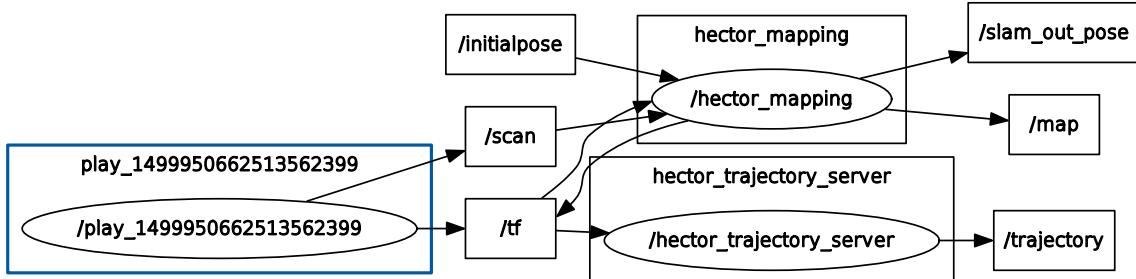


FIGURE 2.8 – Nœuds et topics d'intérêt et actifs lors de l'exécution d'Hector SLAM

La figure 2.8 expose les nœuds et topics actifs lors d'un l'utilisation d'Hector SLAM à partir d'un jeu de données pré-enregistrées dans un fichier bagfile. Les nœuds actifs sont présentés au sein d'ellipses, les topics au sein de rectangles sous la forme `/<topic-name>` et les noms des packages auxquels les nœuds appartiennent figurent au dessus de chacun d'entre-eux. Enfin, les flèches signalent quels sont les nœuds responsables de la publication des topics, et quels sont les nœuds qui s'y sont inscrits. Le package généré par la lecture du bagfile est représenté en bleu.

On s'intéresse particulièrement aux topics en sortie d'Hector SLAM. `/slam_out_pose` donne la localisation du robot en terme de position et d'orientation, `/trajectory` définit la trajectoire parcourue par le robot et `/map` donne la représentation de son environnement. Ce dernier topic correspond à une grille d'occupation, à savoir une discréétisation de l'espace sous forme de grille où chaque case porte la probabilité qu'elle soit un obstacle. Dans la pratique la carte est de dimensions 2048×2048 avec une résolution de $0.05m$. Les résultats issus de l'approche probabiliste s'expriment selon trois valeurs : 0 pour une case vide, 1 pour une case occupée et -1 pour une case inconnue. Chaque nouvelle acquisition attestant l'occupation d'une case augmente sa probabilité, et inversement lorsqu'une acquisition indique que la case est libre. Lorsque la probabilité portée par une case dépasse un seuil interne à Hector SLAM, la case sera considérée comme occupée et passera à la valeur 1⁶

L'intégration d'Hector SLAM au projet s'est déroulée en plusieurs temps. Nous avons d'abord pu prendre l'outil en main avant l'achat du LIDAR grâce à l'utilisation de données de test contenues par des bagfile. Durant cette étape, les résultats pouvaient être visualisés grâce à RViz, un outil graphique inclu dans ROS⁷. La figure 2.9 illustre le résultat obtenu pour un jeu de tests issu de la RobotCup German Open 2011. Les zones non explorées y figurent en gris foncé, les zones explorées mais vides en gris clair et les zones occupées en noir. Le robot est représenté par trois composantes orthogonales de couleur bleue, verte et rouge et la trajectoire parcourue est une courbe rouge.

Dans un deuxième temps, il a fallu adapter Hector SLAM pour recevoir en entrée les

6. Ce seuil est fixé à 0.5

7. Pour la distribution Kinetic de ROS, RViz est présent dans les paquets d'installation dits "Desktop"[17].

données du LIDAR, ce qui revient principalement à créer des fichiers de lancement (*launchfile*) adaptés aux périphériques d'acquisition. Dans notre cas nous avons créé deux fichiers distincts : l'un pour le mode d'exécution standard et l'autre pour un mode "debug". Nous avons également mis en place un système de définition des transformations spatiales entre les dispositifs matériels. Cette démarche sera présentée dans la partie 2.3.1.

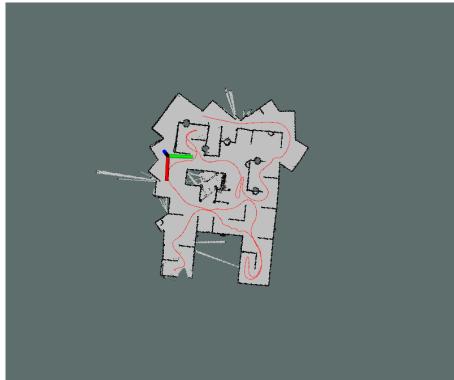


FIGURE 2.9 – Visualisation avec RViz d'un jeu de données pré-enregistrées

2.3 Réalisations et architecture logicielles

2.3.1 Le réseau ROS complet

Hector SLAM a été intégré à un réseau ROS plus large prenant en compte les spécificités de notre application. Il était entendu que l'IHM ne serait pas inclue dans ce réseau et serait développée avec Qt Creator (voir partie 2.3.2).

La définition du réseau doit alors s'inscrire comme une solution aux problématiques suivantes :

- le contrôle du LIDAR ainsi que la lecture et l'émission des nuages de points calculés
- l'intégration d'Hector SLAM
- la communication des résultats de cartographie et de localisation vers l'IHM

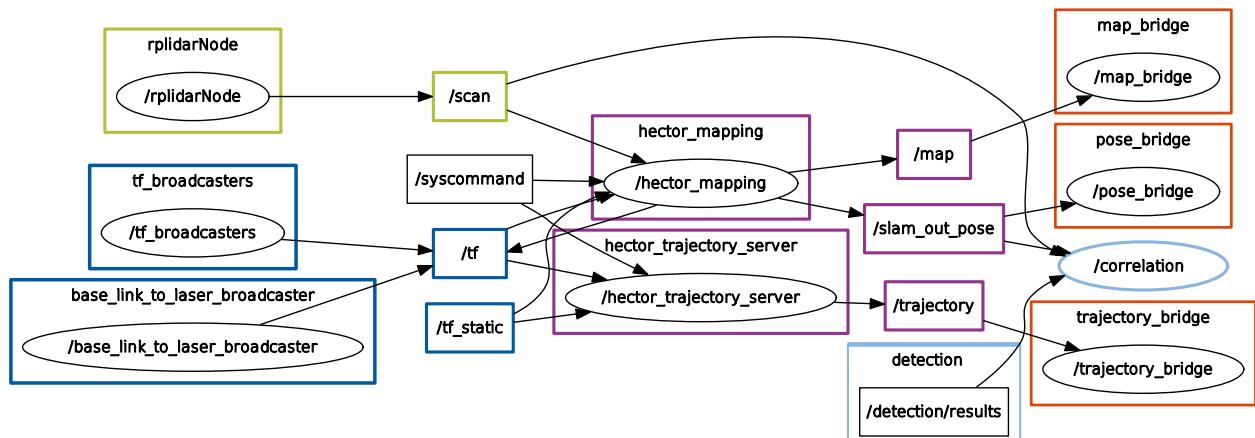


FIGURE 2.10 – Réseau ROS intégré au système

Le premier point, représenté en vert sur la figure 2.10, est assuré par un nœud embarqué sur le Raspberry Pi régissant la communication avec le LIDAR. Il s'interface avec plusieurs fichiers d'en-tête qui constituent la SDK du LIDAR. Cet élément du réseau ROS est fourni par SLAMTEC dans son kit de développement. Sa mise en œuvre a demandé de comprendre les principales méthodes de la SDK et la portée fonctionnelle de chaque fichier d'en-tête. Par ailleurs, outre l'installation de ROS sur Raspberry, la mise en place du matériel a nécessité la conversion des données UART vers USB grâce à un chipset fourni par le constructeur, l'ouverture du port série dédié au périphérique et la définition de règles communues à Alban Chazot et moi régissant les accès distants au Raspberry. À cet effet, la connexion au Raspberry Pi depuis le poste de travail passe par l'accès à un shell sécurisé (SSH), que nous avons voulu non interactif. L'utilisateur amené à se connecter au nano-ordinateur sous le nom d'hôte par défaut « PI » doit passer par une authentification par clés RSA. L'authentification par mot de passe étant désactivée, nous exécutons automatiquement cette procédure ainsi qu'un script en vue de restreindre la surface d'attaque potentielle. Le fichier `/home/pi/.ssh/authorized_keys` permet de mettre en œuvre cette politique :

```
1 # execute ssh_cmd_wrapper.sh with SSH_ORIGINAL_COMMAND argument,
  which must be locally existing ROS node name
2 command="/home/pi/scripts/ssh_cmd_wrapper.sh $SSH_ORIGINAL_COMMAND"
  ssh-rsa <RSA public key> <user at machine>
```

En sortie, ce nœud publie le topic `/scan`, correspondant au format de messages `sensor_msgs/LaserScan` précédemment évoqué (voir Annexe 1).

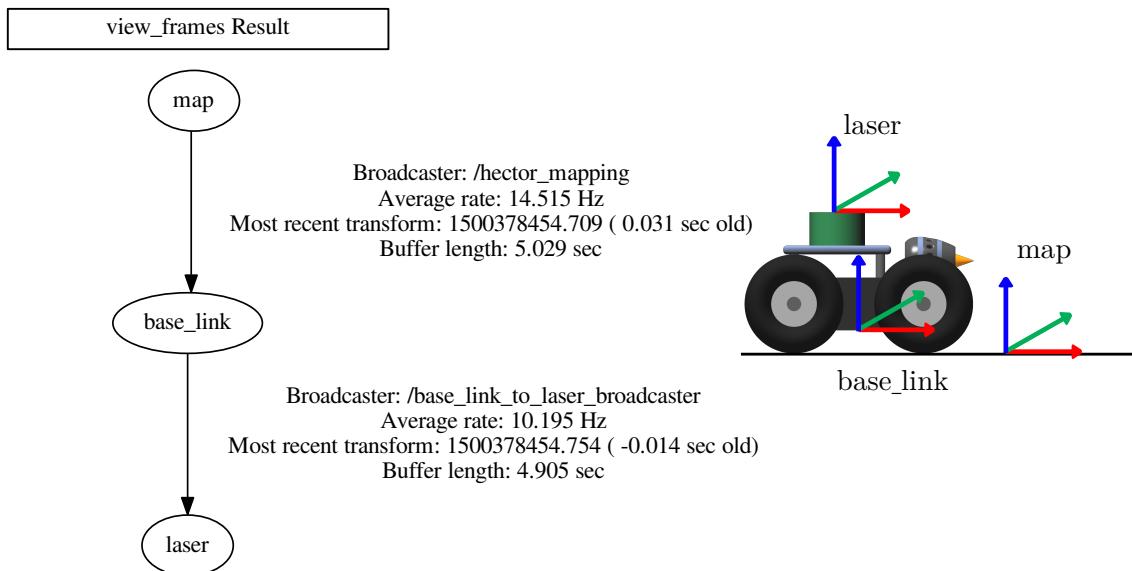


FIGURE 2.11 – Systèmes de coordonnées et transformations au sein du système

L'intégration d'Hector SLAM demande la définition de plusieurs systèmes de coordonnées correspondant aux divers périphériques d'intérêt. Les transformations spatiales entre ces systèmes étant également requises par Hector SLAM, elles sont publiées au travers d'un package appelé tf[18][19]. Les nœuds et topics impliqués dans ce processus sont représentés en bleu foncé sur la figure 2.10.

tf distingue deux types de transformations : les transformations statiques et les transformations dynamiques. Dans le premier cas, il suffit de définir l'identifiant d'un repère de référence (`frame_id`) et d'un repère fils (`child_frame_id`) ainsi que la transformation que subit ce dernier, en s'assurant qu'elle ne sera jamais amenée à

changer. Dans le second cas, il faut généralement passer par la création d'un noeud publant la valeur de ce décallage régulièrement durant l'exécution. Pour mieux saisir les enjeux soulevés par l'intégration d'Hector SLAM, la figure 2.11 donne les systèmes de coordonnées définis, ainsi que les nœuds implémentés (les *broadcasters*) responsables de la publication des transformations.

Le système *map* est un repère fixe, ancré au sol, qui sert de référence au positionnement du robot sur le long terme[20]. *base_link* est un repère mobile par rapport à *map* attaché sous la plateforme mobile en son centre. Une première transformation entre *map* et *base_link* est requise par Hector SLAM. Elle est calculée par le nœud */tf_broadcasters* (voir figure 2.10) à partir des relevés odométriques fournis par le robot. Une fois intégrée par Hector SLAM, cette transformation est corrigée puis ré-émise par */hector_mapping* : c'est le topic */tf* de la figure 2.10. Enfin, le nœud */base_link_to_laser_broadcaster* publie régulièrement la transformation entre *base_link* et *laser*, c'est-à-dire le différentiel de position entre la base du robot et le LIDAR. Celui-ci n'étant jamais amené à changer nous sommes dans le cas d'une transformation statique définie de la manière suivante dans le launchfile d'Hector SLAM en mode d'exécution standard⁸ :

```

1 <!-- Laser is set 25cm above robot base. Node arguments must be :
2 dx dy dz yaw pitch roll frame_id child_frame_id
   publishing_frequency_in_ms
3 -->
4
5 <node pkg="tf" type="static_transform_publisher"
6   name="base_link_to_laser_broadcaster"
7   args="0 0 0.25 0 0 0 base_link laser 100" />
```

Nous obtenons ainsi un moteur de SLAM fonctionnel à partir duquel nous devons construire une IHM. Pour ce faire, nous avons défini trois nœuds appelés *bridges* représentés en orange sur la figure 2.10. Chacun d'entre-eux s'abonne à l'un des topics en sortie d'Hector SLAM. Ensuite, les données utiles sont formatées par appel à une classe statique nommée *Serializer* et communiquées à l'IHM par le biais d'un client TCP implémenté avec l'API socket. La sérialisation répond à un schéma générique composé d'une en-tête (*header*) et d'une charge utile (*payload*). Il est représenté sur la figure 2.12.



FIGURE 2.12 – Définition du paquet générique de transmission de données à l'IHM

Les éléments contenus par le header sont *tID*, un identifiant donnant le type de la donnée transmise, *pSize*, la taille du payload et un *padding* à 0 qui vise à ce que le *header* atteigne 20 octets. La formation du *payload* dépend quant à elle des données à transmettre. La principale difficulté réside en la transmission de la grille d'occupation contenue par le topic */map*. Celle-ci contenant plus de 4 000 000 valeurs entières, nous nous sommes focalisés sur l'émission de mises à jour locales à chaque acquisition, plutôt que globales⁹. Ainsi, le *payload* se construit de la manière

8. En mode “debug” cette transformation est non avenue puisqu’elle est déjà comprise dans le *bagfile* responsable du rejet des données

9. Il en va de même pour la trajectoire où, dans un même souci d’optimisation, nous effectuons un différentiel entre les valeurs précédemment acquises et la nouvelle acquisition

suivante afin de désencombrer la bande passante et réduire la taille de la structure de données qui l'encapsule.

```

1 // Fill a buffer with occupancy grid local updates
2 int Serializer::serializeMap( const nav_msgs::OccupancyGrid& map,
3                               const nav_msgs::OccupancyGrid&
4                               old_map,
5                               std::string *buffer ) {
6
7     std::string header, payload;
8
9     // Check data consistency
10    if( map.info.width != old_map.info.width ||
11        map.info.height != old_map.info.height ) {
12        std::cout << "Serialization error : Map and old_map with
13                  different dimensions" << std::endl;
14        return -1;
15    }
16
17    // Fill payload with updated cases' value and index
18    for( int i = 0; i < map.info.width * map.info.height; ++i ) {
19        if( map.data[i] != old_map.data[i] )
20            payload.append( std::to_string(i) + ","
21                            + std::to_string(map.data[i]) + "," );
22    }
23    payload.append( "EOP" );
24
25    // Fill header, return error if its size exceeds 20 bytes
26    header.append( MAP_ID );
27    header.append( "," + std::to_string(payload.length()) + "," );
28    if( Serializer::addZeroPadding(&header) < 0 ) return -1;
29
30    // Fill buffer with header and payload
31    buffer->append( header );
32    buffer->append( payload );
33    return 0;
34 }
```

Enfin, la figure 2.10 fait également état du nœud `/correlation` abonné aux résultats de détection d'objets d'intérêt et au topic `/scan`. Il est responsable du croisement de ces données afin de déterminer, quand cela est possible, les coordonnées d'un objet détecté à partir du flux vidéo. La technique employée ici consiste à trouver pour chaque résultat de détection, la position du robot à l'instant de la détection ainsi que les faisceaux du LIDAR susceptibles d'avoir atteint l'objet. Lorsque ces derniers existent¹⁰ nous devons transformer la distance et l'angle associés dans le système de coordonnées de la carte. L'algorithme suivant donne les principes implémentés à cet effet.

10. Le plan parcouru par les faisceaux du LIDAR doit être compris entre les extrémités haute et basse de l'objet.

Algorithme 1 : Algorithme de calcul des résultats de corrélation

Entrées : R les derniers résultats de détection reçus sous la forme :

dimensions « planes » de l'objet (w, h) et ses coordonnées sphériques (ϕ, θ)

t un entier non signé, estampille de R

Données : S l'ensemble des scans n'ayant jamais été corrélés

P l'ensemble de positions du robot n'ayant jamais été corrélées

$scan$ un timestamp et un ensemble $coord$ de paires de coordonnées polaires (θ, r)

$pose$ un timestamp, une position $p = (x, y, z)$ et une orientation

$q = (pitch, roll, yaw)$

$angle$ l'angle auquel se trouve l'objet dans le système de coord. du scan

$matched$ couple (θ, r) caractérisant la position de l'objet détecté

début

si $S = \emptyset \vee P = \emptyset$ **alors**

retourner -1

sinon

 /* find temporally nearest scan and pose with respect to t
 */

$scan \leftarrow findNearestScan(t)$

$pose \leftarrow findNearestPose(t)$

si $scan = 0 \vee pose = 0$ **alors**

retourner -1

sinon

pour $r \in R$ **faire**

 /* check if current object r crosses LIDAR laser */

si $(r.\theta - \frac{r.h}{2} < \frac{\pi}{2}) \wedge (r.\theta + \frac{r.h}{2} > \frac{\pi}{2})$ **alors**

$angle \leftarrow r.\phi$

pour $s \in scan.coord$ **faire**

si $s.\theta > angle$ **alors**

break

$matched \leftarrow s$

 /* add object position and dimension in result

 structure */

si $matched.r \neq \infty$ **alors**

$r.x \leftarrow pose.p.x + matched.r \times \cos(matched.\theta + pose.yaw)$

$r.y \leftarrow pose.p.y + matched.r \times \sin(matched.\theta + pose.yaw)$

$r.z \leftarrow LIDAR_LENGTH + matched.r \times \tan(\frac{\pi}{2} - r.\theta)$

$r.w \leftarrow 0.5 \times 2 \times matched.r \times \tan(\frac{r.w}{2})$

$r.h \leftarrow 0.5 \times 2 \times matched.r \times \tan(\frac{r.h}{2})$

retourner 1

2.3.2 Interface Homme-Machine avec Qt Creator

L'Interface Homme-Machine de l'application a été implémentée en C++ avec l'API Qt, facilitant notamment la réalisation de l'interface graphique au moyen de l'environnement de développement Qt Creator. Dans l'API Qt, les éléments graphiques sont appelés *widgets* et dérivent de classes définies dans les bibliothèques internes, telles que `QWidget`, `QOpenGLWidget` ou encore `QDialog` pour ne citer que celles qui ont été utiles au projet. Qt fournit également un mécanisme de communication inter-classes thread-safe et type-safe, par le biais d'éléments appelés signaux et slots¹¹. Ceux-ci ont la particularité d'occuper la démarche de liaison entre les parties communicantes et ne nécessitent pas de classe dédiée à cet effet. Ils permettent de mettre en place simplement des connexions *one-to-many*, *many-to-one* ou encore *many-to-many*. Un signal est une signature de fonction membre d'une classe, qui pourra être émis par ses instances. Les slots sont quant à eux des fonctions membres désignées par la macro `Q_SLOTS`. La mise en œuvre de signaux et de slots est illustrée dans l'exemple suivant, tiré du code source du projet.

```

1 // sensorDataAvailable signal connection to setWifibotInfo slot
2 QObject::connect( wc_, &WifibotClient::sensorDataAvailable,
3                   w_, &MainWindow::setWifibotInfo );
4
5 // signal emission in WifibotClient class
6 emit sensorDataAvailable(robotSensors);
7
8 // data computing in MainWindow callback
9 void MainWindow::setWifibotInfo(SensorData sd)
10 {
11     // dispatch info to sensor widgets through other signals and
12     // slots mechanisms
13     emit setIRLeftValue(sd.IRLeft);
14     emit setIRRRightValue(sd.IRRight);
15     emit setBatteryValue(sd.batVoltage);
16     emit setOdomLeftValue(sd.odometryLeft);
17     emit setOdomRightValue(sd.odometryRight);
18     emit setSpeedLeftValue(sd.speedFrontLeft);
19     emit setSpeedRightValue(sd.speedFrontRight);
20 }
```

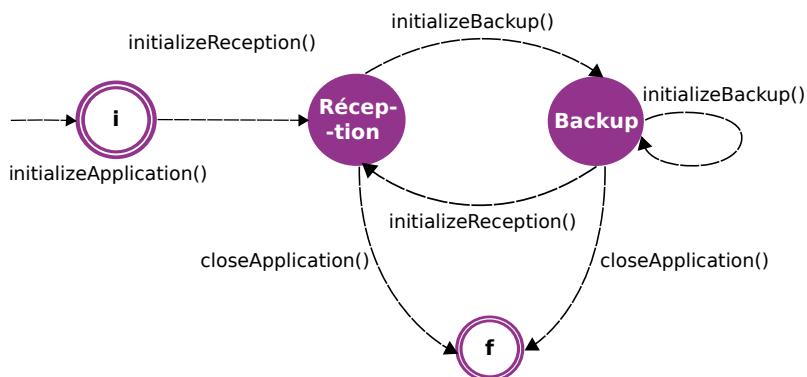


FIGURE 2.13 – Machine à état de l'application Qt

D'un point de vue formel, le fonctionnement de l'application est régi par une machine à états relativement simple, représentée figure 2.13. Ces états facilitent la gestion des

11. Ce mécanisme peut être mis en parallèle avec les callbacks en C ou C++ ou encore les signaux fournis par l'API boost en C++

instances utiles à n'importe quel instant de l'exécution en dissociant clairement ce qui est du ressort de chacun des modes de fonctionnement définis. Le mode *Réception* satisfait l'exigence principale du projet, à savoir présenter en temps-réel les résultats de SLAM tout en assurant le contrôle du robot. Le mode *Backup* répond à une fonctionnalité additionnelle offrant la possibilité à l'utilisateur de rejouer des données précédemment enregistrées. Le passage d'un état à l'autre est symbolisé par des flèches portant le nom de la méthode invoquée à cet effet. Ces méthodes visent à libérer la mémoire relative aux ressources dynamiques du mode courant, à instancier correctement les ressources graphiques ou de contrôle propres au mode requis et finalement, à changer une variable représentant l'état du système. On note que les fonctions de transition sont des slots atteints par des interactions spécifiques de l'utilisateur sur la fenêtre graphique.

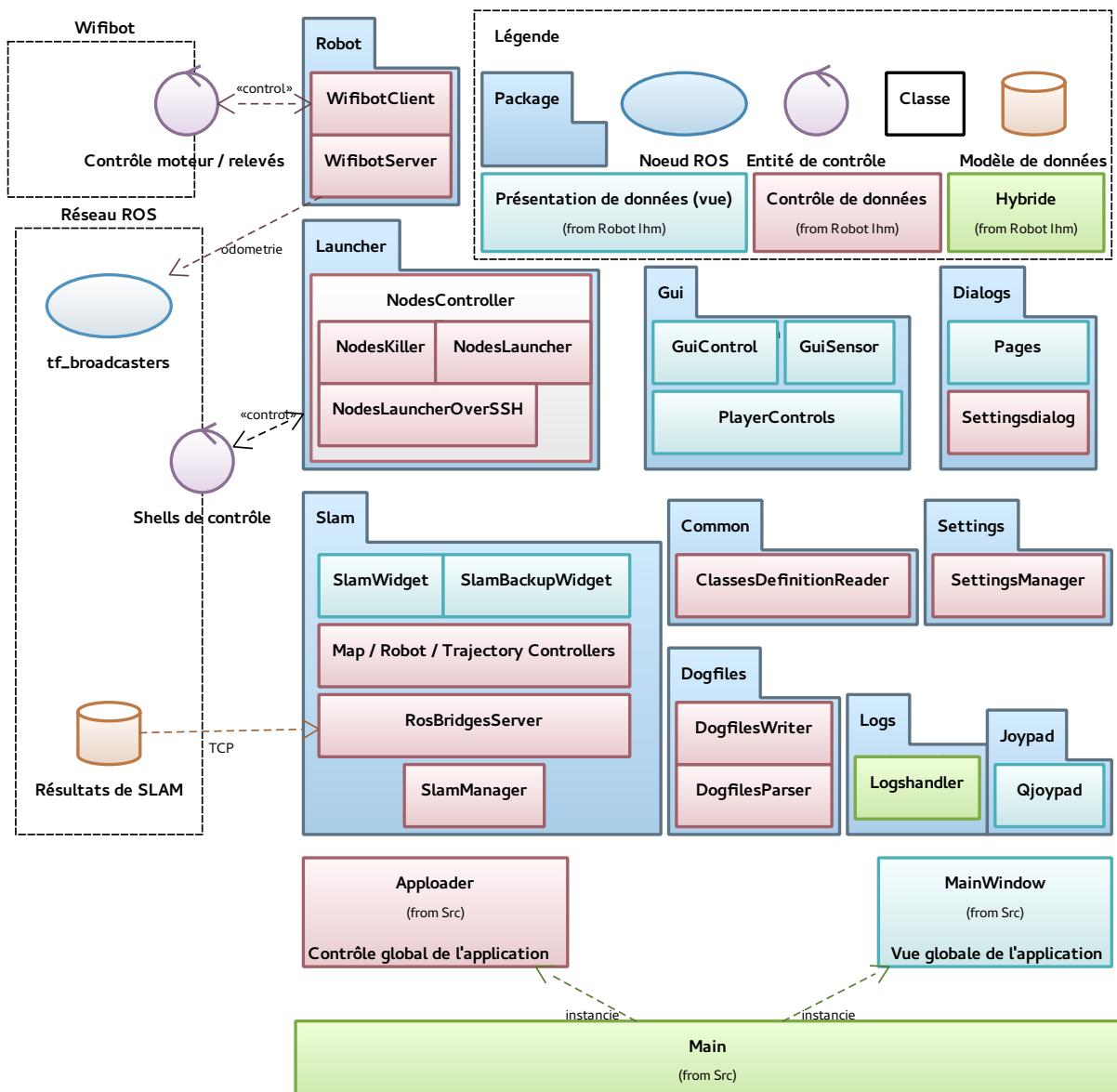


FIGURE 2.14 – Modélisation logicielle de l'Interface Homme-Machine

La figure 2.14 expose quant à elle les classes et packages développés avec Qt pour répondre aux exigences fonctionnelles et d'interface précédemment formulées. Sont également représentés les moyens de communications mis en place entre l'IHM et les

autres éléments constituant le projet : le réseau ROS et la couche d’exploitation du Wifibot.

Le point d’entrée de l’application est la classe `main.cpp` qui instancie la fenêtre principale de l’application `MainWindow`[22] et une classe `Apploader` spécifique à notre architecture. Cette dernière maintient et met à jour la machine à états présentée en 2.13 gérant ainsi le cycle de vie de toutes les instances.

Le package *SLAM* traite les données en sortie du réseau ROS.

La classe `ROSBridgesServer` instancie un serveur TCP consacré à la communication avec les noeuds de *bridge* décrits dans la partie 2.3.1. Ces données sont ensuite transmises à des contrôleurs dédiés à chaque type de données (carte, position ou trajectoire) qui assurent le parsing des paquets reçus puis émettent des signaux aux *widgets* de rendu. Dans notre cas, deux *widgets* sont susceptibles de rendre ces données : `SlamWidget` ou `SlamBackupWidget` respectivement associés au mode *Réception* ou *Backup*. Ces *widgets* héritent de la classe Qt `QOpenGLWidget` fournissant les fonctionnalités de rendu graphique d’OpenGL dans un contexte Qt. La figure 2.15 donne un aperçu de l’IHM en mode *Réception* afin de présenter les différents éléments de l’application. La fidélité des résultats sera quant à elle discutée dans la partie 3.

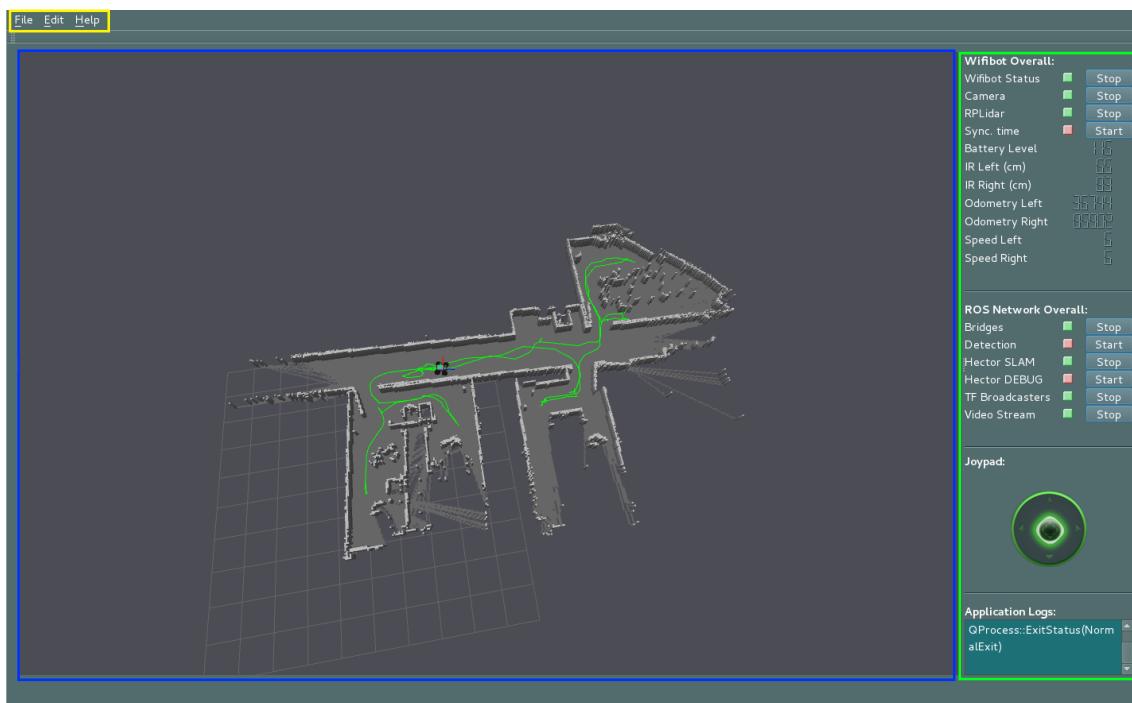


FIGURE 2.15 – Rendu visuel de l’IHM en mode Réception

Nous pouvons visualiser la fenêtre principale (`MainWindow`) complétée d’instances de *widgets* responsables du rendu d’éléments d’intérêt :

- **un widget de rendu OpenGL**, en bleu, fournit une représentation de la carte, du robot par le biais d’un modèle 3D et de sa trajectoire (position et orientation en tout temps)
- **un panneau de contrôle**, en vert, est découpé en quatre sections : le Wifibot, le réseau ROS, un joystic virtuel et une console de logs
- **une barre de menu**, en jaune, permet la sauvegarde de jeux d’acquisition, le chargement d’une sauvegarde pour la rejouer, l’édition de paramètres de l’application ou l’affichage du manuel utilisateur

La figure 2.16 expose le rendu global en mode *Backup*. Celui-ci se distingue particulièrement du mode d’acquisition par la présence –au sein du panneau de contrôle– d’un *player* offrant les fonctionnalités d’un lecteur multimédia classique.

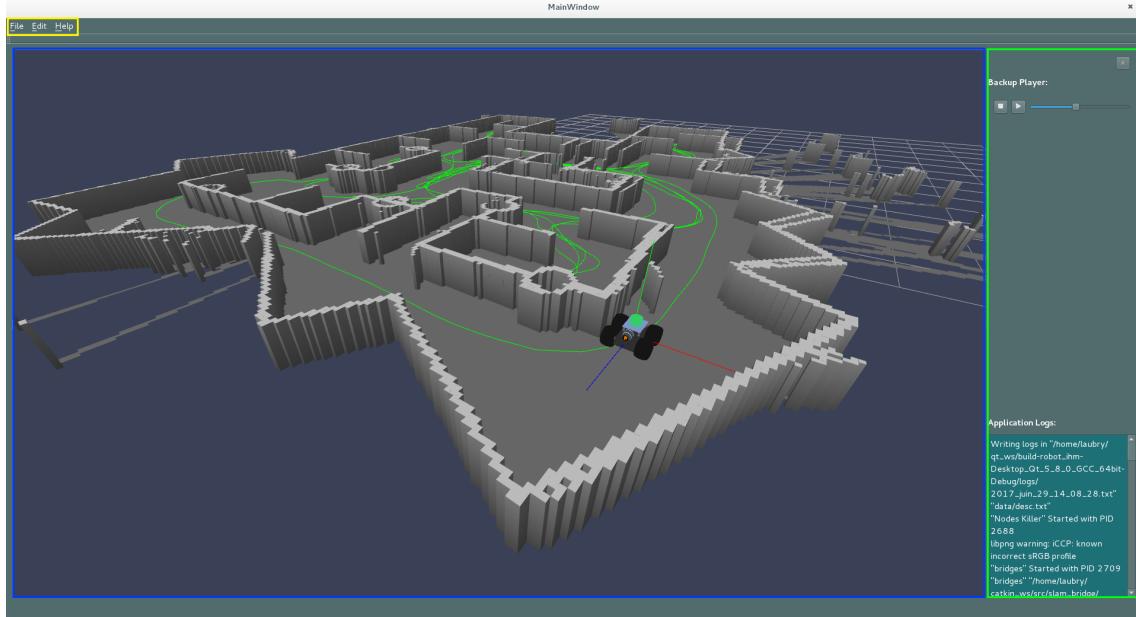


FIGURE 2.16 – Rendu visuel de l’IHM en mode rejeu

À cet effet, un format d’enregistrement des données propre à l’application a été défini en XML. Ce format, appelé *DOG*¹² est géré par les classes du package **DogFiles** : **DogFilesWriter** pour l’écriture et **DogFilesParser** pour la lecture. Des techniques de compressions ont été appliquées pour les données de taille critique devant être sauvegardées. Ainsi, les données décrivant la carte sont traitées par un algorithme d’encodage par répétition (Run Length Encoding) tandis que toutes les composantes des vecteurs de trajectoire du robot sont gérées par la méthode suivante :

```

1 // Multiply entry by 100, truncate it and returns its hexadecimal
   value. Negative number are concatenated with '-' char and
   encoded like positive values.
2 QString SlamWidget::floatToHex(float v)
3 {
4     return (int)(v * 100) >= 0.0f
5         ? QString::number( (int)(v * 100), 16 )
6         : QString('-' + QString::number( (int)-(v * 100), 16 ))
7 }
```

Un algorithme similaire a été appliqué pour l’ensemble des orientations du robot. Ces techniques ont permis de réduire significativement les données à sauvegarder à l’aide d’une compression sans perte dans un premier cas et une perte de l’ordre du centième de case (soit 0.0005 m) dans un second cas.

2.3.3 Éléments de modularité mis en place

La réalisation de l’application a d’emblée été associée à un objectif de modularité afin d’appréhender au mieux la perspective de stages futurs et / ou d’une possible

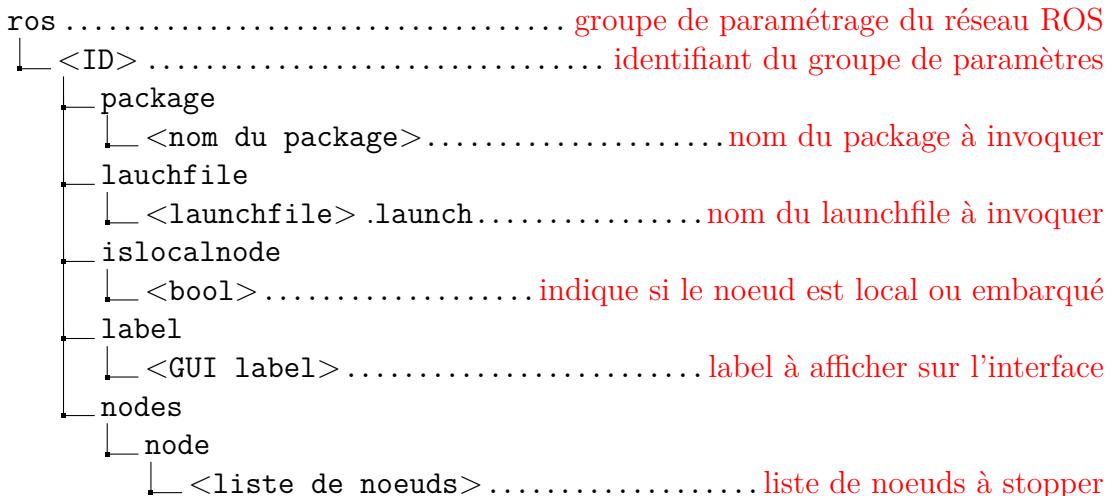
12. Pour Detection and Occupancy Grid

adaptation industrielle du projet. L'utilisation de ROS allant dans ce sens, il paraissait intéressant d'appliquer ce principe à l'IHM, tant dans l'architecture du logiciel qu'au travers des fonctionnalités proposées à l'utilisateur final.

Le premier point se retrouve principalement dans l'implémentation de l'**Apploader** qui gère le cycle de vie des instances d'une manière communément compréhensible : création, mise-à-jour puis destruction. D'autre part, l'interface graphique et les éléments de communication avec les nœuds peuvent eux-mêmes être modifiés à convenance par le biais de l'IHM. Nous exposons ici les principes qui sous-tendent cette démarche, nous amenant à balayer les points suivants :

- le paramétrage modulaire du réseau ROS
- les éléments graphiques attestant l'état de chaque nœud
- le contrôle du réseau ROS depuis l'IHM

Qt offre un système de paramétrage des applications qui s'appuie sur des fichiers au format texte et sur une classe **QSettings** indépendante de la plateforme utilisée¹³. Cette dernière permet l'écriture de paramètres selon le paradigme clé / valeur et hiérarchisés sous forme de groupe. Dans notre IHM, une classe statique **SettingsManager** est utilisée pour écrire de tels paramètres, soit à l'initialisation de l'application soit lors de la configuration de celle-ci par l'utilisateur. Nous avons défini les groupes suivants relativement au réseau ROS :



L'utilisateur peut mettre à jour ou supprimer les valeurs par défaut et créer de nouvelles entrées directement depuis l'application sous **Edit > Settings > ROS** (voir figure 2.17).

Ces paramètres sont ensuite parsés, permettant la présentation et l'utilisation de chaque module ROS de manière uniforme. D'un point de vue graphique, la classe **GuiControl** est responsable de la création dynamique des éléments de l'interface représentant les modules ROS, à partir de la clé de configuration **ros/<ID>/label/<GUI label>** tels que représentés figure 2.18.

Parallèlement, chaque nouvel identifiant sous **ros/<ID>** entraîne l'instanciation de la classe **NodesLauncher** ou **NodesLauncherOverSSH** –en fonction du caractère local ou non du package– et de la classe **NodesKiller**. Ces objets exécutent un shell Linux (**/bin/bash**) au sein d'un membre Qt de type **QProcess** permettant l'exécution du launchfile afférent lorsque l'utilisateur presse le bouton “Start” du module. Ils sont

¹³. En l'occurrence, sur un système Unix, de tels fichiers seront stockés au format **.ini** sous **~/.config/<filename>.ini**

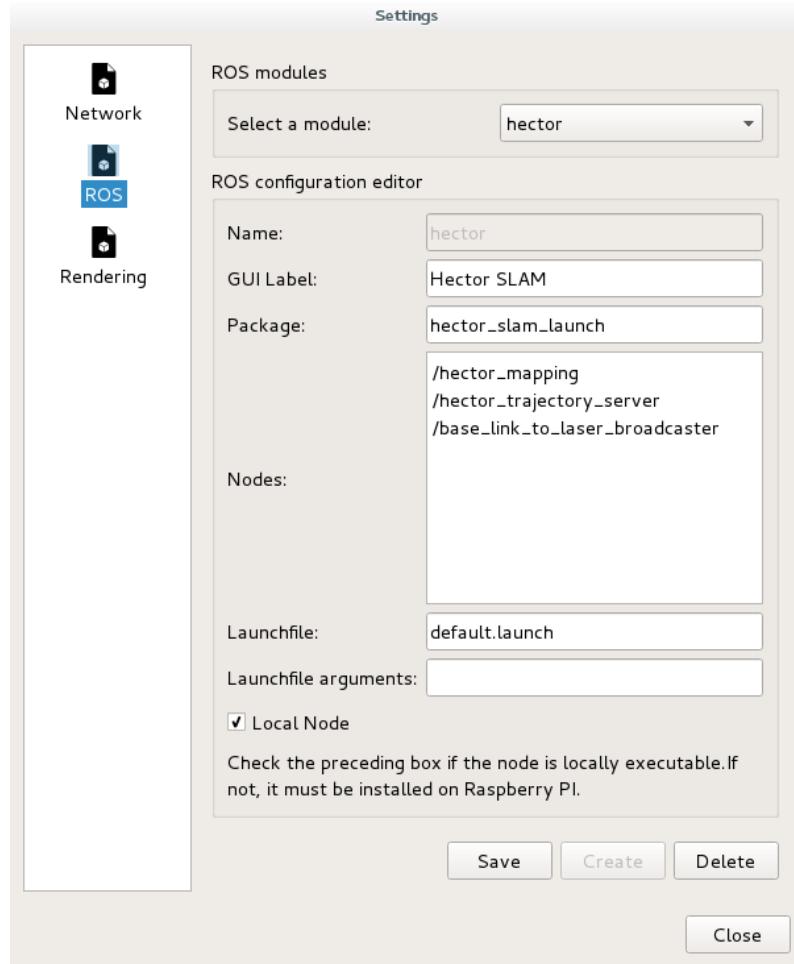


FIGURE 2.17 – Boîte de dialogue de configuration du réseau ROS depuis l'IHM

aussi responsables de la redirection des sorties du processus vers les sorties dédiées à la journalisation des évènements de l'application¹⁴. La classe `NodesKiller` va quant à elle émettre des commandes système vers le ROS Master afin de tuer les nœuds d'un package lorsque l'utilisateur souhaitera mettre fin à leur exécution.

Pour résumer, un utilisateur voulant ajouter un nouveau package ROS à l'application devra le compiler de manière classique dans son workspace Catkin, éditer un *launchfile* permettant d'exécuter le ou les nœuds afférents puis simplement renseigner les informations requises depuis l'IHM. Sous la section `Edit > Settings > Network`

14. La classe `LogsHandler` est utilisée pour journaliser les messages d'une part dans des fichiers datés du répertoire d'exécution de l'application et d'autre part, dans la console du panneau de contrôle

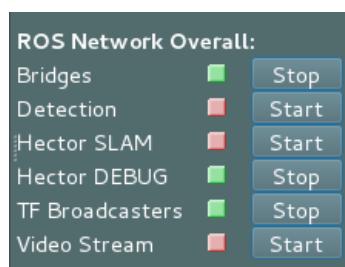


FIGURE 2.18 – Contrôle du réseau ROS créé dynamiquement sur l'interface graphique

l'utilisateur pourra également configurer les noms d'hôtes et adresses IP distants, adaptant ainsi le système à d'autres modèles ou instances de robots. Si l'opérateur venait à utiliser une plateforme dotée de capteurs communiquant leurs résultats à un package ROS dédié, le contrôle de ces nouveaux exécutables pourrait être couvert par l'application en quelques manipulations seulement.

Chapitre 3

Bilan et perspectives

3.1 Organisation du travail

3.1.1 Application du référentiel qualité interne

Nous exposons ici les éléments fournis par SII ayant guidé de manière significative la conduite du projet. Ceux-ci seront plus ou moins détaillés en fonction du temps consacré à leur mise en place.

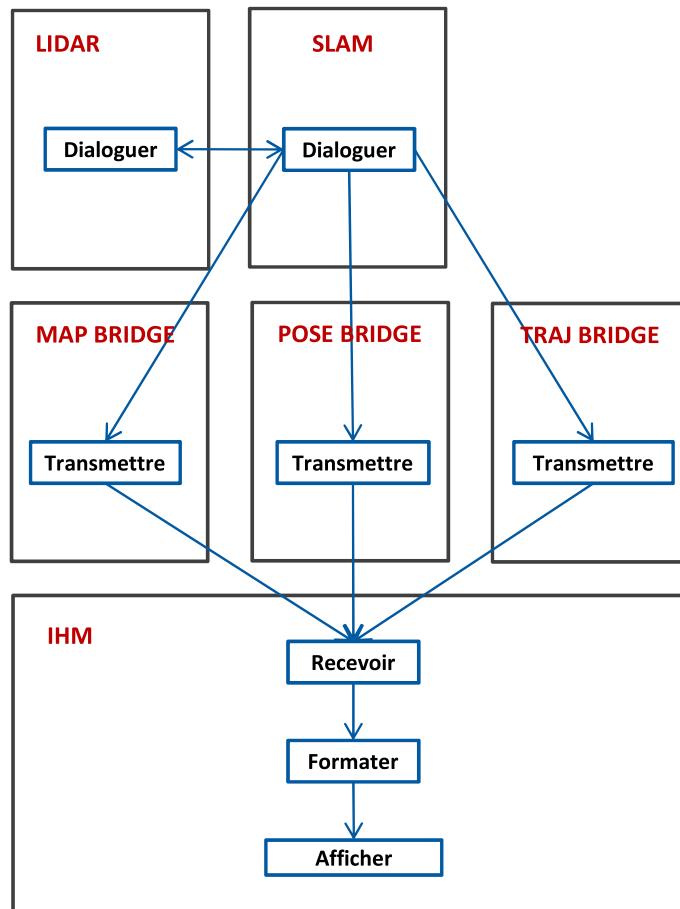


FIGURE 3.1 – Modèle de définition des modules logiciels et relations entre leurs fonctions

Les premiers éléments organisationnels qui ont été formalisés s'incarnent par deux documents de spécification appelés STBL et DCL. Le premier vise, comme son nom l'indique, à exprimer le plus objectivement possible le besoin qui justifie la réalisation logicielle, notamment au moyen d'exigences représentées figure 2.2. Cette énumération haut-niveau permet d'appréhender simplement et rapidement le périmètre du logiciel ciblé. Cependant, la réalisation de ce document en bonne et dûe forme nécessite une liste exhaustive d'exigences fonctionnelles vues comme des ensembles de fonctions qu'il convient de décrire en terme de rôle, d'entrées, de sorties et de traitements. Dans le cadre de ce stage, nous définissons des fonctions comprises dans des modules distincts et étant reliés comme présenté sur la figure 3.1. Puis, pour chacune des fonctions, on va définir des sous-fonctions, idéalement jusqu'à un niveau de granularité maximal où tous les types de données sont énumérés. Par exemple, on établit que la fonction *Transmettre* du module **Map_bridge** regroupe les trois sous-fonctions *Recevoir un message*, *Formater un message* et *Émettre un message*. Afin d'illustrer cette démarche, nous donnons ici pour exemple la spécification de la fonction *Transmettre* : *Formater un message*.

1. Rôles

Cette fonction est activée à l'arrivée d'une demande de traitement de message sur la liaison SLAM \iff MAP_BRIDGE. Elle intervient après que la fonction de réception ait attesté de la consistance du message reçu. Elle permet d'extraire la charge utile du message reçu et d'en réduire significativement le volume avant envoi. Les données en sorties sont sérialisées. Cette extraction s'effectue en comparant OCCUPANCY_GRID avec une copie locale de la dernière carte reçue OLD_OCCUPANCY_GRID.

2. Entrées

Désignation	Type de données
Message OCCUPANCY_GRID	OccupancyGrid

FIGURE 3.2 – Entrées de la fonction *Transmettre* : *Formater un message*

3. Sorties

Désignation	Type de données
Message MAP	OccupancyGridUpdate
MessageINI_MAP	IniOccupancyGrid

FIGURE 3.3 – Sorties de la fonction *Transmettre* : *Formater un message*

Notons que la spécification des types de données d'entrées-sorties est donné dans un document externe. Relativement à l'exemple précédent, le type d'OCCUPANCY_GRID est décrit en Annexe 3.

4. Traitements

NB : Les traitements sont identifiés de manière unique au sein d'un document, ils satisfont également des règles permettant d'être traités automatiquement par des logiciels de suivi de projet. Nous donnons ici un exemple succinct permettant d'en saisir le sens. Dans la pratique une sous-fonction donne lieu à plusieurs traitements.

[REQ_STBL_MAP_BRIDGE_FORM_1]

Si OLD_OCCUPANCY_GRID existe :

On compare le champ “data” de OCCUPANCY_GRID et de OLD_OCCUPANCY_GRID.

On crée la structure MAP à partir des valeurs de “data” qui diffèrent.

MAP est une chaîne de caractères dont les valeurs sont séparées par des ",".

[FIN_REQ]

Le Document de Conception Logicielle est quant à lui intervenu plus tard dans l'avancée du stage. Il consiste à formaliser les éléments conceptuels, en termes d'architecture physique et logicielle du projet. Ces briques ayant été largement explicitées au long de ce rapport nous n'étayerons pas d'avantage ce point.

Un planning présentant une granularité hebdomadaire a été effectué au mois de mars, ce document est présenté en Annexe 4. Cette réalisation vise d'une part à définir et ordonner les tâches à réaliser et, d'autre part, à estimer l'impact temporel de chacune d'entre-elles. Ce document a été réalisé dans une optique d'organisation personnelle mais a également constitué un outil d'évaluation et de communication avec M. Daumand qui a été en grande partie affecté en missions hors de l'agence. Il s'agissait donc pour nous de fixer les jalons clés, les *dead-lines* et les versions du logiciel à produire afin qu'il suive de manière pragmatique l'avancée du travail.

3.1.2 Vers une conduite AGILE adaptée

Au référentiel interne de qualité du logiciel se sont ajoutées certaines bonnes pratiques issues de la formation en Architecture et Sécurité du Logiciel prodiguée à l'INSA Centre Val-de-Loire. En particulier, nous avons utilisé le gestionnaire de versions Git au travers du système de gestion de dépôts (forge) GitLab auquel nous avons appliqué une stratégie de création de branches dite « Git Flow ». Ce modèle d'utilisation de

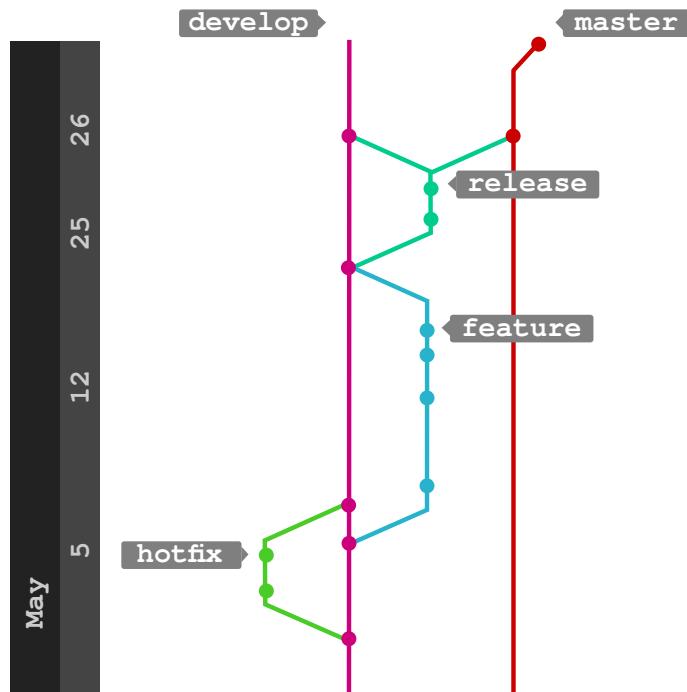


FIGURE 3.4 – Politique de gestion de branches Git adoptée

Git vise à minimiser les conflits et les régressions, accentuer la visibilité des tâches en cours ou terminées et à scinder clairement les phases de développement du logiciel. À cet effet nous avons adopté la démarche illustrée sur la figure 3.4 qui se lit de bas en haut et où les commits sont représentés par des points. Cela consiste à travailler sur une branche de développement (`develop`), à partir de laquelle nous créons une branche par fonctionnalité (`feature`), éventuellement une branche `hotfix` pour une correction de bug et, enfin, une branche `release` qui va aboutir sur la dernière version stable du logiciel. Cette branche est ensuite répercutée sur `develop` et `master`, dans le premier cas pour continuer les actions de développement et dans le second, afin de permettre la récupération des sources ou leur déploiement.

Aussi, l'**Uploader** a constitué une refonte majeure de l'architecture du logiciel qui a été conceptualisée et développée en équipe. Afin de répartir les tâches que nécessitait son implémentation, nous avons pu expérimenter l'utilisation d'un Kanban recensant :

- les tâches à effectuer, par exemple *Enregister les données de cartographie dans le format DOG*
- la complexité relative de chacune d'entre-elles
- les dépendances de certaines tâches les unes par rapport aux autres

Définir la « complexité » d'une tâche revient à lui attribuer un score en se référant aux scores attribués pour les tâches précédentes. Les méthodes AGILES –notamment SCRUM– préconisent le recours à des échelles de quantification relatives plutôt qu'à des jours hommes ou d'autres échelles se voulant précises. Concrètement on peut utiliser les valeurs de la suite de Fibonacci qui suivent ce que l'on appelle la courbe d'incertitude, à savoir qu'au plus une valeur est élevée, au plus l'écart avec la valeur suivante sera grande. Nous avons estimé la complexité des unités de travail à réaliser selon une méthode également empruntée à SCRUM, appelée *planning poker*. Tous les participants disposent d'un jeu de carte qui, dans notre cas, comporte les nombres $\frac{1}{2}$, 1, 2, 3, 5, 8, 13, ∞ . L'effort de réalisation est ensuite estimé en même temps pour une tâche donnée, puis soumis à discussion dans le cas de désaccord. Cette pratique est à la fois rapide, ludique et présente l'intérêt de dénier d'emblée la quantification de l'influence mutuelle des participants.

Ce Kanban a ainsi été adopté dans une optique d'ordonnancer les tâches et, accessoirement, de les répartir entre les membres de l'équipe. Il a aussi permis d'attester visuellement de nos avancées respectives et du nombre de tâches en cours permettant un allègement ou une répartition de la charge si besoin. La consultation du nombre de tâches restantes a également joué dans nos décisions de poursuivre telle ou telle fonctionnalité au profit d'autres.

3.2 Résultats en vue d'un prolongement

3.2.1 Un module de SLAM en adéquation avec les attentes du projet

Nous nous intéressons dans un premier temps à la fidélité des résultats de SLAM et ensuite à une appréciation de l'ensemble du projet, incluant également les travaux d'Alban Chazot et bien sûr la vision de M. Daumand.

La figure 3.5 donne un aperçu de la précision des résultats de SLAM. Elle met en parallèle les résultats cartographiques du système avec un plan d'évacuation des locaux dans lesquels l'acquisition a pu être menée. Pour des raisons pratiques, toutes les pièces n'ont pu être scannées, puisque celles-ci hébergent des collaborateurs ou directeur d'agence menant leurs activités.

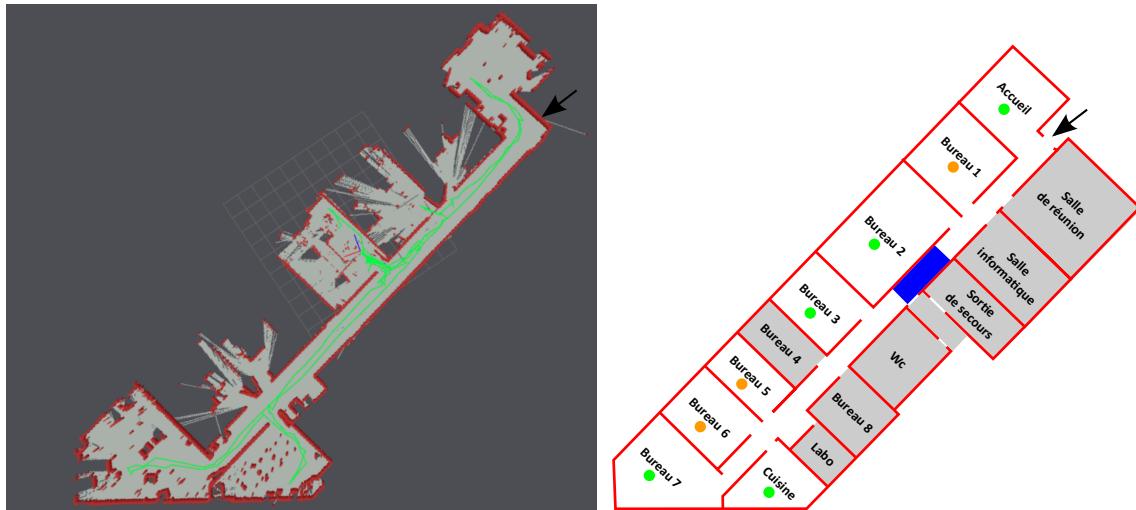


FIGURE 3.5 – À gauche cartographie des locaux avec le logiciel, à droite plan d'évacuation correspondant

Le plan des locaux est complété par des cercles de couleur verte pour les pièces qui ont été au moins à moitié explorées et de couleur orange pour les pièces visibles à moins de 50%. Enfin nous représentons sur fond gris les pièces qui n'ont pas été scannées du tout. Le couloir central a été parcouru dans toute sa longueur et la porte d'entrée est pointée par une flèche noire sur les deux représentations, afin d'éviter toute confusion quant à la façon de les interpréter.

On remarque empiriquement que les lieux cartographiés sont reconnaissables sans trop de difficultés, que leur agencement correspond à la réalité perçue et que les proportions des salles sont pour la plupart fidèles. On peut aussi noter la précision du rendu des obstacles fixes : portes, pieds de tables, de chaises ou du baby-foot situé au centre de l'accueil dont trois pieds sur quatre sont visibles et correctement situés. La géométrie des murs est également respectée, ce qui peut être attesté par les angles droits des pièces ou la régularité du couloir.

Cependant le résultat n'est pas sans défauts et certaines erreurs méritent d'être discutées. L'imperfection la plus flagrante se situe au niveau du bureau 2. Selon la cartographie celui-ci est aussi long que le bureau 3, tandis qu'il est, dans les faits, presque deux fois plus long. Cette déformation se retrouve en effet sur toutes les acquisitions qui se sont déroulées de la manière suivante :

- départ du bureau 3
- sortie du bureau 3 en direction du bureau 2
- déplacement dans un couloir de 8 mètres
- arrivée dans le bureau 2

Lorsque la plateforme longe le couloir, le moteur de SLAM est confronté à une problématique typique : les points de repères extraits sont les mêmes à des temps et positions différentes. En effet, il n'est pas possible d'extraire des caractéristiques spatiales autres que des murs droits, jusqu'à ce que l'entrebailement de la porte du bureau 2 ne soit visible. Dans ce cas, l'association de données interne au processus

de SLAM va fusionner les repères perçus à différents points de la zone traversée, générant des erreurs de cartographie et de localisation. Ce phénomène est observable lors du téléguidage du robot à travers le couloir, Hector SLAM va retourner des positions successives identiques sur quelques mètres alors que la position réelle du robot a évolué. La zone bleue sur le plan d'évacuation représente la portion de l'espace parcouru sur laquelle Hector SLAM a estimé que la plateforme faisait du « sur-place ».

Une réponse à cette problématique pourrait être de se fier davantage à l'odométrie, puisque les encodeurs des roues du robot détiennent l'information jusqu'ici manquante : le robot a avancé ou reculé. Or Hector SLAM a été choisi spécifiquement pour sa faculté à se baser en priorité sur les données issues du LIDAR. Ce moteur de SLAM présente donc des limites intrinsèques pour une utilisation dans des environnements lisses, où peu de repères spatiaux évidents sont perceptibles. L'amélioration des résultats dans notre couloir reviendrait à utiliser un LIDAR à plus grande portée, capable de déceler des caractéristiques plus lointaines¹.

Enfin, la réussite du projet dans sa globalité a pu être attestée par les résultats d'une présentation devant des collaborateurs commerciaux et directeurs de projet qui est intervenue le 13 juin 2017. Cette étape a permis de recueillir des avis extérieurs sur le projet lui-même et d'estimer sa potentielle viabilité commerciale et technique. Ayant été positivement perçu, le système démonstrateur réalisé a été présenté brièvement auprès d'interlocuteurs de Nexter Systems, enclins à organiser une rencontre à cet effet au mois de septembre.

3.2.2 Pourquoi et comment envisager la continuité de SRT2M ?

La présentation du projet à un client tel que Nexter Systems étant un objectif établi depuis la phase d'analyse préliminaire, nous avons veillé à maximiser le potentiel d'appropriation de ce dernier par de tierces personnes. Cette volonté s'est incarnée par la mise en place des éléments suivants :

- un manuel utilisateur interactif et ergonomique présenté au sein d'une interface web
- un manuel d'installation, agrémenté de résolutions problèmes pouvant être rencontrés lors de cette phase
- la création d'une documentation automatique et, là aussi, interactive grâce au logiciel Doxygen
- l'automatisation de l'installation pour la distribution Debian 8, notamment au moyen du gestionnaire de paquets apt, et la gestion des dépendances manquantes sur la station hôte
- l'adoption de ROS répondant à un critère de haute flexibilité fonctionnelle et matérielle
- l'adaptation des possibilités de l'IHM au caractère modulaire des éléments du middleware ROS

Par ailleurs, ce projet mené de bout en bout a suscité un foisonnement d'idées applicatives, dont nous rappelons ici le scénario majeur : une application militaire facilitant l'accès à des informations stratégiques. Tout d'abord, la mise en œuvre d'algorithmes de SLAM aboutit généralement sur des systèmes capables de naviguer

1. On rappelle que la portée maximale du RPLidar A2 est de 6m.

en autonomie ou semi-autonomie. Moyennant une adaptation du matériel à cet effet, cette fonctionnalité peut être atteinte rapidement. Elle constitue aussi un pré-requis aux applications envisagées.

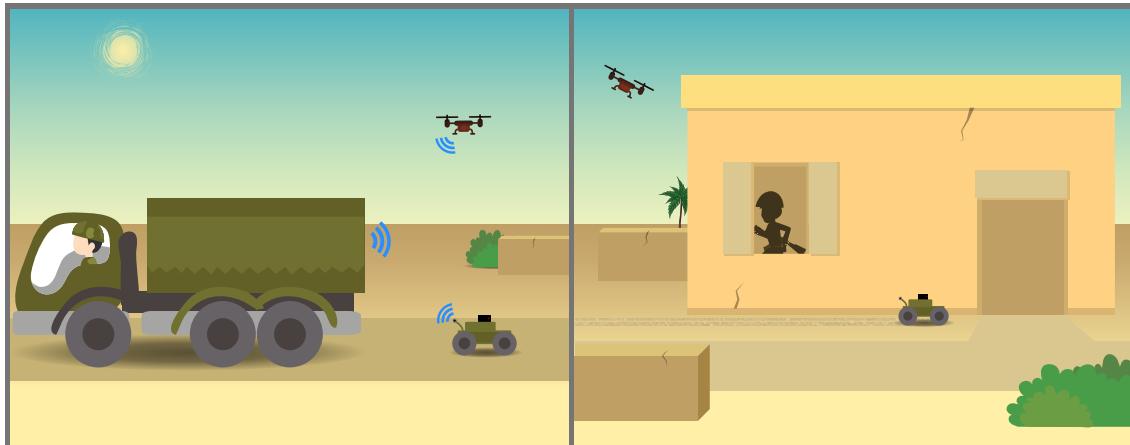


FIGURE 3.6 – Illustration : déploiement et approche furtive de la flotte

Le scénario principal que nous cherchons à illustrer ici est composé d'une flotte de véhicules autonomes, à la fois aériens et terrestres, pilotés à distance par un opérateur (voir figure 3.6). Le système logiciel embarqué est distribué sur chacun des terminaux mobiles, chacun d'entre-eux étant atteignable par ondes radio ou WiFi sur un canal de communication chiffré. L'opérateur va émettre un ordre de mission correspondant à une zone à explorer, une niveau de furtivité et un temps imparti comprenant l'aller et le retour des modules. Une unité maîtresse sera responsable de la répartition des trajectoires à emprunter pour l'ensemble de la flotte, tandis que les calculs peuvent être répartis ou centralisés sur une station de travail abritée. Si le véhicule maître est indisponible ou défectueux, une autre unité endosse son rôle immédiatement. La



FIGURE 3.7 – Illustration : réception de données unifiées

réception des données (figure 3.7) sera quant à elle unifiée en une seule cartographie permettant un prise en compte globale des jeux de données reçues.

On peut également envisager l’immersion du système dans des environnements qualifiés de USAR. Comme beaucoup d’applications robotiques issues du framework ROS, Hector SLAM a été éprouvé lors de compétitions dédiées à la recherche de victimes. Dans ce cas, il convient de définir quelles sont les situations de secourisme ciblées par le système : incendie, raz-de-marée, expositions radio-actives ou à d’autres substances toxiques ? La réponse à ces questions permettra d’évaluer le coût du matériel adapté à ces scénarios ainsi que les temps de recherche, spécification, développement et tests susceptibles d’assurer un nouveau produit minimum viable.

3.3 Retour d’expérience

3.3.1 Apports et difficultés du projet

Ce projet s’est trouvé complexe et stimulant à tous les niveaux. Les phases de spécification du besoin ont été fastidieuses tant du fait de la rigueur des documents requis qu’au regard de mon manque de vision intial et de ma difficulté à me représenter ce que serait le système en bout de stage. Cela s’explique notamment par une méconnaissance initiale du domaine de la robotique et du secteur de la défense. Les aides conjointes de MM. Daumand et Hafiane, de vastes recherches documentaires et un intérêt certain pour ce sujet où tout était à construire ont fort heureusement motivé une rapide appropriation des problématiques et enjeux afférents.

Les outils utilisés tels que ROS et à plus faible mesure Qt, ou les fondements théoriques du SLAM puis d’Hector SLAM ont également nécessité des temps de compréhension et d’assimilation conséquents. Bien que n’ayant pas outrepassé le temps imparti à la recherche de solutions techniques, il m’est avéré difficile de passer plusieurs semaines à approfondir des connaissances très spécifiques –comme dans le cas de ROS– sans avoir la moindre idée de ce que donneront les débuts de l’implémentation. Cet effet tunnel s’est trouvé renforcé du fait des délais d’acquisition du matériel et des incertitudes liées au fonctionnement du robot.

Aussi, nous avons eu la chance d’appréhender la réalisation d’un système à naître, sans qu’aucune étude préalable n’ait été menée. Ce point a à la fois constitué un atout majeur dans le choix de ce stage, mais également une incertitude indéniable qui a compliqué la visualisation du système final. Par exemple, la quantification précise du temps alloué aux diverses étapes de conception ou de réalisation s’est révélée être une pratique nouvelle et à la fois primordiale pour mener à bien ce genre de projets. Ces difficultés ont peut-être avant tout été surmontées par la communication au sein de l’équipe, permettant de désamorcer les situations de doutes et de consolider sereinement une vision du projet à moyen terme.

Enfin, j’ai particulièrement apprécié l’alternance entre une période assez longue de travail individuel, et un travail en équipe –certes restreinte– dans une seconde phase du développement. Dans le premier cas, cela force à une organisation, une prise de décisions et de responsabilités individuelles accrues qui n’ont que très peu été expérimentées lors de projets scolaires au long-cours. Le deuxième temps de développement avec Alban Chazot a permis l’expérimentation d’outils issus de méthodes AGILES et adaptés à notre équipe minimale. Cette deuxième phase s’est faite sur un ton plus détendu, les fonctionnalités primordiales du système ayant été présentées et validées.

3.3.2 Évolution personnelle au sein de la structure

Bien qu'elle soit inclue dans un groupe international, l'agence SII Bourges affiche un « esprit *startup* » indéniable. L'équipe sur place est à taille humaine, d'une moyenne d'âge située autour d'une trentaine d'années et présente des normes décontractées soutenues par une hiérarchie horizontale. La société SII dénote d'une envie de séduire les plus jeunes actifs en brevetant le slogan #FUNgenieur qui vise à “*dépoussiérer l'image du geek austère*”, en proposant salles de jeu, de sport ou de détente dans ses agences et, finalement, en misant sur un management de proximité accru. À cet effet, Mme Aurélie Merlin, Campus Manager rattachée à l'agence Île-de-France encadre la communauté de stagiaires et participe à la faire vivre par le biais de défis hauts en couleurs : concours photo, présentation des stages en trois minutes, après-midi dédiés aux échanges entre stagiaires. Cette culture d'entreprise atypique fait de SII une entreprise favorable à une intégration réussie, permettant d'y envisager facilement un début de carrière professionnelle.

Souhaitant exercer dans le domaine de la sécurité informatique, et mettre rapidement à profit la formation reçue à l'INSA Centre Val-de-Loire, j'ai accepté un poste de consultante en cyber-sécurité au sein de SII. Cette contractualisation concerne les clients ASD du groupe, et plus particulièrement la société MBDA pour laquelle SII s'inscrit en tant que prestataire depuis plusieurs années. Dès la rentrée prochaine, ma mission consistera dans un premier temps à appliquer des politiques de durcissement de systèmes d'exploitations Linux et Windows. Dans cette optique, je serais affectée au site du Dynasteur de l'agence SII IDF. Cette embauche s'accompagne d'une possibilité de financement de formations certifiantes dans les domaines de la sécurité informatique (réseau, système ou applicative), constituant un facteur de motivation non négligeable.

Conclusion

Pour conclure ce rapport, nous reviendrons sur les problématiques majeures à prendre en compte afin d'envisager une potentielle industrialisation du système développé. Nous établirons ensuite le bilan des apports de ce stage d'un point de vue personnel, en tant que stagiaire et élève ingénierie. Puis nous tenterons de dégager quelques bénéfices que peuvent en tirer SII et l'INSA Centre Val-de-Loire au moins sur le court terme.

D'abord le travail accompli jusqu'ici laisse volontairement de côté les tests des différentes briques logicielles. Cette étape indispensable se chiffre généralement comme étant deux fois supérieure au développement, en terme de temps de réalisation. Bien qu'adapté à un démonstrateur, le matériel utilisé devra également être repensé en intégralité pour s'adapter à une application professionnelle. S'il est convenu que le moteur de SLAM représente un premier pas vers un véhicule autonome, la panoplie de périphériques à disposition mérite d'être complétée pour atteindre cet objectif. Nous arrivons jusqu'ici à conférer au système une perception de l'environnement de la plateforme mobile et de la position de la plateforme elle-même. Avec un LIDAR à technologie 2D –comme c'est le cas du RPLidar A2– il est cependant impossible de déceler les obstacles en dessous et au dessus du plan défini par les rayons laser. Ces obstacles pouvant altérer l'état du matériel ou plus généralement la planification de chemin au sein de la carte, une conduite autonome du véhicule ne peut être envisagée dans ces conditions. Afin d'atteindre pleinement cette fonctionnalité, on peut envisager de se doter d'un LIDAR qui cartographie l'espace en trois dimensions. On peut également équiper le dispositif mobile d'un jeu suffisant de capteurs infrarouges, capables de détecter l'ensemble des obstacles susceptibles d'atteindre la plateforme quelque soit leur hauteur. On souligne aussi un manque de sécurisation des données et communications inhérentes au projet. Les requêtes et réponses entre le poste de travail et le robot correspondent au fonctionnement normal du Wifibot : une fois l'accès au routeur WiFi établi, aucune procédure visant à l'authentification ou à la confidentialité n'est effectivement mise en place. Ce point revêt une intérêt particulier –pour ne pas dire critique– si l'on admet que l'outil développé puisse servir à la défense nationale.

En termes de rétrospectives, ce stage a indéniablement impliqué l'acquisition de nouvelles connaissances techniques, théoriques et organisationnelles. Il a aussi eu le mérite de ne pas fermer la porte au domaine de la sécurité bien qu'il en soit apparemment éloigné. Les enseignements tirés de ce projet se retrouvent aussi bien dans ses spécificités techniques ou théoriques, que dans sa transversalité au regard des aspects traités. En effet, il a su soulever des problématiques de développement logiciel pur, des problématiques systèmes en considérant l'ubiquité de certains composants et des problématiques réseau assez bas-niveau notamment au travers de l'utilisation de l'API sockets. La globalité du projet a également suscité l'intérêt de parties prenantes

internes et externes à SII, constituant une note très positive pour l'équipe projet. Le travail accompli jusqu'ici ayant été reçu favorablement, nous pouvons espérer que SII et l'INSA CVL soient confortés dans leur démarche de partenariat. Nous les souhaitons enclins à organiser de nouveaux stages orientés vers le développement robotique, les problématiques de SLAM ou de classification automatique d'objets associées à des technologies disruptives et en plein essor.

Glossaire

API

Application Programming Interface est une interface donnant accès à un ensemble de méthodes et fonctions normalisées facilitant le développement d'applications.

ASD

Aero Space Defense est une Business Unit définie par SII.

bagfile

Un fichier bag (ou bagfile) est un fichier spécifique à l'écosystème de ROS qui contient des enregistrements de messages sérialisés variés et les timestamps afférents. ROS permet l'enregistrement et la lecture de tels fichiers par le biais d'outils tels que l'exécutable rosbag. Ces fichiers sont utilisés à des fins d'analyse ou de simulations de situations pratiques pendant lesquelles des nœuds émettent et / ou reçoivent des données.

BAM

Banque Assurance Mutuelle est une Business Unit définie par SII.

BU

Une Business Unit est une unité organisationnelle au sein d'une entreprise, s'articulant autour d'un domaine d'activité donné. Par la création de BU, l'entreprise entend généralement augmenter son chiffre d'affaires et sa marge brute en conférant d'avantage d'autonomie financière et décisionnelle à ces unités stratégiques. On parle également de département, de division ou de domaine fonctionnel.

Catkin

Catkin est le nom de la chaîne de compilation utilisée par ROS. Basée sur des macros CMake, catkin automatise un certain nombre de tâches, dont la recherche de packages ROS. Ainsi, Catkin permet la génération d'exécutables, de bibliothèques, d'interfaces exportées ou de scripts auto-générés, chacune de ces cibles devant appartenir et être générées depuis un package contenant le code source requis.

classe

La classe d'un objet correspond à un champ textuel qui permet de le qualifier. Ce label est fourni par le réseau neuronal et résulte des opérations de détection et de classification. Par exemple, on peut observer les classes "humain", "chaise" ou "ordinateur" pour un réseau neuronal destiné à être employé dans un environnement de bureau.

DCL

Document de Conception du Logiciel.

EKF

Le filtre de Kalman (KF) est une méthode d'estimation permettant de caractériser l'évolution d'un système dans le temps à partir de mesures imprécises ou bruitées. Ce type de filtre dispose non seulement d'une capacité prédictive à partir de l'état précédent, mais aussi d'une capacité de correction du modèle et des erreurs. Le filtre de Kalman étendu (EKF) permet l'application du filtre de Kalman à des modélisations non linéaires, moyennant un surplus calculatoire.

ETR

Energie Transport Retail est une Business Unit définie par SII.

GeoTIFF

GeoTIFF spécifie un format de description associé à des images TIFF. Ce standard appartient au domaine public. Le format TIFF pour Tagged Image File Format vise à encapsuler des images numériques pour leur adjoindre des informations diverses, il est quant à lui déposé par Adobe.

Hector SLAM

Hector SLAM est le nom d'un métapackage ROS disponible par le biais du gestionnaire de version Git. Il contient entre-autres un nœud de SLAM, nommé hector_mapping. Hector SLAM est sous licence BSD et a été développé par des chercheurs de l'université de Darmstad (Allemagne) dont Stefan Kohlbrecher et Johannes Meyer. Ses fondements théoriques apparaissent dans une publication datant de 2011[13].

IDF

Île-de-France.

IHM

Interface Homme-Machine.

IMU

Inertial Measurement Unit, désigne des données issues de capteurs inertIELS, typiquement présents dans des systèmes robotiques ou des véhicules aériens sous la forme de gyromètres et d'accéléromètres. De telles mesures qualifient le roll, le pitch et le yaw (roulis, tangage et lacet en français), à savoir les valeurs des composantes sur les trois degrés de liberté d'un véhicule.

Kanban

Mot japonais signifiant "étiquette". Dans le domaine du développement logiciel le Kanban (issu de la méthode japonaise du même nom) consiste à instaurer des métriques visuelles attestant l'avancement d'un projet tout en facilitant sa gestion. Il s'agit généralement d'un tableau où chaque colonne correspond à une étape du processus de réalisation et au travers duquel on déplace des items de gauche vers la droite en fonction de l'état dans lequel ils se trouvent.

KNN

K-Nearest Neighbor est un algorithme qui permet de définir la sortie associée à une entrée donnée en considérant les K plus proches voisins connus de cette entrée. La notion de voisinage est sous-tendue par l'application d'une formule de distance entre l'entrée à traiter et les échantillons connus.

LIDAR

LIght Detection And Ranging est un dispositif physique muni d'un ou de multiples faisceaux lumineux ainsi que de capteurs permettant la mesure de distances spatiales entre le matériel émetteur et les obstacles rencontrés par les ondes émises.

NTIC

Nouvelles Technologies de L'Information et de la Communication.

POC

Proof Of Concept (preuve de concept).

points de repères

Les points de repères sont des caractéristiques observables par le biais de capteurs du robot, lui permettant de se situer dans son environnement et de le cartographier. La qualité d'un point de repère s'évalue en fonction des caractéristiques suivantes :

- la facilité de ré-observabilité
- la possibilité de discriminer des points de repères individuels
- leur quantité dans l'environnement
- leur caractère stationnaire

Un exemple typique est une ligne droite et des coins bien définis tels que des murs délimitant une pièce.

QHSE

Qualité Hygiène Sécurité Environnement. Située au niveau de l'agence Île-de-France, la cellule QHSE veille activement sur le respect de la qualité opérationnelle, tout en s'inscrivant dans une démarche d'amélioration continue. Elle est en charge de la préparation et du maintien des niveaux de certifications de l'entreprise. Enfin, elle est impliquée dans la mise en pratique d'une démarche RSE, notamment par la prise en compte des impacts environnementaux de l'activité.

R&D

Recherche et Développement.

ROS

Robot Operating System.

RSE

Responsabilité Sociétale de l'Entreprise.

SA à directoire et conseil de surveillance

Une Société Anonyme est une forme juridique de société commerciale, dont la dénomination sociale permet entre-autre de protéger l'anonymat de ses actionnaires. SII correspond à une SA à directoire et conseil surveillance. Moins répandue qu'une SA à conseil d'administration[3], ce statut mène à identifier distinctement deux organes de gouvernance dont le directoire représente la partie exécutive. Le conseil de surveillance endosse quant à lui les tâches de nomination et de contrôle du directoire.

SCRUM

SCRUM est une méthode AGILE apparue au début des années 1990, basée sur l’empirisme et l’amélioration continue. Les principes clés sont aujourd’hui exposés dans un document de référence appelé SCRUM guide.

SDK

Software Development Kit.

serveur de paramètres

Dans un écosystème ROS, le serveur de paramètres est un dictionnaire partagé entre les différents noeuds actifs, accessible grâce aux APIs réseaux. Il n'est pas destiné à supporter une montée en charge importante, de telle sorte que son utilisation doit se limiter à stocker les paramètres d'utilisation des noeuds. Ces derniers pourront récupérer leurs variables de configuration au *runtime* en invoquant le serveur. Le serveur de paramètres utilise le protocole XML-RPC et est implémenté à l'intérieur du ROS Master.

services

Les services sont des éléments de communication inter-noeuds qui permettent l'envoi de requêtes et de réponses. La création de services passe par la définition textuelle des types de données attendues dans la requête et la réponse.

SI

Système d’Information.

SII

Société pour l’Informatique Industrielle.

SLAM

Le SLAM pour Cartographie et Localisation Simultanées (de l’anglais Simultaneous Localization And Mapping) est une discipline permettant de conférer à un dispositif mobile la perception de son environnement externe, sous la forme d’une cartographie des points de repères stationnaires, tout comme de ses propres positions et orientations au sein de cet environnement. Le SLAM est mis en œuvre par le biais de capteurs internes (mesures odométriques ou inertielles) et externes (LIDAR, RADAR, caméra).

SRT2M

Système Robotique Tactique Multi-Missions pour la surveillance et l'aide à la prise de décisions dans les milieux à risques. C'est le nom donné au système final intégrant les briques logicielles et physiques issues de mon stage et de celui d'Alban Chazot.

STBL

La Spécification Technique du Besoin Logiciel est un document généralement réalisé par un client pour spécifier le besoin qui justifie la réalisation d'un logiciel.

TCP

Transport Control Protocol est un protocole de la couche transport qui permet entre-autre la remise d'accusés de réception à l'émetteur (flag ACK) dans un environnement client-serveur. Cette particularité vise à assurer la fiabilité de la communication établie.

tf

Tf, pour *The transform library* est le nom d'un package ROS qui permet de définir et mettre en relation différents systèmes de coordonnées utiles

aux noeuds de calcul, et de mettre à jour ces relations au cours de leurs déplacements respectifs.

topics

Les topics permettent la communication entre les différents noeuds. Un noeud est responsable de la publication des données du topic (commandes de contrôle par exemple) et un noeud distinct souscrit à ce même topic pour les récupérer (par exemple le noeud qui va traiter ses commandes). Typiquement, la publication d'un topic se fera en C++ par l'appel à la méthode advertise(). Parallèlement, la souscription à un topic nécessite l'appel à la méthode subscribe().

USAR

Urban Search And Rescue, est un terme qui qualifie les environnements, scénarios ou situations impliquant la recherche, la mise à l'abri et parfois les soins de victimes en espace urbain. Un exemple typique consiste à évacuer des victimes d'un immeuble incendié.

XML

eXtensible Markup Language est un langage informatique balisé et normalisé, traditionnellement utilisé à des fins de transmission de données via Internet car il est indépendant de toute plateforme.

XML-RPC

XML-RPC est un protocole réseau de type *Remote Procedure Call* permettant l'échange de données et l'appel de méthodes par des exécutables distincts. La définition des données comme des procédures est faite par le biais du langage XML.

Bibliographie

- [1] SII Siège Social - Paris, *Document de référence incluant rapport annuel financier. Exercice 2015/2016*, 172 p.
- [2] SII Ile de France - Paris, *Memeto Agence Ile-de-France. Missions, Organisation, Processus*, Version 11, 102 p, Mai 2017.
- [3] Wikipédia, *Société Anonyme*, https://fr.wikipedia.org/wiki/Soci%C3%A9t%C3%A9_anonyme 21 juin 2017.
- [4] Frank Canton relayé par le Ministère de l'intérieur, *Armement et défense, traditions du Cher*, <https://www.interieur.gouv.fr/Archives/Archives-des-dossiers/2016-Dossiers/Le-Cher-Armement-et-defense-traditions-du-Cher> janvier 2016.
- [5] Cognitive Robotics, Massachusetts Institute of Technology, Søren Riisgaard and Morten Rufus Blas *SLAM for Dummies A Tutorial Approach to Simultaneous Localization and Mapping*, https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam blas_repo.pdf Spring 2005
- [6] Cyril Stachniss, Udo Frese, Giorgio Grisetti, *Give your algorithm to the community*, <https://www.openslam.org> Spring 2005
- [7] ROS Documentation, *Master* <http://wiki.ros.org/Master>
- [8] ROS Documentation, *Nodes* <http://wiki.ros.org/Nodes>
- [9] ROS Documentation, *Topics* <http://wiki.ros.org/Topics>
- [10] ROS Documentation, *Services* <http://wiki.ros.org/Services>
- [11] ROS Documentation, *catkin/package.xml* <http://wiki.ros.org/catkin/package.xml>
- [12] ROS Documentation, *catkin/CMakeLists.txt* <http://wiki.ros.org/catkin/CMakeLists.txt>
- [13] S. Kohlbrecher and J. Meyer and O. von Stryk and U. Klingauf, *A Flexible and Scalable SLAM System with Full 3D Motion Estimation* http://www.sim.informatik.tu-darmstadt.de/publ/download/2011_SSRR_KohlbrecherMeyerStrykKlingauf_Flexible_SLAM_System.pdf 2011
- [14] Stefan Kohlbrecher, Christian Rose, Dorothea Koert, Paul Manns, Florian Kunz, Benedikt Wartusch, Kevin Daun, Alexander Stumpf and Oskar von Stryk, *RoboCup Rescue 2016 Team Description Paper Hector Darmstadt* http://www.robocup2016.org/media/symposium/Team-Description-Papers/RescueRobot/RoboCup_2016_RescueR_TDP_HectorDarmstadt.pdf 2016
- [15] Site officiel de l'équipe de développement et de maintien d'Hector SLAM, <http://www.teamhector.de/>

- [16] Stefan Kohlbrecher, *Hector SLAM Git repository* https://github.com/tu-darmstadt-ros-pkg/hector_slam 29 mars 2011
- [17] ROS Documentation, *Debian install of ROS Kinetic* <http://wiki.ros.org/kinetic/Installation/Debian>
- [18] ROS Documentation, *tf* <http://wiki.ros.org/tf>
- [19] Foote, Tully, in Technologies for Practical Robot Applications (TePRA), *tf : The transform library* http://wiki.ros.org/Papers/TePRA2013_Foote?action=AttachFile&do=get&target=TePRA2013_Foote.pdf avril 2016
- [20] Wim Meeussen, *Coordinate Frames for Mobile Platforms* <http://www.ros.org/reps/rep-0105.html> 27 octobre 2010
- [21] *ROS Press Kit* <http://www.ros.org/press-kit/>
- [22] Qt Documentation, *QMainWindow Class* <http://doc.qt.io/qt-4.8/qmainwindow.html#details>

Annexes

Annexe 1

Cette annexe donne la définition du message `nav_msgs/LaserScan.msg`. Ce type de message est fourni en sortie du nœud `/rplidarNode` par le biais du topic `/scan` présenté figure 2.10. Le type `Header` encapsulé est également défini.

1 Définition du message `nav_msgs/LaserScan.msg`

```
1 # Single scan from a planar laser range-finder
2 #
3 # If you have another ranging device with different behavior (e.g.
4 # a sonar
5 # array), please find or create a different message, since
6 # applications
7 # will make fairly laser-specific assumptions about this data
8
9 Header header          # timestamp in the header is the
10 acquisition time      # of the first ray in the scan.
11                         # in frame frame_id, angles are measured
12                         # around
13                         # the positive Z axis (counterclockwise,
14                         # if Z is up)
15                         # with zero angle being forward along the
16                         # x axis
17
18 float32 angle_min      # start angle of the scan [rad]
19 float32 angle_max      # end angle of the scan [rad]
20 float32 angle_increment # angular distance between measurements [
21     rad]
22
23 float32 time_increment # time between measurements [seconds] - if
24     your
25                         # scanner is moving, this will be used in
26                         # inetrpolating position of 3d points
27 float32 scan_time       # time between scans [seconds]
28
29 float32 range_min       # minimum range value [m]
30 float32 range_max       # maximum range value [m]
31
32 float32[] ranges        # range data [m] (Note: values < range_min
33     or > range_max should be discarded)
34 float32[] intensities   # intensity data [device-specific units].
35     If your
36                         # device does not provide intensities,
37                         # please leave
38                         # the array empty.
```

2 Définition du message std_msgs/Header.msg

```
1 # Standard metadata for higher-level stamped data types.
2 # This is generally used to communicate timestamped data
3 # in a particular coordinate frame.
4 #
5 # sequence ID: consecutively increasing ID
6 uint32 seq
7 #Two-integer timestamp that is expressed as:
8 # * stamp.sec: seconds (stamp_secs) since epoch (in Python the
#   variable is called 'secs')
9 # * stamp.nsec: nanoseconds since stamp_secs (in Python the
#   variable is called 'nsecs')
10 # time-handling sugar is provided by the client library
11 time stamp
12 #Frame this data is associated with
13 # 0: no frame
14 # 1: global frame
15 string frame_id
```

Annexe 2

Cette annexe illustre la teneur des fichiers `package.xml` et `CMakeLists.txt` requis dans l'environnement catkin. Ils définissent le package `slam_bridge` réalisé pour ce projet, contenant les noeuds `map_bridge`, `trajectory_bridge` et `pose_bridge`.

1 package.xml

```
1 <?xml version="1.0"?>
2 <package>
3
4   <!-- Description du projet -->
5   <name>slam_bridge</name>
6   <version>1.0.0</version>
7   <description>The slam_bridge package</description>
8   <maintainer email="lisa.aubry[at]insa-cvl.fr">laurbry</maintainer>
9   <license>TODO</license>
10
11  <!-- Outil de build -->
12  <buildtool_depend>catkin</buildtool_depend>
13
14  <!-- Dépendances de build -->
15  <build_depend>roscpp</build_depend>
16  <build_depend>geometry_msgs</build_depend>
17  <build_depend>rospy</build_depend>
18  <build_depend>std_msgs</build_depend>
19  <build_depend>nav_msgs</build_depend>
20
21  <!-- Dépendances d'exécution -->
22  <run_depend>roscpp</run_depend>
23  <run_depend>geometry_msgs</run_depend>
24  <run_depend>rospy</run_depend>
25  <run_depend>std_msgs</run_depend>
26  <run_depend>nav_msgs</run_depend>
27
28 </package>
```

2 CMakeLists.txt

```
1 cmake_minimum_required(VERSION 2.8.3)
2 project(slam_bridge)
3
4 ## Add support for C++11, supported in ROS Kinetic and newer
5 add_definitions(-std=c++11)
6 add_definitions(-g)
7
8 find_package(catkin REQUIRED COMPONENTS roscpp geometry_msgs
9     std_msgs nav_msgs rospy)
10 catkin_package(CATKIN_DEPENDS roscpp geometry_msgs rospy std_msgs
11     nav_msgs)
12 #####
13 ## Build ##
14 #####
15 ## Declare a C++ executable
16 include_directories(${catkin_INCLUDE_DIRS} src )
17
18 add_executable(trajetory_bridge
19     src/TrajectoryBridge.cpp
20     src/TrajectoryMain.cpp
21     src/Serializer.cpp
22 )
23
24 add_executable(pose_bridge
25     src/PoseBridge.cpp
26     src/PoseMain.cpp
27     src/Serializer.cpp
28 )
29
30 add_executable(map_bridge
31     src/MapBridge.cpp
32     src/MapMain.cpp
33     src/Serializer.cpp
34 )
35
36 ## Specify libraries to link a library or executable target against
37 target_link_libraries(pose_bridge
38     ${catkin_LIBRARIES}
39 )
40
41 target_link_libraries(map_bridge
42     ${catkin_LIBRARIES}
43 )
44
45 target_link_libraries(trajetory_bridge
46     ${catkin_LIBRARIES}
47 )
48
49 #####
50 ## Install ##
51 #####
52 ## Mark other files for installation (e.g. launch and bag files,
53     etc.)
54 install(DIRECTORY launch/
55     DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}/launch/
55 )
```

Annexe 3

Cette annexe détaille les types de données énumérés dans le document de Spécification Technique de Besoin Logiciel qui ont été évoqués plus en amont dans ce rapport, section Application du référentiel qualité interne. On rappelle que la présentation de ces types est motivée par l'idée d'exposer une démarche documentaire sous son aspect formel – le contenu des données elles-mêmes ou leur signification n'ayant pas de réel intérêt ici. Notons qu'à chaque fois qu'un champ issu d'une structure de données correspond également à un type structuré, on déroulera en cascade la définition jusqu'à ce que tout puisse être décrit avec des types de base.

1 Message OCCUPANCY_GRID

Topic : /map

Type : nav_msgs/OccupancyGrid

Emetteur : /hector_mapping

Destinataire : /map_bridge

Fréquence d'émission moyenne : 0,5 Hz

Description : Emis par SLAM, donne la probabilité d'occupation de la grille représentant la carte

Contenu :

Contenu	header	info	data
Type	std_msgs/Header	std_msgs/MapMetaData	int8[]

FIGURE 3.8 – Contenu du message Occupancy_Grid

1.1 Header

Contenu	seq	stamp	frame_id
Type	uint32	time	string

FIGURE 3.9 – Contenu du message Header

1.2 Info

Contenu	map_load_time	resolution	width	height	origin
Type	time	float32	uint32	uint32	geometry_msgs/Pose

FIGURE 3.10 – Contenu du message Info

1.3 Origin

Contenu	position	orientation
Type	geometry_msgs/Point	geometry_msgs/Quaternion

FIGURE 3.11 – Contenu du message Origin

1.4 Position

Contenu	x	y	z
Type	float64	float64	float64

FIGURE 3.12 – Contenu du message Position

1.5 Orientation

Contenu	x	y	z	w
Type	float64	float64	float64	float64

FIGURE 3.13 – Contenu du message Orientation

1.6 Data

Représente une carte à 2 dimensions où chaque cellule contient sa probabilité d'occupation. Les lignes ont la priorité dans l'indexage de la carte. La carte commence à la position (0, 0). Les probabilités sont comprises dans [0, 100]. -1 représente une case non explorée.

Annexe 4

1 Planning hebdomadaire

Cette annexe restitue les éléments de planification hebdomadaire mis en place pour ce projet. Pour des raisons de présentation au sein de ce document, le planning a été scindé selon les parties suivantes :

- un aperçu global du planning, excluant le nom des tâches et intégrant les jalons, dead-lines et incrément produits clés
- une présentation détaillée des tâches de réalisations documentaires, et de préparation de l'environnement de travail
- une présentation détaillée des tâches d'étude et des tâches de réalisation liées à ROS
- une présentation détaillée des tâches de réalisation liées à l'IHM, d'automatisation, de gestion de versions et de tâches annexes

Les versions du système répondent au code couleurs suivant : gris correspond aux tâches effectuées avant la première release, violet après la première et bleu après la seconde.

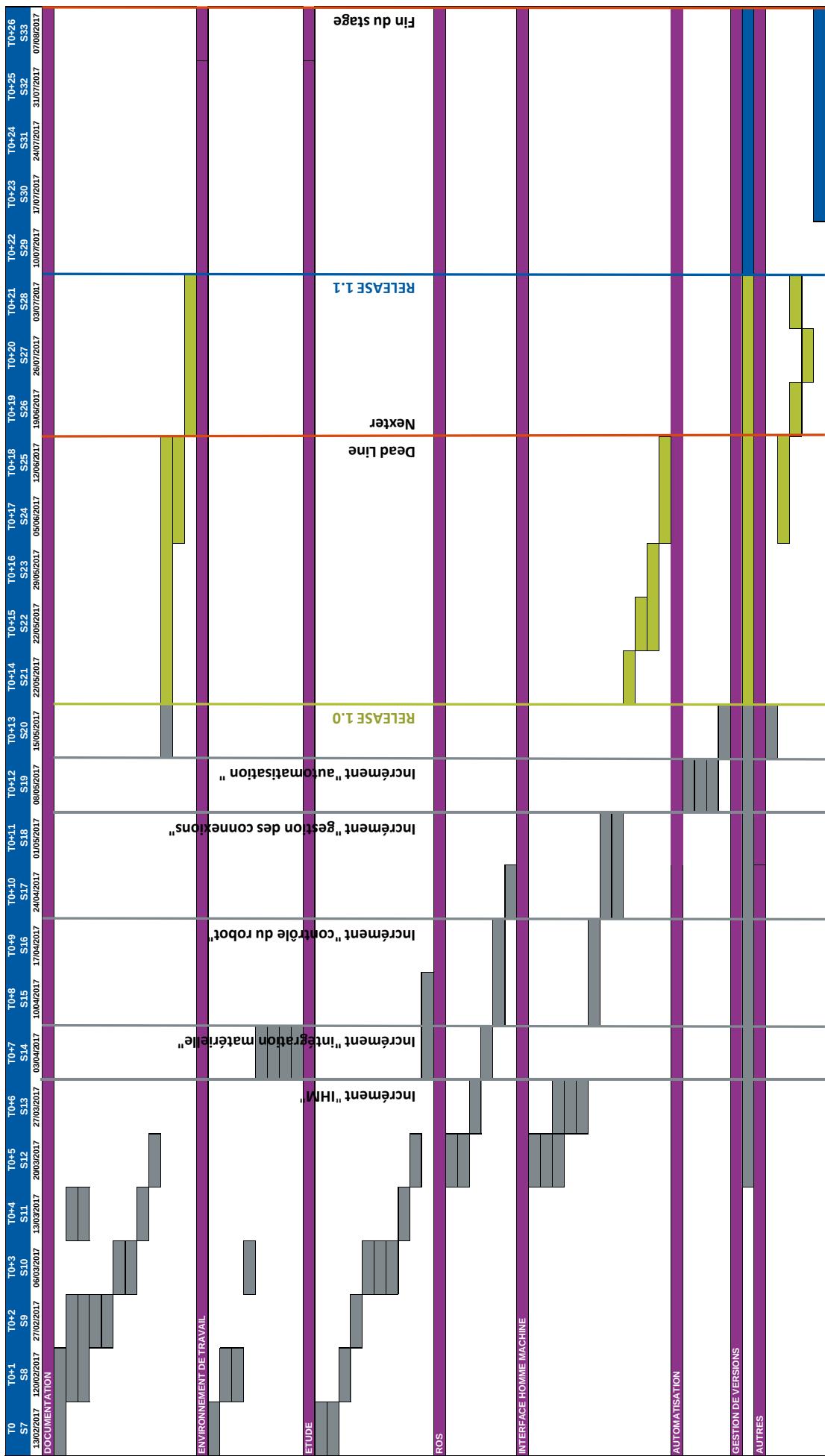


FIGURE 3.14 – Aperçu du planning complet, jalons et dead-lines clés, sans la dénomination des tâches

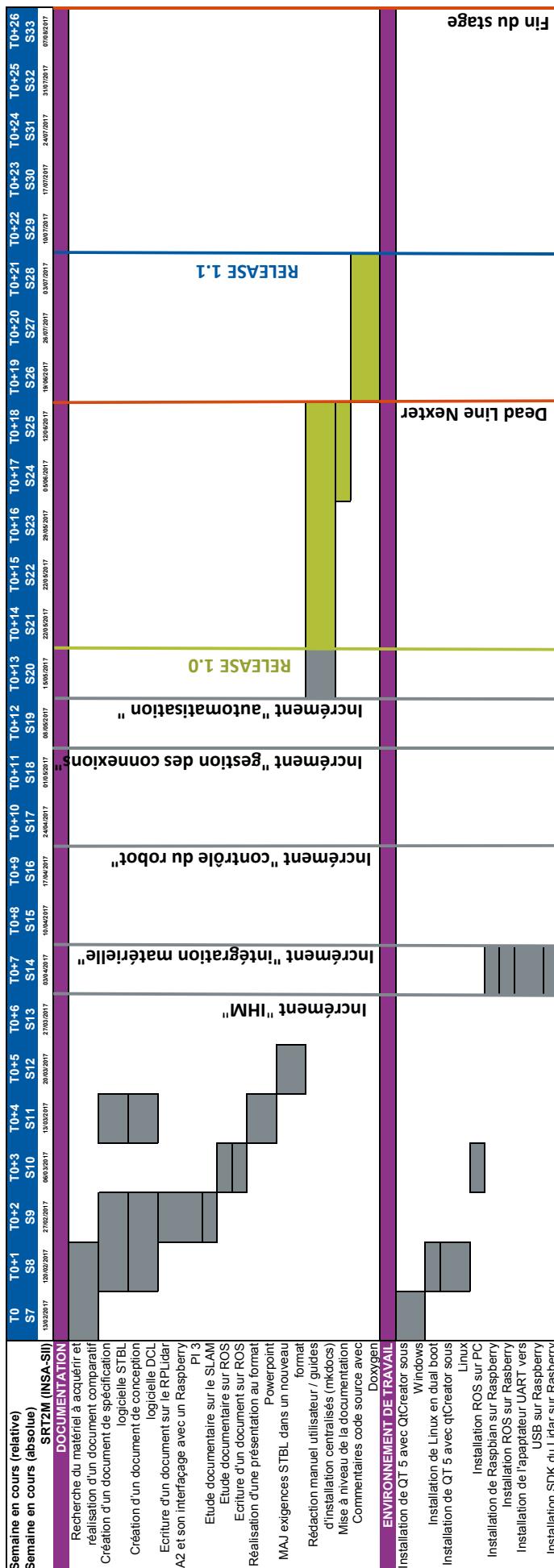


FIGURE 3.15 – Détail du planning, partie 1

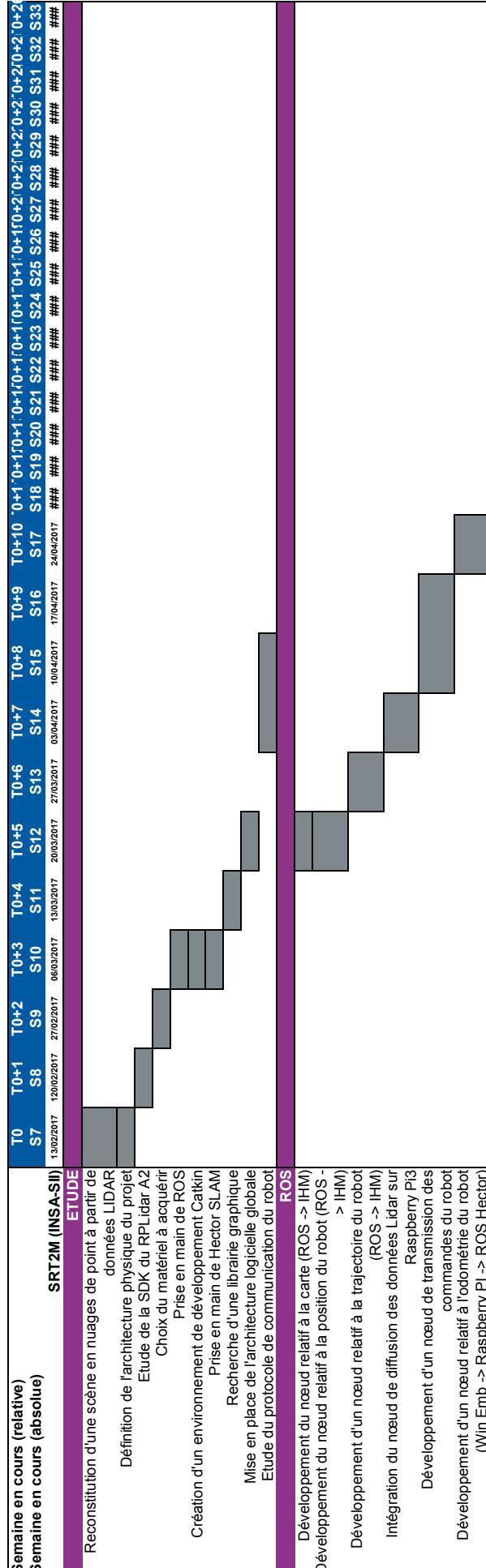


FIGURE 3.16 – Détail du planning, partie 2

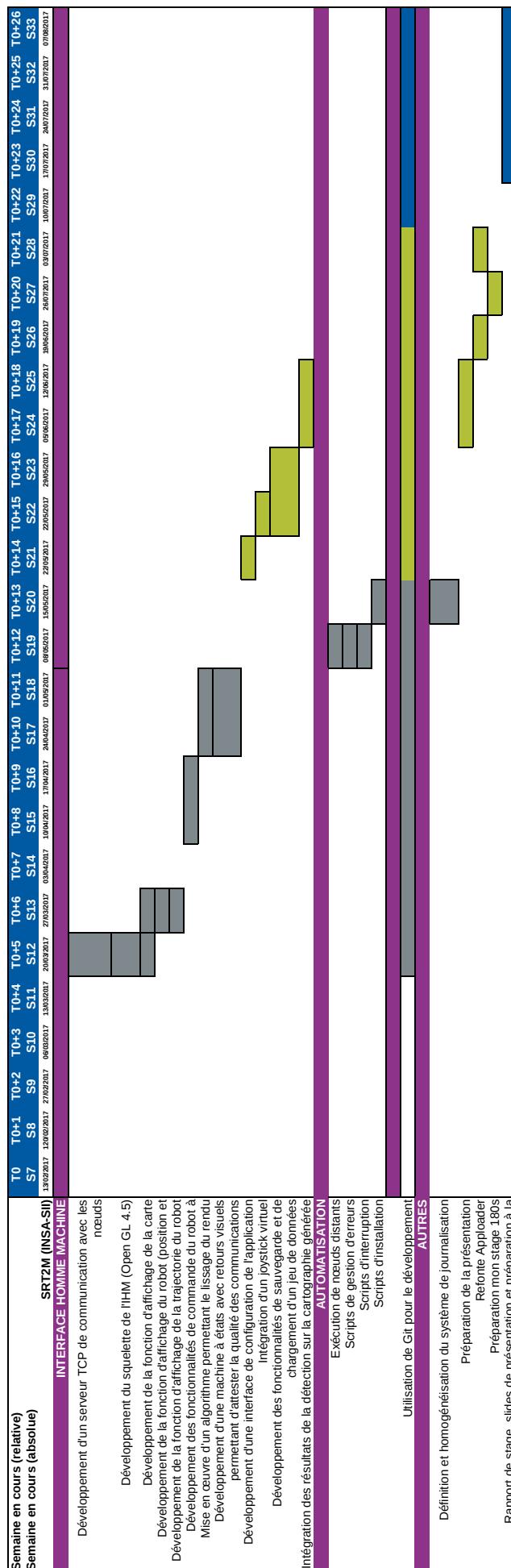


FIGURE 3.17 – Détail du planning, partie 3