



## Résumé

Pour répondre aux besoins qui motivent leur réalisation, les systèmes robotiques mobiles doivent s'appuyer sur une connaissance plus ou moins fiable de leur environnement. On distingue la perception de l'environnement externe du robot de la perception que le robot a de lui-même –en tant que plateforme mobile– au sein de cet environnement. Ces deux versants, qui tiennent une place prépondérante dans le domaine des véhicules autonomes ou intelligents, sont connus sous le nom de SLAM (Localisation et Cartographie Simultanées). Ce rapport aborde ces questions au travers de la spécification, de la conception et de l'implémentation de briques logicielles destinées à contrôler un robot mobile et rendre état de son environnement externe tout comme de sa localisation en temps réel. Le projet décrit dans ce document recouvre le système complet, allant des périphériques physiques d'acquisition de données jusqu'à la présentation des résultats au travers d'une IHM interactive.

In order to carry out specific tasks, robotic systems must rely on a consistent environment perception. Two types of environment perception can be distinguished. The first is about external environment, and the second is about the location of the mobile platform in this environment. These two complementary aspects are known as SLAM (Simultaneous Localisation And Mapping). This report addresses these issues through the specification, the conception and the implementation of software components.

## 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 inaltérables 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 ses orientations 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, Directeur de site, collaborateurs qui ont jalonnés mon quotidien et enrichi mon expérience professionnelle nouvelle dans ses aspects techniques, organisationnels et bien sûr humains.

# Table des matières

<b>Remerciements</b>	<b>1</b>
<b>1 L'environnement du stage, la Société pour l'Informatique Industrielle</b>	<b>2</b>
1.1 Introduction . . . . .	2
1.2 Principaux repères et évolution de la structure SII . . . . .	2
1.2.1 Signalétique générale : du groupe à l'agence parisienne . . . . .	2
1.2.2 Évolution de SII en France et à Bourges . . . . .	3
1.3 Une mission R&D pour bâtir de nouvelles offres . . . . .	4
1.3.1 SII Research dans l'agence berruyère . . . . .	4
1.3.2 Fruits du partenariat avec l'INSA Centre Val-de-Loire . . . . .	5
<b>2 Définition et mise en oeuvre d'un système de SLAM en temps-réel</b>	<b>6</b>
2.1 De l'expression du besoin à une spécification complète . . . . .	6
2.1.1 Vision et cas d'utilisation . . . . .	6
2.1.2 Formalisation d'exigences . . . . .	7
2.1.3 Matériel et Interfaces . . . . .	8
2.2 Théorie et choix techniques . . . . .	10
2.2.1 Cartographie et Localisation Simultanées . . . . .	10
2.2.2 Le méta-système d'exploitation ROS . . . . .	11
2.2.3 Hector SLAM . . . . .	13
2.3 Réalisations et architecture logicielles . . . . .	15
2.3.1 Le réseau ROS complet . . . . .	15
2.3.2 Interface Homme-Machine avec Qt Creator . . . . .	18
2.3.3 Eléments de modularité mis en place . . . . .	22
<b>3 Bilan et perspectives</b>	<b>23</b>
3.1 Organisation du travail . . . . .	23
3.1.1 Application du référentiel qualité interne . . . . .	23
3.1.2 Vers une conduite AGILE adaptée . . . . .	23
3.2 Des résultats en vue d'un prolongement . . . . .	23
3.3 Des résultats qui laissent envisager une suite . . . . .	23
3.3.1 SRT2M à la fin du stage . . . . .	23
3.3.2 Pourquoi prolonger le projet ? . . . . .	23
3.4 Retour d'expérience . . . . .	23
3.4.1 Apports et difficultés du projet . . . . .	23
3.4.2 Évolution personnelle au sein de la structure . . . . .	24

# Table des figures

1.1	BU actives en IDF . . . . .	3
1.2	Répartition sectorielle de SII en IDF[2] . . . . .	4
1.3	Wifibot embarquant un RPLidarA2 et une caméra Theta S . . . . .	5
2.1	Diagramme de cas d'utilisation du système de cartographie et de localisation d'un robot mobile . . . . .	7
2.2	Exigences de Spécification Technique du Besoin Logiciel . . . . .	8
2.3	Architecture physique du projet . . . . .	9
2.4	Architecture physique du projet . . . . .	10
2.5	Logo du méta-système d'exploitation ROS (issu du kit de presse ROS[21]) .	11
2.6	Noeuds et topics d'intérêt et actifs lors de l'exécution d'Hector SLAM . . . . .	13
2.7	Visualisation avec RViz d'un jeu de données pré-enregistrées . . . . .	14
2.8	Réseau ROS intégré au système . . . . .	15
2.9	Systèmes de coordonnées et transformations au sein du système . . . . .	16
2.10	Définition du paquet générique de transmission de données à l'IHM . . . . .	16
2.11	Machine à état de l'application Qt . . . . .	19
2.12	Modélisation logicielle de l'Interface Homme-Machine . . . . .	20
2.13	Rendu visuel de l'IHM en mode Réception . . . . .	21
2.14	Rendu visuel de l'IHM en mode rejeu . . . . .	22

# 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 appliquée à la robotique. Les briques logicielles implémentées devront permettre de cartographier l'environnement d'un robot mobile, ainsi que de le localiser en quasi-temps réel. Ces deux versants s'appuient prioritairement sur des mesures spatiales, fournies par un LIDAR embarqué sur le robot.

Ce stage s'inscrit 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. Il permet à un opérateur 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 d'une flotte de robots terrestres et aériens qu'il peut téléguider à distance. À mesure que l'exploration progresse, l'opérateur visualise d'une part les données cartographiques et la localisation des véhicules au sein de cette carte, et d'autre part, des flux vidéo émanant de caméras à 360 ° également embarquées. Ces flux vidéo sont soumis à des traitements qui permettent de détecter des objets d'intérêts et de les classifier de manière automatique. C'est Alban Chazot, étudiant ingénieur à l'INSA Centre Val-de-Loire qui a assuré, pendant son stage, la réalisation de cette fonctionnalité. Les classifications sont ensuite incrustées sur la carte générée, de manière à ce que l'utilisateur puisse visionner la classe, la position et la taille des entités détectées en une seule et même zone de rendu.

Le système ainsi mis en place ouvre le champ à divers scénarios d'utilisation. Ceux-ci trouvent leur pertinence dès lors que le recours à des opérateurs humains sur le terrain concerné présente un risque ou de fortes contraintes. Un scénario possible est l'exploration d'environnements accidentés (incendies, incidents nucléaires), où la localisation de victimes (USART) et l'estimation des dégâts sont réalisables de manière sécurisée.

Annonce du plan.

# Chapitre 1

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

### 1.1 Introduction

Ce 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. Cette dernière englobe la structure berruyère qui a hébergé le stage. À ce titre, sa portée et son organisation seront particulièrement détaillées.

Nous spécifierons ensuite les axes stratégiques qui justifient la mise en oeuvre du projet. Une section décrira l'apport souhaité du projet pour SII tandis que la section suivante reviendra sur un acteur particulier de l'écosystème de l'entreprise : l'INSA Centre Val-de-Loire, qui s'inscrit en partenaire financier, stratégique et organisationnel en amont de ce projet.

### 1.2 Principaux repères et évolution de la structure SII

#### 1.2.1 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 une des neuf agences françaises de la société. Elle est placée sous la responsabilité du Directeur d'Agence, M. Didier Bonnet. Son administration est abritée sur le site de Kennedy<sup>1</sup> – principalement dédié aux opérations de gestions – et recouvre deux sites supplémentaires : celui du Dynasteur<sup>2</sup> et celui de Bourges<sup>3</sup> 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.

**BU actives en Ile-de-France**  
(2015/2016)



\*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 responsable 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.

### 1.2.2 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 relever 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 à travers 22 sites sur le territoire.

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 :

- **Les secteurs stratégiques**, que sont ASD et Télécoms. Etant des générateurs forts de valeur, ils concernent des marchés mûrs où SII a su établir des relations pérennes avec ses clients grand-compte.

1. 104 Avenue du Président Kennedy, 75016 Paris

2. 6, 12, 10 Rue Andras Beck, 92360 Meudon

3. 14, allée Charles Pathé, 18000 Bourges

### Répartition sectorielle de SII en Ile-de-France (2015/2016)



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

- **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 billettiques 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 Confidential Defense.

## 1.3 Une mission R&D pour bâtir de nouvelles offres

### 1.3.1 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
- Elargir les connaissances internes en communiquant autour des savoir-faire sollicités
- Bâtir de nouvelles offres à partir de tout ou partie du POC

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

Ce projet s'inscrit 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 stage s'est d'emblée accompagné de l'objectif ambitieux d'être véhiculé auprès de représentants de l'entreprise Nexter Systems, à son terme ou durant des stages suivants. 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.

### 1.3.2 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 découlant du partenariat.

Cette relation privilégiée mène 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. Par ailleurs, elle permet de favoriser l'accueil de stagiaires et d'alternants en provenance de l'institut.



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

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).

Ce 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 par le biais 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.

Les périphériques matériel 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 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 détectés()*).

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

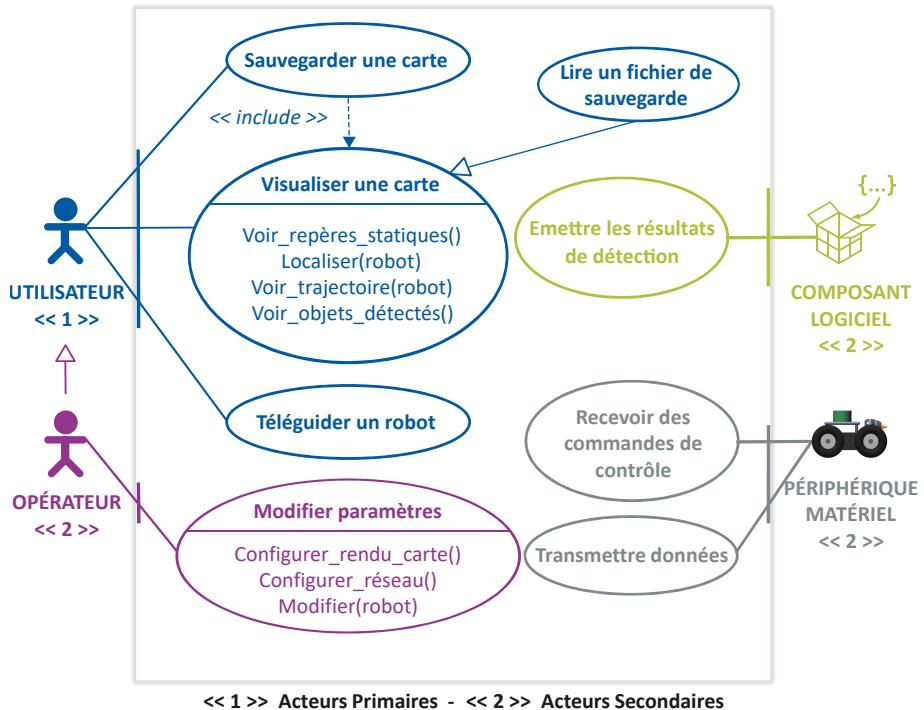


FIGURE 2.1 – Diagramme de cas d'utilisation du système de cartographie et de localisation d'un robot mobile

### 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 système<sup>1</sup>.

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.

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

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 <sup>ème</sup> et 1 <sup>ère</sup> 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

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 physique et logicielle 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 devions 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.

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, selon les 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

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

Par ailleurs, le robot mis à disposition du projet est un Wifibot Lab v3, 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 réuni les modules de calcul et l'IHM.

Nous obtenons ainsi l'archctecture physique schématisée sur la figure 2.3, au travers de laquelle sont définis quatre modules principaux, notés  $M_i$ .

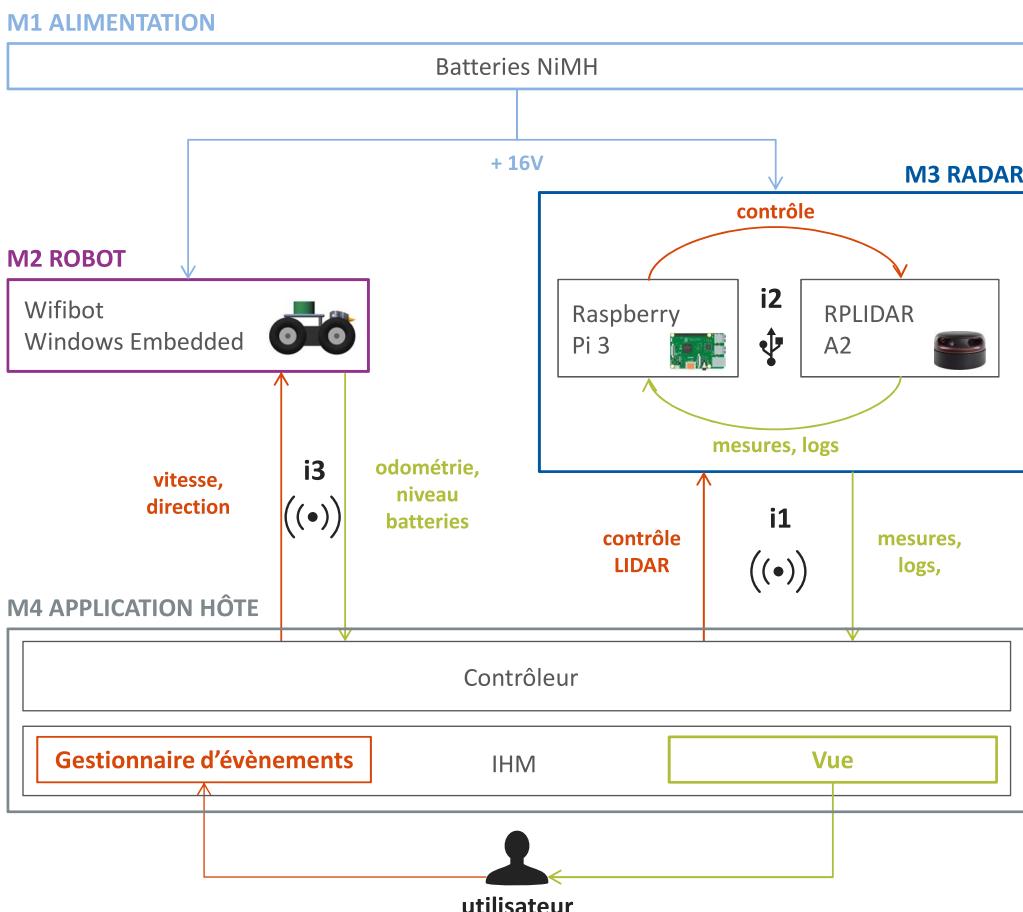


FIGURE 2.3 – Architecture physique du projet

$M_1$  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 Linux<sup>2</sup>.

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

Les interactions entre les briques matérielles sont représentées par des flèches d'entrées / sorties ou internes aux modules, représentées de la couleur orange quand il s'agit de requêtes et de la couleur verte pour les réponses afférentes. Par ailleurs, des interfaces de communications inter-modules notées  $i_i$  apparaissent 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 WiFi<sup>3</sup>  $i_1$ . Ces commandes transitent par le Raspberry Pi 3 qui les formattera et les transmettra au LIDAR via la liaison USB  $i_2$ . Ce cheminement de données peut être parcouru dans le sens inverse pour remonter les données acquises par le LIDAR à savoir, des mesures de distances spatiales, communiquées tous les  $360^\circ$ , pouvant être perçues comme des nuages de points.

Parallèlement,  $M_4$  communique directement des ordres de vitesse et de direction au Wifibot par le biais de 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 Simultanées

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 2.4 donne les grandes lignes.

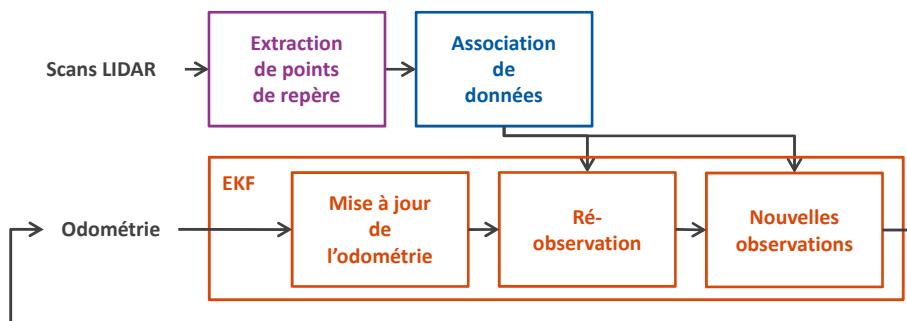


FIGURE 2.4 – Architecture physique du projet

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  $360^\circ$  sur l'axe horizontal (scan plan), 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  $0.45^\circ$  et  $1.35^\circ$ .

Les données odométriques (ou odométrie) permettent la prédiction de la position du robot dans le plan. Ce calcul se base généralement sur les relevés de capteurs internes au robot (différentiel de vitesse entre les deux roues motrices, donné par des encodeurs internes). La difficulté réside lorsqu'il s'agit de corrélérer dans le temps et avec précision ces données, avec

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

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 2.4, consiste à caractériser les points de repère 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 2.4, vise à associer les données collectées, à savoir reconnaître les points de repère 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, le filtre de Kalman étendu est utilisé pour estimer un vecteur d'état du robot (à savoir sa position et son orientation). Dès lors que les briques d'extraction de points de repère 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 2.4. 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 repère 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 repère connus et visible, tandis qu'une innovation forte aura l'effet inverse. Le troisième point permet d'ajouter à notre système les points de repère 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 projet. 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 méta-système d'exploitation ROS



FIGURE 2.5 – Logo du méta-système 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 dédiées à la robotique –notamment par le biais de bibliothèques– 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 fourni 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 internes<sup>4</sup>. 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 X.**

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.

```
catkin_workspace/ ..... racine de l'environnement catkin
  └── src/ ..... emplacement des codes sources
      ├── CMakeLists.txt ... invoqué par CMake lors de la configuration de l'environnement
      ├── package_1/
      │   ├── CMakeLists.txt ... invoqué par CMake lors de la compilation de package_1
      │   ├── package.xml ..... manifeste du package_1
      │   ├── default.launch ..... fichier de lancement des noeuds du paquet
      │   └── src/ .. contient les sources du paquet, implémentant un ou plusieurs noeuds
      ├── package_2/
      │   ├── CMakeLists.txt
      │   ├── package.xml
      │   └── src/
```

Le fichier `package.xml`[11] est le manifeste des paquets. Placé à la racine de chacun 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 noeud du paquet, les sources étant généralement stockées sous le répertoire `catkin_workspace/src/<package_name>/src/`. **Les annexes X et Y donnent des exemples correspondant à ces deux fichiers.**

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 :

---

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

```
1 | > roslaunch 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 noeud 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 un outil 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 implémentations 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 métapackage<sup>5</sup> 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, Hector SLAM s'adapte à une utilisation sur des drones 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.

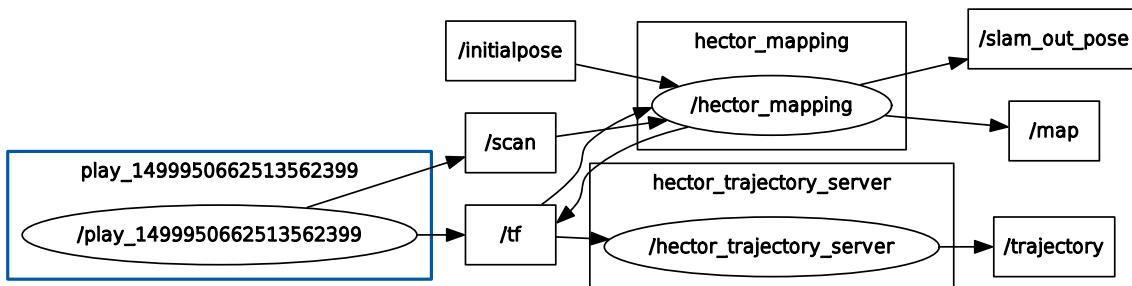


FIGURE 2.6 – Nœuds et topics d'intérêt et actifs lors de l'exécution d'Hector SLAM

La figure 2.6 expose les nœuds et topics actifs lors d'un l'utilisation d'Hector SLAM à partir

5. Un métapackage est simplement pourvu d'un manifeste qui permet de référencer des paquets regroupés logiquement mais faiblement couplés d'un point de vue fonctionnel

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 signalisent 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  $2048 \times 2048$  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 démontre 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  $1^6$

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 ROS<sup>7</sup>. La figure 2.7 illustre le résultat obtenu pour un jeu de test issu de la RobotCup German Open 2011. Dans un deuxième temps, il a fallu adapter Hector SLAM pour recevoir en entrée les données du LIDAR, ce qui revient principalement créer des fichiers de lancement (*launchfile*) adaptés au 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 des éléments qui définissent les transformations spatiales entre les différents périphériques matériels et qui seront évoqués dans la partie suivante.

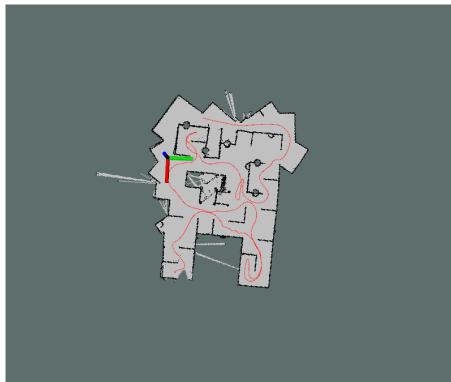


FIGURE 2.7 – Visualisation avec RViz d'un jeu de données pré-enregistrées

Ces différents composants et la manière dont ils ont été réalisés font l'objet de la partie 2.3.1.

---

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].

## 2.3 Réalisations et architecture logicielles

### 2.3.1 Le réseau ROS complet

Hector SLAM a été intégré à un réseau 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

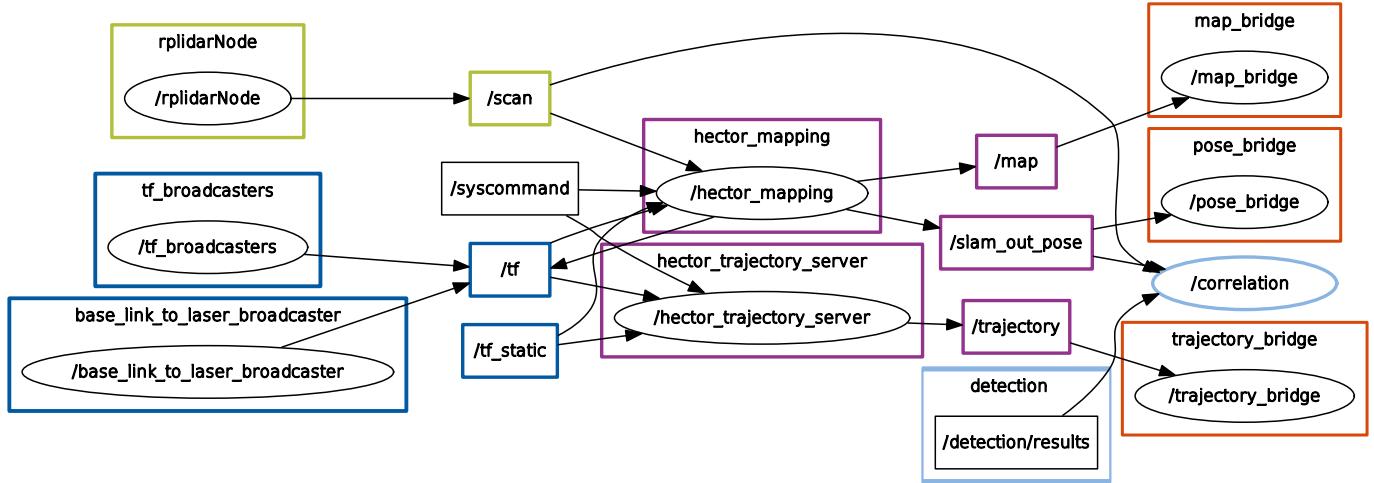


FIGURE 2.8 – Réseau ROS intégré au système

Le premier point, représenté en vert sur la figure 2.8, est assuré par un noeud 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 le constructeur du LIDAR 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 l'installation d'un driver permettant la conversion des données depuis un l'UART vers l'USB, ainsi que la définition de règles d'utilisation du port série sur lequel est raccordé le LIDAR. En sortie, ce noeud publie le topic `/scan`, correspondant au format de messages `sensor_msgs/LaserScan` précédemment évoqué.

L'intégration d'Hector SLAM demande la définition de plusieurs systèmes de coordonnées correspondant au divers composants du système. 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 noeuds et topics impliqués dans ce processus sont représentés en bleu foncé sur la figure 2.8. `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 qui subit ce dernier, en s'assurant qu'il ne sera jamais amené à changer. Dans le second cas, il faut généralement passer par la création d'un noeud publiant la valeur de ce décallage régulièrement durant l'exécution. Pour mieux saisir les enjeux soulevés par l'intégration d'Hector SLAM, la figure 2.9 donne les systèmes de coordonnées définis, ainsi que les noeuds implémentés (les *broadcaster*) responsables de la publication des transformations.

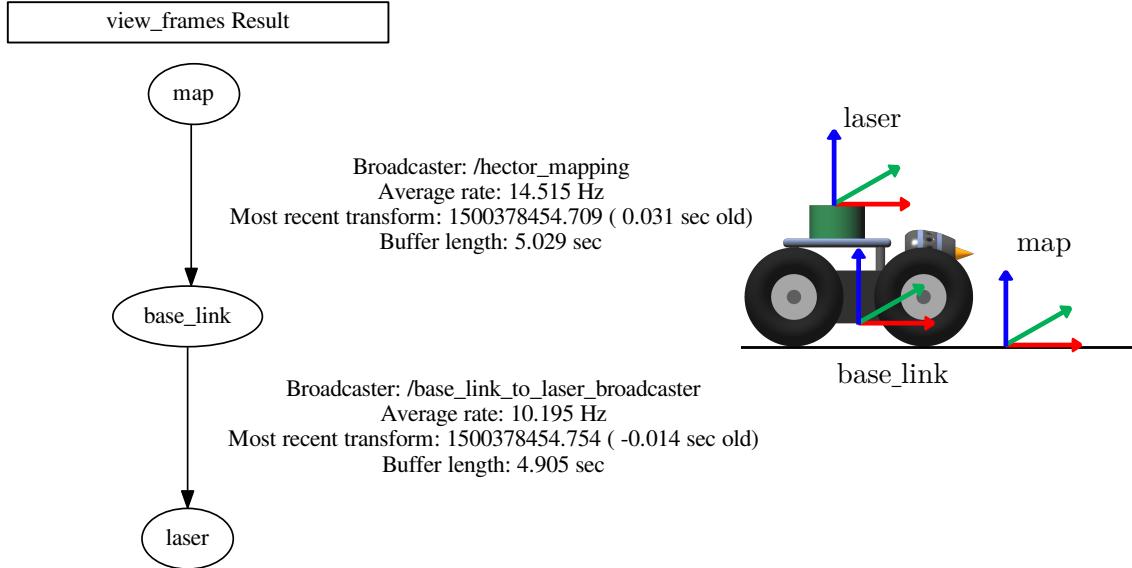


FIGURE 2.9 – Systèmes de coordonnées et transformations au sein du système

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 noeud */tf\_broadcaster* (voir figure 2.8) 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 2.8. Enfin, le noeud */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 standard<sup>8</sup> :

```

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 packages appelés *bridges* représentés en orange sur la figure 2.8. 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 2.10.



FIGURE 2.10 – 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 qu'il atteigne

8. En mode “debug” cette transformation est non avenue puisqu’elle est déjà comprise dans les données à rejouer, à savoir le bagfile

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 4000000 de valeurs entières, nous nous sommes focalisé sur l'émission de mises à jour locales à chaque acquisition, plutôt que globales<sup>9</sup>. Ainsi, le **payload** se construit de la manière 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& old_map,
4                               std::string *buffer ) {
5
6     std::string header, payload;
7
8     // Check data consistency
9     if( map.info.width != old_map.info.width ||
10        map.info.height != old_map.info.height ) {
11         std::cout << "Serialization error : Map and old_map with different
12             dimensions" << std::endl;
13         return -1;
14     }
15
16     // Fill payload with updated cases' value and index
17     for( int i = 0; i < map.info.width * map.info.height; ++i ) {
18         if( map.data[i] != old_map.data[i] )
19             payload.append( std::to_string(i) + ","
20                             + std::to_string( map.data[i] ) + "," );
21     }
22     payload.append( "EOP" );
23
24     // Fill header, return error if its size exceeds 20 bytes
25     header.append( MAP_ID );
26     header.append( "," + std::to_string( payload.length() ) + "," );
27     if( Serializer::addZeroPadding( &header ) < 0 ) return -1;
28
29     // Fill buffer with header and payload
30     buffer->append( header );
31     buffer->append( payload );
32     return 0;
33 }
```

Enfin, la figure 2.8 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 ou les faisceaux du LIDAR susceptibles d'avoir atteint l'objet. Lorsque ces-derniers existent<sup>10</sup> nous devons

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

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

transformer la distance et l'angle associés dans le système de coordonnées de la carte.

---

#### Algorithme 1 : Algorithme de calcul des résultats de corrélation

---

**Entrées :**  $R$  les derniers résultats de détection reçus sous la forme : dimensions plane de l'objet ( $w, h$ ) et ses coordonnées sphériques ( $\phi, \theta$ )  
 $t$  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 ( $\theta, r$ )  
 $pose$  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 ( $\theta, r$ ) 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
|   |   |   |    $r.x \leftarrow pose.p.x + matched.r \times \cos( matched.\theta + pose.yaw )$ 
|   |   |   |    $r.y \leftarrow pose.p.y + matched.r \times \sin( matched.\theta + pose.yaw )$ 
|   |   |   |    $r.z \leftarrow LIDAR\_LENGTH + matched.r \times \tan( \frac{\pi}{2} - r.\theta )$ 
|   |   |   |    $r.w \leftarrow 0.5 \times 2 \times matched.r \times \tan( \frac{r.w}{2} )$ 
|   |   |   |    $r.h \leftarrow 0.5 \times 2 \times matched.r \times \tan( \frac{r.h}{2} )$ 
|   |   |   
```

---

### 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 fourni également un mécanisme de communication inter-classes thread-safe et type-safe,

par le biais d'éléments appelés signaux et slots<sup>11</sup>. 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. Ce mécanisme est illustré dans l'exemple suivant, tiré du code source du projet à mettre en annexe plutôt....

```

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 slots
12     // 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 }
```

D'un point de vue formel, le fonctionnement de l'application est régi par une machine à état relativement simple, représentée figure 2.11.

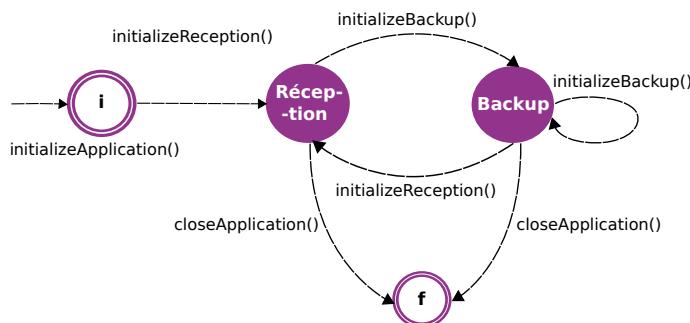


FIGURE 2.11 – Machine à état de l'application Qt

Ces états servent à gérer l'ensemble des 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 au 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.

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++

La figure 2.12 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.

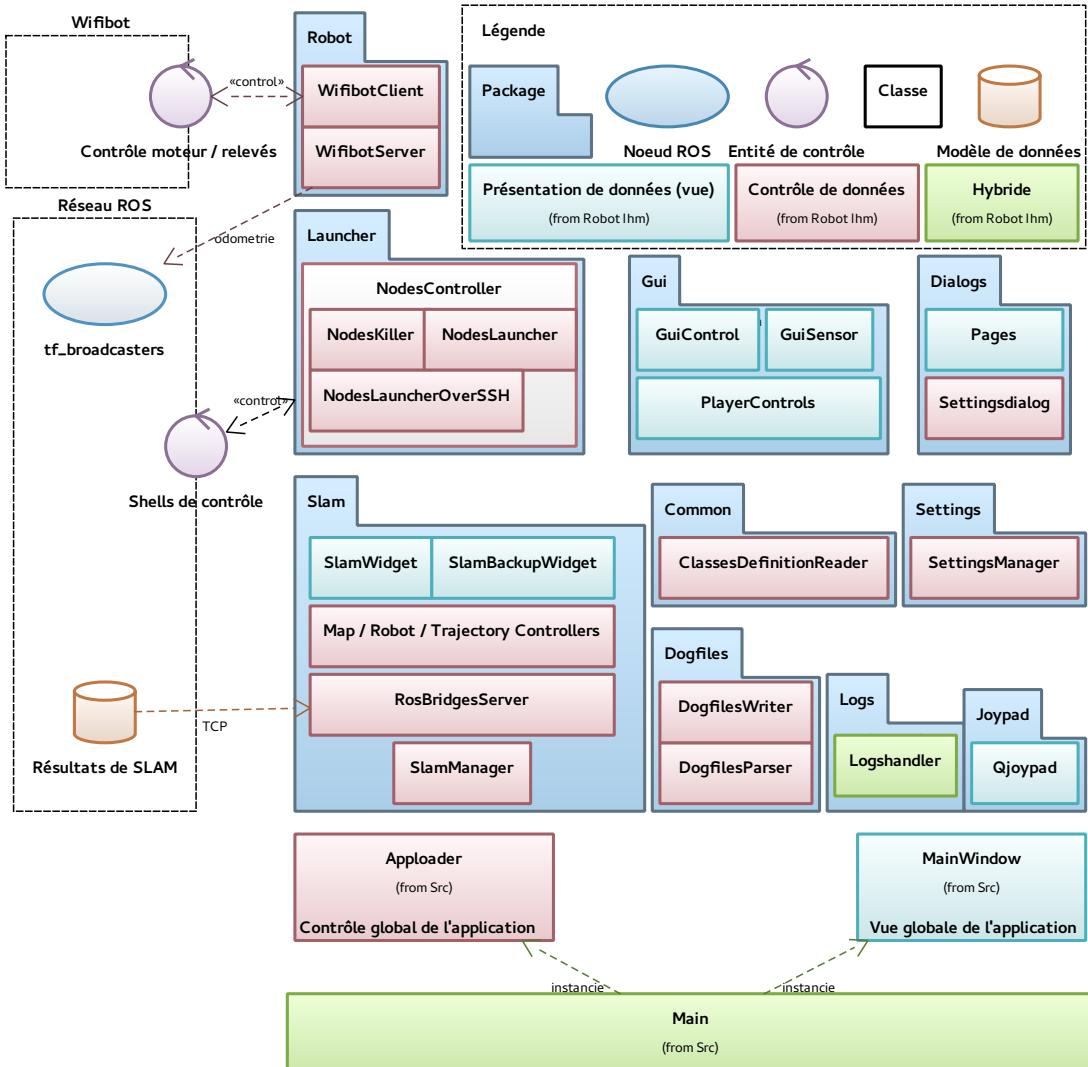


FIGURE 2.12 – Modélisation logicielle de l'Interface Homme-Machine

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 2.11 gérant ainsi le cycle de vie de toutes les autres 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 bridges 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 2.13 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.

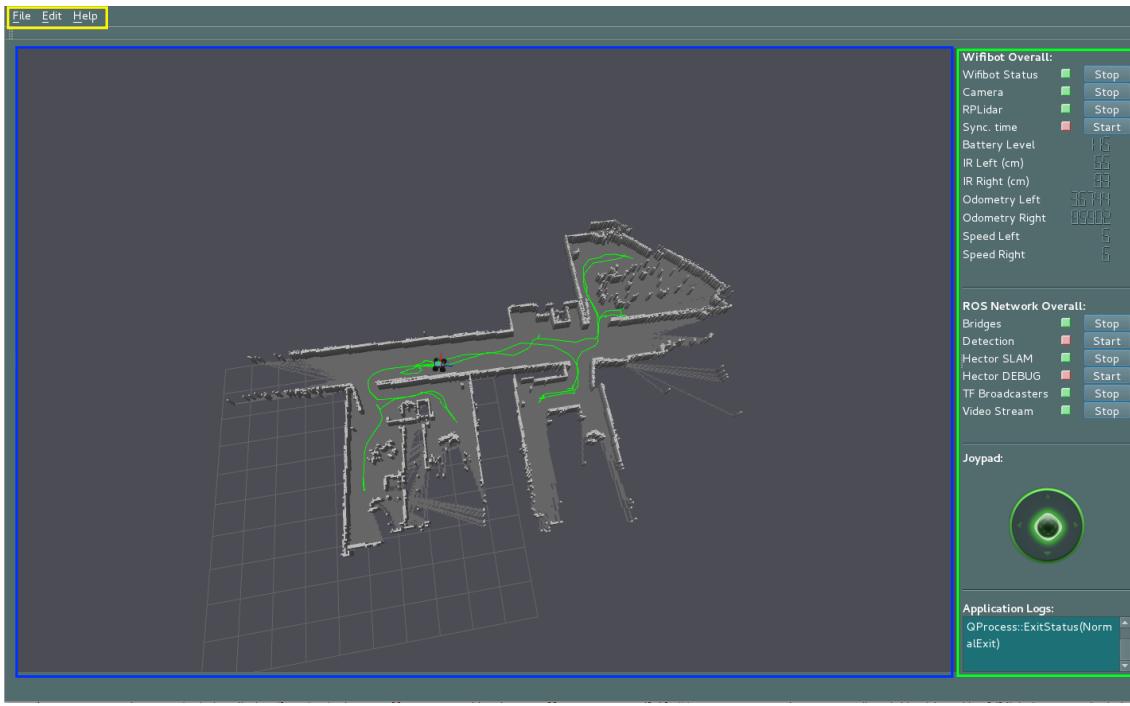


FIGURE 2.13 – Rendu visuel de l'IHM en mode Réception

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, fourni 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 2.14 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.

À cet effet, un format d'enregistrement des données propre à l'application a été défini en XML. Ce format, appelé *DOG*<sup>12</sup> est géré par les classes du package `DogFiles` : `DogFilesWriter` pour l'écriture et `DogFilesParser` pour la lecture. Des techniques de compressions ont été mises en œuvre 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 ));
```

12. Pour Detection and Occupancy Grid

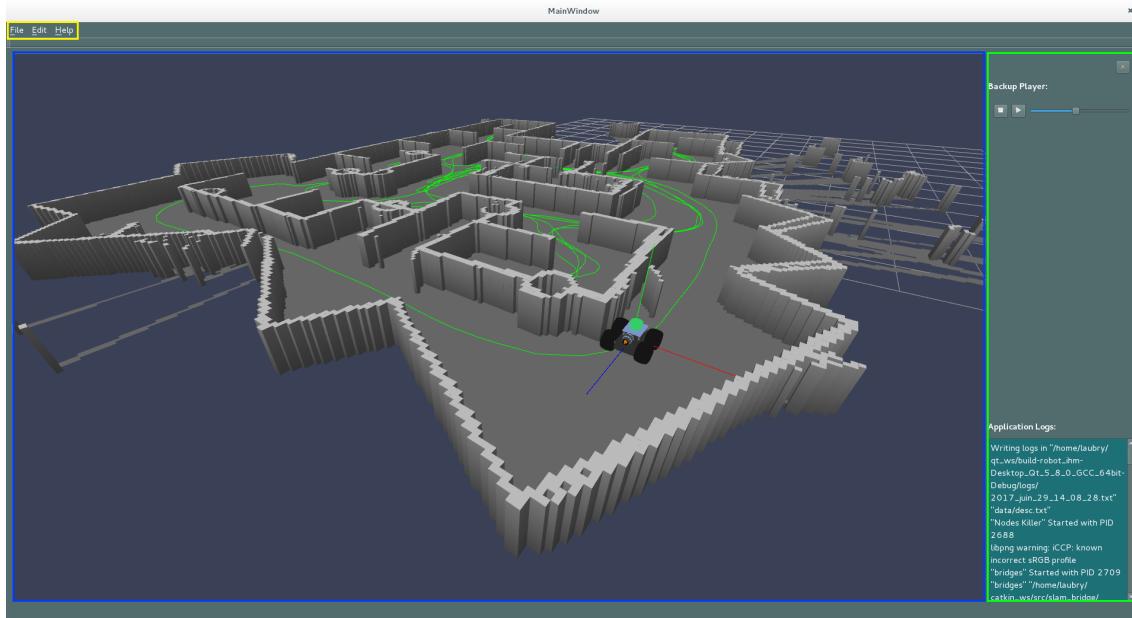


FIGURE 2.14 – Rendu visuel de l’IHM en mode rejeu

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

Gestion des logs

Lissage des murs

### 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 adaptation industrielle du projet. L’utilisation de ROS allant dans ce sens, il paraissait intéressant d’appliquer cet état d’esprit à l’IHM, tant dans l’architecture du logiciel que dans les fonctionnalités proposées à l’utilisateur final.

Le premier point se retrouve principalement dans l’implémentation de l’**Uploader** 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 noeuds peuvent eux-même 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 noeud
- le contrôle du réseau ROS depuis l’IHM

Qt offre un système de paramétrage des applications par le biais de fichiers texte et d’une classe **QSettings** indépendante de la plateforme utilisée<sup>13</sup>.

---

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

# Chapitre 3

## Bilan et perspectives

### 3.1 Organisation du travail

#### 3.1.1 Application du référetiel qualité interne

#### 3.1.2 Vers une conduite AGILE adaptée

### 3.2 Des résultats en vue d'un prolongement

### 3.3 Des résultats qui laissent envisager une suite

#### 3.3.1 SRT2M à la fin du stage

Résultats : présentation devant des collaborateurs commerciaux et directeurs de projet qui va se prolonger sur une présentation client au mois de septembre

Ce que l'on a mis en place pour que le projet se prolonge :

modularité, MAJ etc

potentiel d'apropriation du projet par de tierces personnes facilité : manuel utilisateur en interface web, troubleshooting, documentation automatisée

#### 3.3.2 Pourquoi prolonger le projet ?

### 3.4 Retour d'expérience

#### 3.4.1 Apports et difficultés du projet

L'adoption des normes de spécification et de conception du logiciel interne a constitué une phase intéressante et complexe à la fois. En effet, la partie Design et Conception de l'IHM, figure 2.2 témoigne d'a priori au sujet de la conception du logiciel. La DCC003 évoque des "modules" –à savoir des exécutables indépendants de l'IHM– dont l'état devrait apparaître dans un panneau de contrôle. Or, supposer de la présence de plusieurs exécutables en amont de la conception est un manquement aux préconisations de formalisation du besoin logiciel. Cette partie qui incombe habituellement au futur propriétaire du logiciel a été réalisée en même temps que la conception, d'avantage dans l'optique d'une découverte des normes de spécification que dans celle de réellement s'appuyer sur les documents produits.

Le travail s'en est trouvé d'une part simplifié, puisqu'il m'était facile de déchiffrer un document que j'avais moi-même écrit, et d'autre part complexifié sur le plan du formalisme des documents à produire. Cela a principalement engendré des écarts entre les documents de spécification et le code source assez importants, tant et si bien que les parties plus poussées de la documentation en sont rapidement devenues obsolètes.

Temps d'appropriation du sujet et du domaine qui constitue une découverte, tout autant que les outils utilisés (ROS) ou que les fondements théoriques découverts (SLAM).

### **3.4.2 Évolution personnelle au sein de la structure**

CDI, sécurité etc...

# 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](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](https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam blas_repo.pdf) Spring 2005
- [6] Cyrill 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](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](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](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](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>