



INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

DÉPARTEMENT DE GÉNIE ELECTRIQUE ET INFORMATIQUE

Projet de Fin d'Etudes

Spécialité : Automatique-Electronique

Filière : Systèmes Informatiques Embarqués Critiques

Conception logicielle pour robot de surveillance autonome Sur base mobile holonome

Auteur :

Laffargue - Lenny

Entreprise :

Elsys-Design

Référent INSA :

Le Corre - Gwendoline

Responsable du stage :

Dassieu-Blanchet - Nicolás

Année 2021-2022

Résumé

Ce rapport fait un bilan des travaux réalisés lors du développement d'un robot de surveillance autonome à roues omnidirectionnelles. Ce robot est développé par Elsys Design à Toulouse depuis 2 ans. Pour cette mission, une équipe de 3 stagiaires a été formée combinant plusieurs spécialités : FPGA, Hardware et Logiciel. Ce rapport fait le bilan de la partie **logicielle** développée pendant ces 6 mois.

Toutes les phases du projet ont été traitées, de la conception à la vérification. Le point de départ est un cahier des charges général. À partir de ce dernier, différentes problématiques ont été résolues telles que la cartographie, la localisation, la trajectoire ou encore la détection d'anomalie. Au niveau logiciel cela correspond aux différentes briques logicielles à intégrer au robot. Une solution a été développée pour chacune de ces briques afin d'obtenir un système complet répondant au mieux au cahier des charges initial.

Lors de ce stage de fin d'études, d'autres missions ont été réalisées afin d'aider au développement dont la conception de l'architecture mécanique du système et la mise en place d'un Linux embarqué. C'est sur ce Linux que le logiciel a été déployé et testé.

Remerciements

Je tiens à remercier en premier lieu mon maître de stage, Nicolás Dassieu-blanchet. Il a su se rendre disponible et rester à l'écoute. Ses conseils ont été très précieux sur l'intégralité du stage et m'ont permis d'accomplir au mieux le projet qui m'a été confié malgré les difficultés rencontrées.

Je remercie également Laurent Carpentier, responsable de l'agence Elsys Design Toulouse et Gabin Gaudré, Ingénieur d'affaires, pour m'avoir accueilli et intégré à l'équipe dans le cadre de ce stage très pertinent pour mon projet professionnel.

Je remercie aussi Tristan Cornière et Isabelle Van Leeuwen avec qui j'ai travaillé durant ces 6 mois. Nous avons pris le temps d'échanger, de se motiver et d'apprendre les uns des autres.

Je remercie toutes les personnes avec qui j'ai eu la chance d'échanger au sein d'Elsys Design. Ils m'ont fait découvrir plus précisément le métier de l'ingénieur en milieu professionnel. J'ai pu m'intégrer et apprendre de leurs expériences.

Enfin, je remercie le GEI de l'INSA Toulouse et toutes les personnes qui ont participé à ma formation. Ce sont cinq années d'apprentissage, d'encadrement, de soutien, de difficultés mais toujours dans la bienveillance et c'est ce qui m'a permis de réaliser ce stage dans d'aussi bonnes conditions. En particulier Gwendoline Le Corre, référente INSA pour ce stage, et Elodie Chanthery, responsable de la cinquième année SIEC.

Table des matières

1	Introduction	1
1.1	Contexte	1
1.2	Problématique	1
1.3	Le rapport	2
2	Cadre et objectifs du stage	3
2.1	Cadre du projet	3
2.2	Objectifs	4
2.3	Domaines de compétences	6
2.3.1	Compétences techniques	6
2.3.2	Gestion de Projet	7
3	Recherches et conception	9
3.1	Choix technologiques	9
3.1.1	Détection d'infractions	10
3.1.2	Localisation	10
3.1.2.1	Le BLE	12
3.1.2.2	Le RFID	12
3.1.3	Détection d'obstacles	13
3.1.4	Traitement de l'infraction	14
3.1.5	Carte de développement	14
3.2	Conception détaillée	16
3.2.1	Répartition des tâches entre partie logique et partie logicielle	16
3.2.2	Interface entre partie logique et partie logicielle	18
3.3	Analyse et prise de recul	20
4	Configuration de l'environnement	21
4.1	Linux embarqué	21
4.1.1	Petalinux	21
4.1.2	PYNQ	22
4.1.3	Un choix discutable	23

4.2	Environnement de programmation	24
4.2.1	Les outils utilisés	24
4.2.2	Architecture logiciel	24
4.2.3	Le mode manuel pour le debogage	25
5	Localisation	27
5.1	Le filtre de Kalman étendu	27
5.2	Comportement du filtre	31
5.3	Prise de recul	35
6	Navigation	37
6.1	La carte	37
6.2	La navigation	38
6.3	Analyse et prise de recul	40
7	Détection d'infractions	41
8	Blocs non-intégrés	43
8.1	IHM	43
8.2	Évitement d'obstacles	44
9	Architecture mécanique	45
9.1	Workflow	45
9.2	La structure	45
10	Conclusions et perspectives	47
Annexes		50
Présentation de la société		50
Roues omnidirectionnelles		52

Glossaire

AXI : Advenced external interface

BLE : Bluetooth low energy

EKF : Filtre de Kalman étendu

ESP32 : Série de microcontrôleur SoC

GEI : Génie électronique et informatique (Département de l'INSA de Toulouse)

IHM : Interface Homme-Machine

I2C : Inter-integrated circuit

IMU : Inertial measurement unit

IP : Intelectual processing core (bloc logique utilisé pour programmer le FPGA)

OS : Operating system

RFID : Radio frequency identification

ROS : Robot operating system

SIEC : Systèmes informatiques embarqués critiques. (Spécialité du GEI)

SLAM : Simultaneous localization and mapping

SoC : System on chip

UART : Universal asynchronous receiver-transmitter

Zybo-Z7-20 : Ordinateur de bord du robot

Chapitre 1

Introduction

1.1 Contexte

Le stage présenté ici a eu lieu dans les bureaux d'Elsys Design à Toulouse et s'inscrit dans le développement d'un robot de surveillance autonome à roues holonomes (Voir page 52). Il s'agit d'un projet interne à l'entreprise servant à la fois de formation et d'évaluation en vue d'une embauche au terme du stage. Le projet de stage pourrait également servir de vitrine technologique. Ce n'est donc pas une mission pour un client.

1.2 Problématique

Le robot de surveillance veut répondre à un besoin de sécurité pour les utilisateurs, qu'ils soient entreprises, entrepreneurs ou même particuliers. Des données sensibles sont souvent stockées dans les locaux et il est indispensable de les garder secrètes.

C'est le rôle donné à un gardien ou un agent de sécurité, solution naturelle et évidente aux problématiques de sécurité des sites dont voici les missions :

- Rester en alerte
- Contrôler en se déplaçant dans les lieux
- Signaler et réagir

Ces missions garantissent un niveau de protection acceptable. Engager un agent de sécurité n'est souvent pas envisageable économiquement. C'est là que ce sujet de stage intervient.

L'objectif est de mettre en place un système capable de détecter une anomalie, de contrôler et de réagir tout en se déplaçant de manière autonome dans les locaux.

Dans ce cadre, une équipe de 3 stagiaires a été formée : un ingénieur Software, un ingénieur FPGA et un ingénieur Hardware.

1.3 Le rapport

Pour commencer, ce rapport posera le contexte du stage, les objectifs, ainsi que les compétences nécessaires pour réaliser ces objectifs.

Le chapitre 3 détaillera la méthodologie employée pour préparer ce projet et toute la partie conception.

Le chapitre 4 présentera la configuration de l'environnement de développement dont le linux embarqué sur la carte.

Les chapitres 5 à 8 présentent les briques logicielles développées pendant le stage. Chaque chapitre correspond à une fonctionnalité répondant aux objectifs fixés : localisation, navigation, détection d'infractions et des briques logicielles qui n'ont finalement pas été intégrées.

Enfin, le chapitre 9 présentera l'architecture mécanique servant de support pour les cartes et les capteurs.

Chapitre 2

Cadre et objectifs du stage

2.1 Cadre du projet

Les stages chez Elsys Design sont utilisés pour intégrer les stagiaires, les former aux méthodologies de l'entreprise et les évaluer en vue d'une embauche post-stage. L'objectif n'est pas de réaliser un projet commercial mais plutôt d'**évaluer la méthode de travail**. C'est un détail qui a son importance dans la mesure où il a influencé certaines décisions sur le projet. Le stage prend place dans le cadre d'un projet interne qui pourra servir de vitrine technologique, il permettra à Elsys Design de présenter son savoir-faire aux nouveaux clients.

Le projet a été lancé en 2019 et est reconduit chaque année pour pousser le développement de plus en plus loin. En 2019, une alternance a pris place avec l'objectif de réaliser une base roulante pouvant être pilotée à distance avec une manette. Cette base roulante présente un châssis métallique et 4 roues omnidirectionnelles. La roue omnidirectionnelle présentée dans l'annexe 2. Une visualisation de la base roulante est disponible dans la figure 2.1.

Cette base roulante est contrôlée par une carte électronique personnalisée STM32F767. Elle dispose du bluetooth pour communiquer avec une manette sans-fil.



FIGURE 2.1 – Base roulante avec 4 roues omnidirectionnelles

En 2020, une équipe de 3 stagiaires a été formée pour réaliser une première version de ce projet : un robot de surveillance autonome. Des éléments ont été mis en place dont le choix d'une carte de développement Ultra96 de chez Avnet pour faire fonctionner les parties logique (PL) et logicielle (PS) du projet. La crise sanitaire a empêché les stagiaires de mettre en oeuvre leur solution. Le stage a eu lieu majoritairement en distanciel.

En 2021 le sujet a donc été reconduit avec une nouvelle équipe de 3 stagiaires. De nouveaux choix ont été faits sur le projet avec l'utilisation d'un Lidar et de ROS. Ces choix ont fait que le projet n'était pas aussi représentatif des compétences d'Elsys Design que voulu : l'utilisation de ROS fait perdre en performance et ne démontre pas suffisamment de compétences bas-niveau, les parties Logicielle et FPGA ne communiquaient que très peu entre elles. Le stage en soit est un succès si l'on s'en tient au cahier des charges mais ne convient finalement pas pour en faire un démonstrateur représentatif de l'entreprise.

L'entreprise a décidé de reconduire une fois de plus ce sujet de stage en 2022 en repartant de la base roulante, mais avec ces nouvelles contraintes : l'utilisation de ROS est interdite et un lien plus fort entre FPGA et logiciel afin d'exploiter réellement la partie logique. J'ai été recruté et intégré dans cette équipe de 3 stagiaires pour réaliser la partie logicielle de ce projet. L'équipe en charge du projet est encadrée par les 3 tuteurs de stages qui servent de référence sur toute la durée du projet : Sébastien Cuvillier, François Couturier et Nicolás Dassieu-Blanchet.

2.2 Objectifs

L'objectif est de développer ce robot de surveillance autonome. Pour réaliser ce projet, trois stagiaires ont été engagés par Elsys Design, responsables de toutes les phases du projet et répartis en trois corps de métier :

- Une développeuse FPGA, Isabelle Van Leeuwen, responsable de la programmation de la partie logique de la carte. Elle définit l'architecture et réalise des accélérations du logiciel (Conception, Code, Test, Verification, Validation)
- Un concepteur hardware, Tristan Cornière, responsable de la réalisation de la carte électronique servant d'interface entre la carte de développement et les capteurs/actionneurs. (Vérification des compatibilités, des niveaux de tension, gestion de l'autonomie, placement, routage)
- Un développeur Software, moi-même, responsable de l'implémentation du contrôle du robot. Il doit respecter la contrainte de non-utilisation de ROS afin d'avoir un contrôle à la fois très bas-niveau en traitant des données proches de la sortie capteur, et haut-niveau avec le système de décision du robot.

Précisons maintenant le cahier des charges technique du robot :

- Le système doit être capable de détecter une intrusion dans les locaux.
- Le robot doit pouvoir se déplacer en autonomie dans les locaux
- Le robot doit pouvoir détecter et éviter les obstacles rencontrés sur le chemin
- Le robot doit pouvoir traiter l'intrusion

Une base roulante possédant des roues omnidirectionnelles 2.1 est à disposition pour la réalisation de ce robot. Celle-ci est capable recevoir une commande de vitesse de déplacement en UART selon 3 axes : Vitesses de translation $V_{x'}$, $V_{y'}$ du robot et vitesse de rotation $V_{z'}$. Le projet de l'année précédente est mis à disposition (Carte de développement, Carte électronique d'interface, FPGA et logiciel). Il est autorisé de récupérer certaines parties du projet tant que les nouvelles contraintes restent respectées.

Le cahier des charges est très général et peu précis car une grande liberté a été laissé pour développer le robot. Le choix des solutions revient exclusivement à l'équipe. Avec les éléments cités, il est déjà possible de cerner les premières problématiques de réflexion. Ceci correspond à la première phase du projet : à partir de cahier des charges, cerner les besoins. Voici les différentes problématiques posées par le cahier des charges.

1. Tout d'abord il faut que le robot puisse **détecter une intrusion**. Ceci doit être fait à tout moment dans toutes les pièces des locaux. Ceci signifie que le système ne prend pas en compte uniquement le robot en lui-même. Des éléments fixes doivent être présents dans chaque pièce des locaux. Ils doivent être capables de détecter des anomalies (intrusions) et de communiquer avec le robot. L'année précédente, des ESP32 connectés à des détecteurs de mouvements avaient été utilisés. Ces derniers communiquaient avec le robot via WiFi. Cette solution est tout a fait réutilisable.
2. Ensuite, le robot doit être capable de **se déplacer en autonomie**. Ceci implique plusieurs problématiques à la fois logicielles et hardware.
 - (a) Il faut que le robot sache dans quelle direction aller, il doit donc savoir où il se trouve. Cette première problématique correspond à la brique **localisation** pour laquelle il a été nécessaire de trouver une solution hardware et logicielle. Cette question avait été traitée via un Lidar et un filtre de Kalman sous ROS l'année précédente. Il n'est pas possible de réutiliser ce système. Une nouvelle solution a été intégralement développée. La base roulante disposait de capteurs intégrés sur lesquels il est possible de s'appuyer. Elle disposait d'encodeurs pour générer des données odométriques.
 - (b) La seconde problématique est la question de la **génération de trajectoire** : quel chemin suivre ?
 - (c) La troisième et dernière problématique concerne le **suivi de trajectoire**.

3. Aussi, la **détection d'obstacles** nécessite une réflexion poussée pour ne pas surcharger le robot, surcharger notre planning et sortir du budget.
4. Enfin, la question du **traitement de l'infraction** a naturellement mené à penser à une caméra capable d'enregistrer la scène via une vidéo ou une série de photos. Cette idée avait monopolisé une grande partie du temps de stage du développeur FPGA en 2021 ce qui n'était pas souhaité sur cette nouvelle version du projet.

Lors du stage 2021, les points 2)a), 2)b), 2)c) et 3) avaient été traités par des noeuds ROS. Cette année, une nouvelle solution bas-niveau été intégralement développée.

D'un point de vue logiciel, chaque problématique correspond à une tâche distincte. Ce sont les principales briques logicielles. La mission est de développer ces briques logicielles. À celles-ci ont été ajoutées d'autres briques mineures dont des briques servant au débogage : un moyen de **commander manuellement** le robot ainsi qu'une **interface web Homme-Machine** pour afficher des informations.

À cela s'ajoute un dernier objectif : la base roulante n'offrait pas la possibilité de fixer facilement tous les éléments hardware à ajouter au système. La **fabrication de la structure mécanique** qui se fixe sur le robot a donc été une autre mission à réaliser au cours de ce stage. Cette structure a ensuite servi à placer les cartes, les capteurs, la batterie et tout autre élément physique nécessaire au fonctionnement du robot. Une imprimante 3D était à disposition pour la réalisation de cette structure.

Un nouvel élément a aussi été ajouté au système dans le but de le rendre entièrement autonome : une **station de charge**. C'est un ajout qui concerne principalement le concepteur hardware mais a tout de même eu une influence sur la manière de penser certaines briques logicielles.

2.3 Domaines de compétences

2.3.1 Compétences techniques

Le développement de la partie logicielle lors de ce stage a fait appel à des compétences déjà rencontrées notamment lors du projet de SIEC en 5ème année. Ce projet consistait à développer le software d'une voiture capable de détecter et d'éviter un obstacle sur sa trajectoire de manière autonome. L'architecture déployée lors du stage se rapproche de l'architecture du projet SIEC. Une carte de contrôle bas-niveau basée sur un STM32 permet de contrôler le robot. Cette carte communique avec une autre carte "ordinateur de bord" servant à la fusion des données et à la prise de décision.

Plusieurs capteurs sont également présents dans les deux projets : des ultrasons et une IMU.

En souhaitant reprendre le fonctionnement général manipulé lors du projet SIEC, il a été choisi d'embarquer un système d'exploitation basé sur Linux dans l'ordinateur de bord, où il est possible de programmer en python. Ce choix a été motivé par la nécessité de gagner du temps sur la mise en place d'un maximum de briques logicielles.

La contrainte sur le middleware ROS ne s'est pas révélée handicapante puisque ROS n'avait pas été utilisé dans le projet académique de SIEC. Tout avait été programmé à la main en python sur l'ordinateur de bord.

Concernant les domaines scientifiques, la mise en place du linux embarqué a rappelé les notions de systèmes d'exploitation pour les systèmes embarqués dont l'enseignement a été reçu lors de la 5ème année. Le calcul de la localisation avec la fusion des données capteurs a nécessité la mise en place d'un filtre de Kalman étendu (EKF). Pour la navigation et l'évitement d'obstacles les travaux effectués lors du projet de SIEC ont servi de source d'inspiration. Enfin, les enseignements de temps réel ont été très bénéfique au cours de ce stage, dont des notions de multithreading avec l'utilisation de mutex ont également été très utiles.

2.3.2 Gestion de Projet

Pour ce projet, l'équipe était en charge de l'intégralité du cycle en V. Le budget devait également être géré et pris compte. Un planning a été mis en place et des réunions périodiques avec les responsables ont contribué au suivi de l'évolution du projet. Les étapes par lesquelles nous sommes passées :

- Rédaction des spécifications à partir du cahier des charges
- Établissement d'un planning
- Choix technologiques
- Conception
- Développement
- Intégration

Les parties tests et validation n'ont pas encore eu lieu à la date d'écriture de ce rapport. C'est pourquoi elle ne sont pas précisées ici.

Afin d'avancer de manière optimale tout en s'assurant de conserver une bonne communication tout au long du stage, un point quotidien entre les membres de l'équipe a été mis en place. Ces points ont permis à chacun de correctement situer son travail par rapport au projet dans la globalité et d'avancer chacun de son côté tout restant informé de ce que faisait le reste de l'équipe. Un github commun a été créé où chacun gérait son dossier de manière indépendante. Diagrams.net a été utilisé pour centraliser tous les schémas.

Les parties hardware, FPGA et software étant individuelles, j'ai établi ma propre organisation au sein du projet github : un second repository dédié à la partie logicielle a été créé, puis ajouté en submodule dans le github général du projet. Cette organisation a permis d'avoir une grande liberté dans la gestion des branches et des commit, tout en prenant soin de mettre à jour le submodule quand une nouvelle version fonctionnelle du logiciel était disponible.

Des réunions ont eu lieu toutes les 2 semaines environ permettant de présenter les avancées du projet, demander du soutien sur certains points et plus généralement d'échanger sur le projet avec l'équipe encadrant le stage.

Un planning sous forme de Gantt a été réalisé à l'aide de GanttProject. Le planning a été détaillé pour chaque brique. À partir de ce Gantt, des jalons ont été fixés et un chemin critique a été identifié. Ce dernier correspondait au chemin de la carte interface entre l'ordinateur de bord et les capteurs/actionneurs du robot.

Le planning était très chargé pour la réalisation de ce projet. Les briques d'évitement d'obstacles et de traitement de l'infraction ont été catégorisées comme optionnelles dès le début du projet. Elles auraient pu être développées si aucune difficulté n'avait été rencontrée.

Enfin, un tableau de commandes était régulièrement mis à jour afin de gérer le budget, vérifier la bonne réception des commandes et surveiller le temps de livraison de chaque composant.

Chapitre 3

Recherches et conception

La première réalisation de ce rapport est la phase de conception du projet. Un mois a été nécessaire. Cette phase a nécessité beaucoup de communication, de schémas à l'aide de diagrams.net, de tableaux à l'aide d'Excel et de recherches scientifiques. Ce chapitre présente les versions finales des schémas et les principaux points de réflexion. Le projet est présenté dans ce chapitre comme il avait été imaginé au départ. Il a ensuite largement évolué au fil du développement mais ceci sera traité dans les autres chapitres.

3.1 Choix technologiques

L'analyse des besoins et les spécifications ont conduit à lister les différentes options technologiques envisageables. Suite aux analyses 2.2, les technologies à déterminer ont été les suivantes :

- La détection d'infractions
- La localisation
- La détection d'obstacles
- Le traitement de l'infraction

Afin de choisir nos technologies, une décision importante pour le projet était nécessaire : le rôle du FPGA. Ce dernier avait 2 possibilités : faire du délestage logiciel, ou bien ne s'occuper que du traitement d'image avec une caméra. Cette deuxième option est intéressante car elle répond à la question du traitement de l'infraction. Cependant cela signifiait qu'aucune autre tâche ne pouvait être réalisée par le FPGA au vu du temps nécessaire pour le traitement d'image. C'est donc la première option qui a été choisie. Cette décision a été prise avec l'accord les personnes responsables du projet. La vidéo a été abandonnée pour mieux se concentrer sur les autres parties du projet. Le FPGA a eu pour rôle de **délester la partie logicielle**. Du point de vue de l'apprentissage, cette décision a permis de créer une réelle collaboration entre FPGA et logiciel car elle a impliqué beaucoup de communication et d'organisation sur toute la durée du stage.

3.1.1 Détection d'infractions

Comme expliqué précédemment, un système avait été mis en place en 2021 pour la détection d'infraction. Il s'agissait d'un réseau de balises branchées sur batteries ou sur secteur et composées d'un détecteur de mouvements connecté à un ESP32. Le robot embarquait également un de ces ESP32 pour réceptionner les signaux des balises. Elles étaient toutes inter-connectées par Wifi Mesh. Il s'agit là de la théorie, dans les faits ce n'était pas réellement du Wifi Mesh qui a été mis en place puisque les balises ne se relayaient pas. Elles étaient toutes directement connectées à la balise réceptrice sur le robot.

Cette technologie a été conservée pour le projet, permettant d'utiliser des balises déjà partiellement fonctionnelles, qu'il a ensuite fallu légèrement modifier pour les intégrer au projet. Le principal défaut de ces balises est leur manque de capteurs différents et des détecteurs de mouvements trop sensibles. Des recherches ont été faites en vue d'améliorer ces balises avec des capteurs de taux de CO₂ en cas d'incendie par exemple. Étant donné qu'il fallait environ 10 balises pour une couverture optimale des 7 pièces, le rajout d'un seul capteur impliquait des coûts importants. Une balise coûtait 20€ avec l'ESP32 et le détecteur de mouvement, soit 200€ pour les 10 balises. Un capteur de CO₂ coûte au minimum 60€ soit une augmentation de 600€. Des détecteurs de mouvement plus fiables entraînait une augmentation de 100€ au minimum. Tout autre capteur amenait une augmentation de plus de 100€. Les mêmes balises ont donc été choisies sans modification hardware. Seules quelques retouches logicielles ont été nécessaires.

3.1.2 Localisation

Le problème de la localisation a nécessité de nombreuses recherches, de sources et de documents. L'idée initiale était de faire une cartographie des locaux en autonomie, puis le calcul de la localisation à partir de la carte générée. Ce sont des algorithmes SLAM (Simultaneous localization and mapping). Plusieurs théories sur l'algorithme de Fast SLAM ont été étudiées [6]. Il s'agit de l'option la moins gourmande en ressources parmi les méthodes de SLAM. Cependant les algorithmes trouvés en lien avec le Fast SLAM se basaient presque tous sur ROS, sur du lidar ou/et sur une camera avec de la reconnaissance d'images. Tout ceci semble ne pas correspondre à notre projet. Seul le lidar aurait pu être utilisable, mais il a été décidé de faire sans cette technologie complexe à manipuler.

La complexité de la mise en place d'une solution faite main semble aussi peu envisageable du fait de la contrainte de temps. En discutant avec les responsables du stage, il s'est avéré préférable de penser à un système de cartographie pré-programmé en dur. Ceci

a permis de se concentrer sur le calcul de la localisation uniquement.

Le système retenu est un système où le robot calcule sa localisation en continu avec des données de localisation relatives. Ces données peuvent présenter une déviation importante dans le temps, c'est pourquoi des données de localisation absolue ont été ajoutées. Ces données doivent être précises et doivent servir à corriger le calcul de localisation relative avant que la déviation devienne trop importante.

Pour le calcul de la localisation relative, le système composé d'une IMU et des données odométriques a été repris. Ce système avait été déployé lors du stage 2021 mais peu exploité. Comme les données étaient traitées par ROS, un tout nouveau modèle de traitement des données a été conçu. L'IMU est le modèle à 9 degrés de liberté de Seeed studio : Grove - IMU 9DOF(lcm20600+AK09918). Il embarque un accéléromètre, un gyroscope et un magnétomètre ce qui permet d'obtenir l'accélération selon 3 axes, la vitesse angulaire de ces 3 mêmes axes et enfin le champ magnétique. Les données magnétiques permettent d'obtenir l'orientation du robot lors de l'initialisation.

Pour fusionner les données de ces deux capteurs, la solution envisagée est un filtre de Kalman étendu. Cette méthode avait été découverte et testée lors des enseignements de 5ème année à l'INSA. Elle a donc été plus simple à déployer qu'une autre méthode inconnue.

Pour la localisation absolue, toutes les possibilités technologiques ont été listées sous forme de tableau comparatif 3.1. Avant la décision sur le rôle du FPGA, l'utilisation de QR Code ou d'amers était une possibilité. De même, le lidar était une option mais nécessitait l'enregistrement des données en tout point, algorithme complexe et avec peu de ressources sur lesquelles s'appuyer dans la communauté. Il restait alors comme possibilités envisageables le bluetooth low energy (BLE) ou le RFID passif ou actif.

Technologie	LOCALISATION						
	Localisation Absolue						
	RFID		BLE		Lidar	Camera	
	Passif	Actif	Amplitude	Temps de propagation	SLAM	Amer	QRCode
Précision	+	+	--	--	++	-	-
Prix	50€ max	~150€		320 €	320 €	0 €	60 €
Temps de mise en œuvre	+	+	+	+	--	--	--
Performance	++	+	+	+	+/-	-	-

FIGURE 3.1 – Tableau comparatif des technologies pour la localisation absolue

3.1.2.1 Le BLE

La localisation avec le BLE par temps de propagation est en réalité une sorte de triangulation : au moins 3 balises sont connectées au robot. Le robot envoie un signal à une balise à un instant t puis attend la réponse de la balise. Lorsqu'il reçoit la réponse à $t + dt$, le robot sait combien de temps il a fallu au signal pour se propager. Il peut en déduire la distance de la balise. Si la position de la balise est connue, il peut en déduire la position du robot. Le BLE par amplitude ne se base pas sur le temps mais le concept est similaire. Il compare l'amplitude du signal à l'arrivée par rapport à l'amplitude au départ et est capable d'en déduire à quelle distance est la balise car l'amplitude diminue avec la distance.

Cette solution a été envisagée mais avec des recherches plus approfondies, deux éléments ont mené à son abandon : la précision est en fait très mauvaise sur cette technologie quand elle est déployée en intérieur (dans des bureaux par exemple). Le signal a de forte chances de traverser des murs ou de rencontrer des éléments électroniques qui vont fausser le signal à l'arrivée. Le décalage obtenu peut être de plusieurs mètres.

Le second élément est le coût. En effet, mettre en place de nouvelles balises BLE en plus des balises de détection implique un coût supplémentaire. Un coût énergétique d'abord puisqu'il est nécessaire d'amener de l'énergie supplémentaire pour chaque balise, mais surtout un coût financier important. La portée du BLE est d'environ 10 mètres ce qui ne couvre pas tous les locaux. Ceci signifie qu'il est nécessaire de mettre plus de 3 balises et donc d'augmenter encore le coût de ces balises.

3.1.2.2 Le RFID

L'option du BLE étant écartée, il ne restait que les RFID. Le fonctionnement serait le suivant : disposer des étiquettes RFID au sol à des points de passages spécifiques dans les locaux. Le robot embarque un lecteur RFID sous sa carrosserie. Le robot démarre d'un point de passage et connaît donc sa localisation initiale grâce à l'étiquette qui s'y trouve. Il se déplace en contrôlant sa trajectoire vers le point de passage suivant. Lorsqu'il l'atteint, une étiquette est détectée et permet de mettre à jour la localisation calculée à partir de l'IMU et de l'odométrie.

Pour préciser le fonctionnement, les points de passages seraient des lignes d'étiquettes RFID se trouvant sur le chemin du robot. Quand une des étiquettes de la ligne est détectée, le robot sait où il se trouve précisément et peut recentrer sa position dans le cas où il détecte une étiquette en bout de ligne. La figure 3.2 illustre la disposition des étiquettes. L'avantage principal d'une telle solution est la précision obtenue par rapport au coût. La portée pouvant être de 2cm, l'erreur sur la position lue par le robot serait faible.

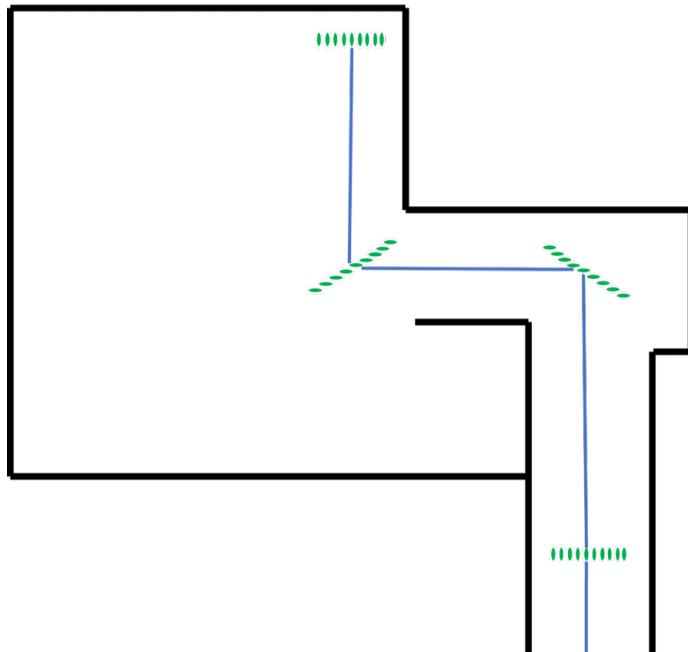


FIGURE 3.2 – Disposition des étiquette RFID sur la trajectoire du robot

Dans cette disposition, le RFID actif n'a pas d'avantages par rapport au RFID passif. En effet, il nécessite de continuellement alimenter les étiquettes disposées au sol. C'est un coût énergétique et financier supplémentaire qui n'a pas d'atouts dans ce contexte d'utilisation.

La solution qui a été retenue est donc le RFID passif. Le robot embarque un Lecteur RFID Click de Mikroe. Les étiquettes sont des étiquettes type 2. Elles suivent la norme ISO 14443A et fonctionnent à 13.56 MHz accessibles à la fois en écriture en lecture. Ceci permettait d'écrire directement la position dans les étiquettes, et donc de les rendre toutes uniques et différenciables

3.1.3 Détection d'obstacles

Pour être capable de détecter des obstacles, plusieurs options étaient disponibles. Les plus importantes figurent dans le tableau 3.3.

Le plus évident était l'utilisation d'ultrasons. La base roulante du robot en intégrait déjà 4 (un par face). Le stage 2021 avait préféré un lidar car les ultrasons étaient complexes à utiliser et présentaient plusieurs problèmes d'interférences, en plus de ne couvrir que partiellement chaque face du robot.

L'idée suivante a été d'utiliser des capteurs laser mais là encore la zone couverte est très faible et il en aurait fallu un nombre conséquent par face pour une détection complète, faisant monter rapidement le coût. La caméra et le lidar ont tous deux déjà été écartés.

Technologie	Détection d'obstacles				
	Ultrasons		Lidar	Infrared proximity sensor	Camera
Déjà présents	3-5 nvc ultrasons : HCSR04				Machine learning
Précision	-	+	++	+	+
Prix	0 €	10-20 €	0 €	10-30 €	
Temps de mise en œuvre	+	++	-	+	--
Performance	++	++	+	++	--
FPGA/Soft	?	FPGA	Soft	?	FPGA

FIGURE 3.3 – Tableau comparatif des technologies pour la détection d'obstacles

C'est finalement une dernière proposition consistant à déployer de nouveaux ultrasons qui a été retenue. L'avantage est la liberté de le positionnement des ultrasons sur le robot. Selon cette idée 5 ultrasons ont été intégrés au projet dans un premier temps, positionnés de manière à couvrir les parties avant du robot : Avant, avant-gauche, avant-droit, droit et gauche. Il s'agit des parties les plus importantes à couvrir. Une couverture à 360 degrés est une idée d'amélioration du projet.

Un autre avantage de cette option est leur mode de communication : Un signal nommé *TRIG* qui active l'ultrason et un signal *ECHO* qui sert de réponse. Le calcul de la distance est effectué du côté de la carte de développement en calculant le temps entre l'envoi du signal *TRIG* et la réponse *ECHO*. Ce mode de fonctionnement permet de s'assurer de ne pas avoir d'interférences entre les capteurs puisqu'il est possible de faire fonctionner les ultrasons les uns après les autres et non tous en même temps. Cette gestion du temps nécessite cependant une grande précision, cette tâche a donc été attribuée au FPGA.

Il en découle un autre avantage : la réduction du nombre de pins nécessaire sur la carte de développement. En connectant tous les ultrasons, près de 10 pins auraient été réservés. Comme les ultrasons fonctionnent à tour de rôle, un MUX/DEMUX a été mis en place (implémenté sur la carte interface du concepteur hardware) réduisant la gestion des ultrasons sur 5 pins : TRIG, ECHO et 3 pins pour la sélection de l'ultrason à activer.

3.1.4 Traitement de l'infraction

Comme cette brique a été placée comme optionnelle sur le projet, un simple module GSM a été choisi pour envoyer un signal d'alerte au propriétaire des locaux. Il ne sera pas implémenté sur le robot mais il a été pris en compte pour la suite de la conception.

3.1.5 Carte de développement

Il ne manquait plus que la pièce maîtresse du projet, la carte de développement. Un schéma général regroupant tous les capteurs a d'abord été réalisé. Il précise les entrées/sorties et les modes de communications. (Figure 3.4)

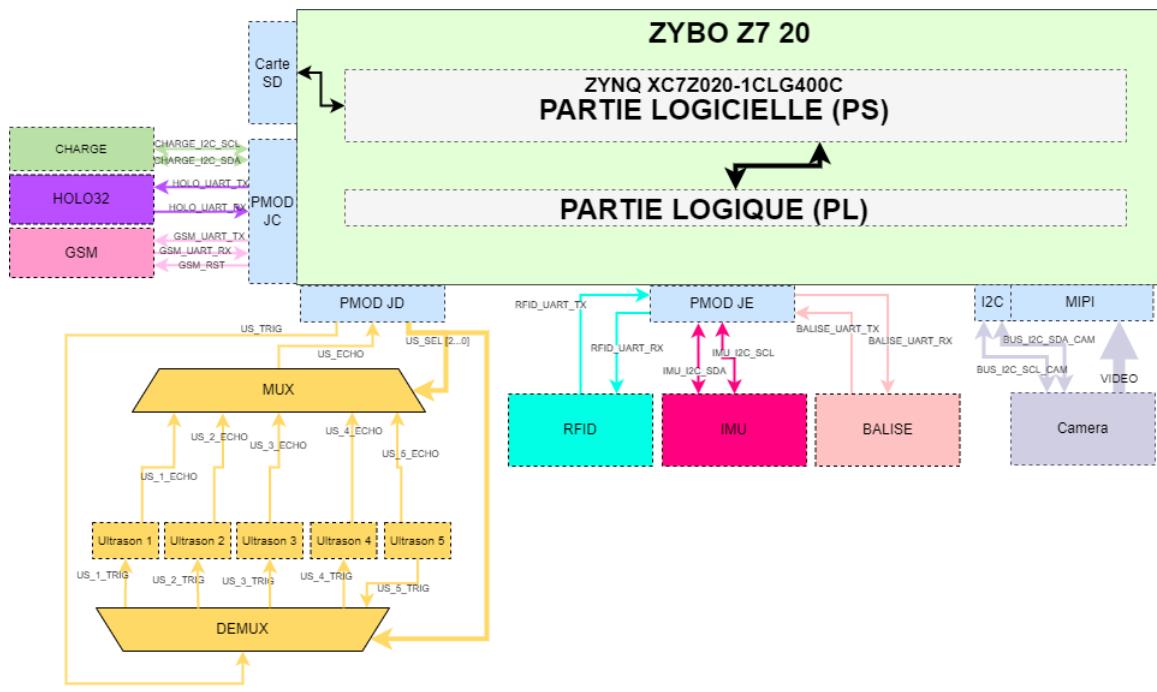


FIGURE 3.4 – Conception générale du système

Le cadre vert représente la carte de développement. Elle est composée de la partie logique (PL) et de la partie logicielle (PS) du projet. Ce qui est important ici est l'extérieur du cadre vert qui a été géré par le concepteur hardware. Il s'est occupé d'acheminer tous les signaux de capteurs vers la carte de développement. Profitons-en pour rappeler la fonction de chaque module, et les capteurs associés si il y en a :

- HOLO32 : Base roulante du robot. Des ordres de vitesses lui sont envoyés. Elle répond avec des données odométriques.
- IMU : Couplée avec l'odométrie précédemment citée pour calculer une localisation relative.
- RFID : Permet de lire des étiquettes RFID qui donnent une localisation absolue précise au robot. Cet élément permet de corriger les dérives de la localisation relative à intervalle régulier afin de ne pas diverger de la trajectoire.
- CHARGE : Données sur la batterie (batterie faible, charge en cours)
- Ultrasons : Servent à la détection d'obstacles
- Camera et GSM : Ne sont pas implémentés dans le systèmes mais sont prévus en cas de volonté d'améliorer le projet.

Après avoir listé toutes les entrées et sorties nécessaires, il a été déterminé que la carte de développement devait pouvoir accueillir 18 GPIO au minimum. Afin de laisser la liberté d'une évolution du projet, un port MIPI et un port USB ont fait partie des spécificités prises en compte pour le choix la carte. Là encore plusieurs options ont été envisagées. La carte du projet 2021 était l'Ultra96-V2. Cette carte offre beaucoup de GPIO à la fois haute et basse vitesse. Elle possède une grande communauté sur laquelle s'appuyer pour débloquer des situations et avancer sur le projet. L'inconvénient de cette carte est qu'un seul modèle était disponible. Avec la pénurie de composants, le délai de livraison de ce modèle était annoncé à 1 an. Si un problème était intervenu sur la carte il aurait fallu en choisir une autre et réadapter tout le projet pour cette nouvelle carte. Ce choix présente un facteur de risque beaucoup trop important, c'est pourquoi une autre carte plus disponible a été préférée.

Le modèle choisi est équivalent en puissance de calcul et est plus accessible : la Zybo-Z7-20 de Digilent. Cette carte embarque un Zynq 7000 qui comporte un processeur ARM et un FPGA series-7 de Xilinx. Elle possède 32 GPIO sous forme de PMOD, un connecteur MIPI et un port USB. Le support est très réactif et le Github de Digilent a semblé suffisamment fourni pour permettre d'avancer aisément. Le site propose de nombreux tutoriels pour la prise en main des différents éléments. De plus, la Zybo propose d'autres ports dans le cas d'une évolution possible du projet : des entrées et sorties audio, HDMI, un port Ethernet, plusieurs LED, des interrupteurs et des boutons. Le délai de livraison était inférieur à une semaine.

3.2 Conception détaillée

3.2.1 Répartition des tâches entre partie logique et partie logicielle

Avec tous les choix technologiques effectués, la conception détaillée du système a pu débuter. Les différentes briques logicielles à mettre en place sont les suivantes, par ordre de priorité :

- GESTION UART (Déplacement du robot) : gestion de la communication avec la base roulante et envoi d'ordres de déplacement
- GESTIONNAIRE : gère le passage entre MODE AUTO et MODE MANUEL
- LOCALISATION : fusion de données capteurs odométrie, IMU et RFID
- NAVIGATION : recherche de trajectoire et suivi de trajectoire.
- DETECTION : vérification des données d'infractions, d'obstacles, de batterie
- IHM : pour l'envoi de données visuelles.

Tout d'abord, les tâches ont été réparties entre la partie FPGA et la partie logicielle. Le FPGA a eu pour rôle de pré-traiter la majorité des données capteur du système. Les besoins pour chaque donnée ont été précisés. Certaines fonctions du logiciel ont également été confiées au FPGA. Toutes les informations ont été regroupées dans le schéma 3.5. Voici les différents blocs du FPGA par ordre de priorités :

- Commandes moteurs et l'odomètre : pour rappel, il s'agit de la communication avec la base roulante en UART. Aucun traitement n'est nécessaire sur les données donc il n'est pas utile de passer par le FPGA à part pour acheminer les signaux jusqu'au logiciel.
- IMU : la communication passe par un bus I2C. Plusieurs registres doivent être lus pour obtenir les 9 valeurs bruts. Une initialisation de l'IMU consistant à faire un 360 degrés au robot est nécessaire pour en déduire une erreur systématique. Cette initialisation doit permettre de compenser l'erreur systématique sur les données utiles.
Il a été décidé ici que le FPGA serait chargé de la gestion de la lecture des registres I2C.
- RFID : le lecteur RFID nécessite une demande de recherche de tag en continue. Côté logiciel, cela signifie un bouclage. Comme il est essentiel de ne pas passer au dessus d'un tag sans le lire, il est préférable que ce soit le FPGA qui en soit chargé. C'est donc ce qui a été décidé.
- Dijkstra : pour le calcul d'un chemin, nous avons opté pour l'algorithme de Dijkstra. Ce dernier agit comme une fonction de génération de trajectoire avant le suivi de trajectoire. L'algorithme de Dijkstra a été attribué au FPGA.
- Données de ronde : cette fonctionnalité a été ajoutée. Elle consiste à demander au robot d'effectuer une ronde toutes les 2 heures. Le FPGA ayant besoin de clocks pour son fonctionnement, il est inutile de monopoliser des ressources du processeur en plus pour le décompte du temps. Le FPGA est capable de communiquer un flag quand 2 heures sont écoulées, ainsi qu'un reset pour que le logiciel puisse signifier qu'il a lu le flag et demander un redémarrage du compteur.
- Données batteries : comme pour la ronde, un flag communiqué par le FPGA au logiciel est suffisant.
- Ultrasons : comme expliquée dans le en 3.1.3, la gestion du temps pour les ultrasons ne peut pas être gérée efficacement par le logiciel. Le FPGA en a été chargé.

Avec ces blocs définis, voici les spécificités pour chaque brique logicielle :

- Commandes moteur et odométrie : le bloc de communication côté logiciel avec la base roulante a déjà été développé lors du projet 2021 et peut être récupéré.
- GESTIONNAIRE : il s'agit du premier bloc de décision. Il doit être capable de passer du mode autonome au mode manuel en s'assurant que le robot a bien annulé l'action en cours.
- LOCALISATION : ce bloc doit intégrer un filtre de Kalman étendu. Il se base sur les données IMU transmises par le FPGA, ainsi que les données odométriques.
- NAVIGATION : il est activé par la détection d'une alerte en mode autonome. Il se base sur la fonction Dijkstra du FPGA et sur le bloc de localisation pour réaliser son suivi de trajectoire.
- EVITEMENT : il utilise les données ultrasons pour corriger les commandes souhaitées par le bloc de navigation et effectuer des manœuvre d'évitement.

3.2.2 Interface entre partie logique et partie logicielle

L'interface entre le logiciel et le FPGA a été le premier grand point d'interrogation du projet. Pour l'ensemble des membres de l'équipe, il s'agit du premier projet liant FPGA et logiciel. La communication entre PS et PL était alors totalement inconnue. Nous avons d'abord listé toutes les données à échanger entre les deux parties dans la figure 3.5. Chaque donnée est liée à la brique logicielle correspondante. Le développement du FPGA a été effectué avec l'outil Vivado. C'est un logiciel qui permet de créer des fichiers hdl pour programmer la partie logique. De ces fichiers, il est possible d'en faire des blocs nommés IP (intellectual Property) que l'on connecte à d'autres IP. Ils représentent différents modules de la carte, dont le processeur. Dans le cadre du projet, une IP RFID, une IMU et une pour les Ultrasons ont été développés par la développeuse FPGA par exemple.

Il existe des IPs pré-existantes pour générer des modules génériques. Une des IP proposée par Vivado est nommée AXI (Advanced eXtensible Peripheral). Il s'agit d'un bus de communication. Chaque IP créée peut être connectée à ce bloc AXI. Une adresse est alors assignée par le module AXI avec laquelle la lecture et l'écriture de données est possible. Le processeur agit comme un maître qui peut écrire ou lire à chaque adresse. Les blocs IPs sont des slaves qui ne font que répondre. C'est de cette manière que le FPGA et le logiciel ont communiqué.

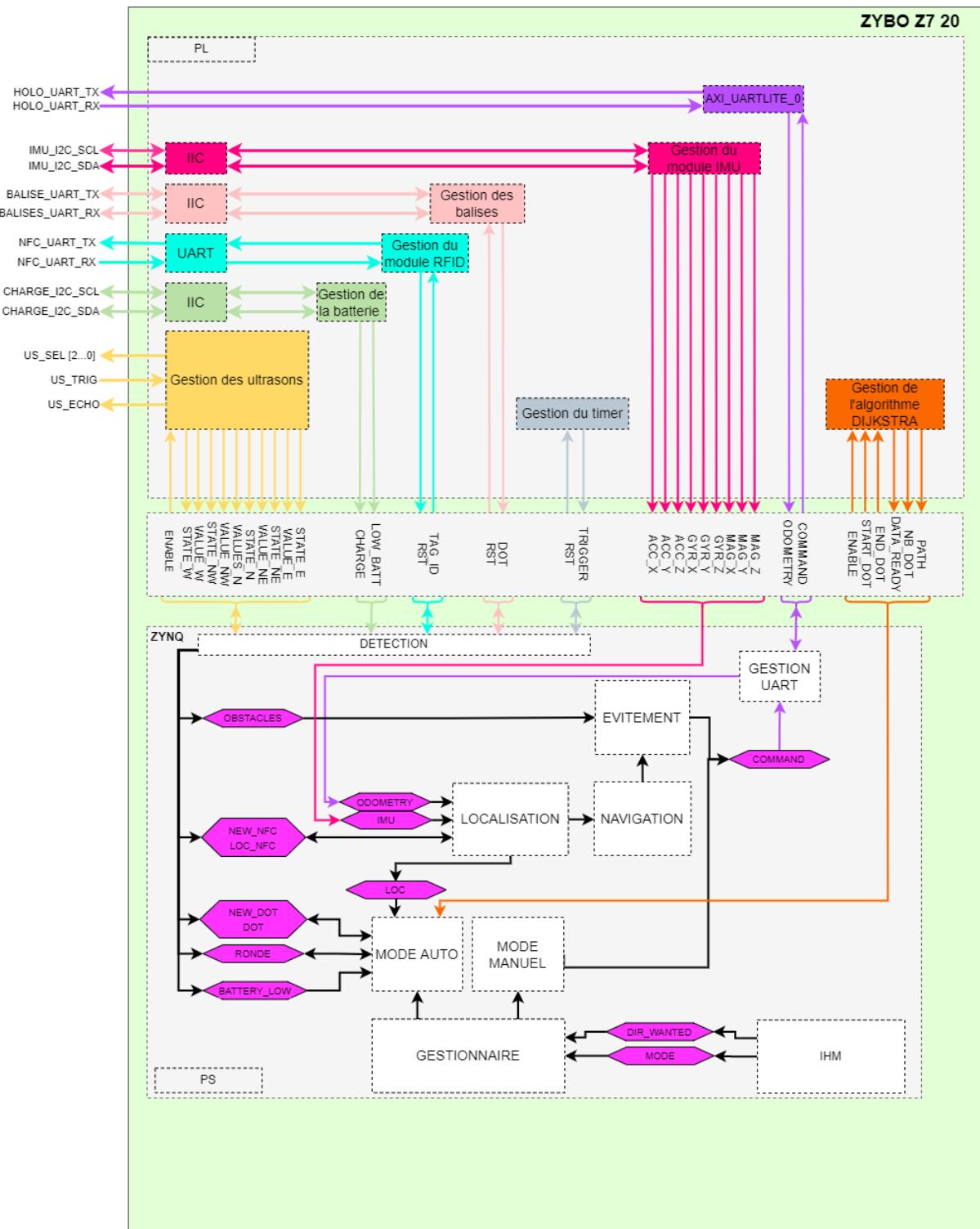


FIGURE 3.5 – Conception générale du système

3.3 Analyse et prise de recul

Pour cette première phase du projet, la méthode employée et les recherches ont été assez efficaces. Nous avons rapidement créé des schémas généraux, puis détaillés permettant ensuite à chacun de pouvoir travailler sans dépendre les uns des autres. Dans notre planning initial nous avions prévu 2 semaines pour la partie conception. Un mois complet était finalement nécessaire. En un mois, la conception détaillée du projet était très précise et a réellement accéléré de nombreux processus par la suite.

La conception plus détaillée côté logiciel, n'était pas suffisamment poussée à la fin de cette phase. Le schéma général définit l'architecture du logiciel mais la description de chaque bloc n'était pas suffisante. Il aurait fallu prendre plus de temps à créer des blocs logiciels précis ainsi qu'à expliciter les inter-connections entre eux. Ces schémas détaillés ont été fait plus tard dans le stage.

Les premières commandes de composants pour le projet ont été passées à partir d'un mois de stage. La développeuse FPGA n'a pas pu commencer son travail immédiatement sans les composants. Il aurait été possible de passer la commande une ou deux semaine plus tôt.

Chapitre 4

Configuration de l'environnement

4.1 Linux embarqué

4.1.1 Petalinux

Lors de la conception, de nombreuses similitudes entre le projet de SIEC à l'INSA et le stage sont apparues. C'est pourquoi un environnement similaire et maîtrisé a été mis en place : un Linux embarqué pour programmer en python. La carte Zybo-Z7-20 possède un port micro SD et peut être configurée pour boot sur ce même port.

Le fabricant de la carte est Digilent. Le site du fabricant propose un guide ainsi qu'un github avec le projet pour installer un OS nommé petalinux. Il s'agit d'un OS basé sur Linux adaptable à toutes formes de cartes de développement.

Pour compiler un tel OS les outils Xilinx : Vivado, Vitis IDE et Petalinux tools ont été téléchargés. Ils permettent de générer les fichiers à monter sur Carte SD. Vivado sert principalement à créer la description hardware de la carte (ports USB, ports Ethernet, GPIO ...). Cette description est ensuite injectée dans un projet petalinux. Le projet peut ensuite être configuré pour changer le lieu de stockage du root file system : en RAM ou en carte SD. Il est aussi possible de configurer la pré-installation de certains packages dont les packages python. Lors de la compilation, le logiciel se base sur cette description pour générer les accès aux différents modules décrits sous Vivado.

Pour se familiariser avec Petalinux, Digilent propose un guide pour une version 2017 [1]. Elle nécessite une machine virtuelle sous Ubuntu 16.04.04 pour fonctionner. La seule configuration a été de changer le stockage du système de fichiers de manière à ce qu'il soit enregistré sur carte SD et non en RAM. De cette manière, toute modification n'est pas perdue à chaque redémarrage de la carte.

Suivre simplement ce guide a demandé près d'une semaine car les machines utilisées manquaient de ressources et la connexion internet était limitée.

Petalinux 2017 n'était pas compatible avec la version de Vivado sur laquelle travaillait la développeuse FPGA. Il a donc fallu changer de version de Petalinux pour en avoir une version plus récente. Dans le Github de Digilent se trouve une version 2020 [2] sans guides cette fois. Le processus de build est le même, le Petalinux a pu être build assez rapidement. Une fois la carte en marche sous Petalinux, il était encore nécessaire de comprendre les spécificités de cet outil : comment accéder aux GPIO et aux différents ports, comment réécrire la partie logique, etc.

De là ont commencé de longues recherches afin de comprendre l'utilisation d'un tel outil avec la documentation de Xilinx [8]. J'ai eu beaucoup de mal à comprendre comment utiliser ce Linux mais au cours de mes recherches, un autre outil basé sur Petalinux a été découvert, une sur-couche de Petalinux nommée PYNQ.

4.1.2 PYNQ

La sur-couche PYNQ est la même que celle utilisée lors du stage 2021. Elle apporte des fonctions intéressantes pour le projet : un environnement pensé pour développer en python, avec des packages simplifiant grandement l'accès aux entrées et sorties de la carte, ainsi qu'un moyen de réécrire la partie logique sans avoir à recompiler le Linux à chaque modification. L'accès au module AXI, qui sert de communication entre PS et PL, est également simplifiée grâce aux librairies proposées, rendant plus simple la création de drivers pour manipuler les différentes IP de la partie logique. L'inconvénient principal est qu'il n'existe pas de projets existant sur les Github de Digilent, de PYNQ ou de Xilinx pour la Zybo-Z7-20, ni d'images pré-compilées. La documentation PYNQ [6] proposait de compiler soi-même le PYNQ. Le fonctionnement est le même que pour Petalinux : réaliser une description de la carte sous Vivado pour compiler le linux.

Les premières tentatives avaient pour objectif de compiler PYNQ 2.6 qui n'est pas la dernière version de PYNQ. Cette phase de compilations a été plus difficile que prévu, due à des dépendances qui n'existaient plus ou des liens erronés. La faible connexion internet et faible puissance de la machine de travail amenaient à près de 8h à 10h de compilation par tentative.

En continuant les recherches en parallèle, un projet sur la Zybo-Z7-20 pour un PYNQ 2.7 a été trouvé [5]. Il s'agit de la version la plus récente de ce Linux. Elle est accessible librement et la personne à l'origine du projet s'est trouvée être très réactive. Après quelques correctifs dans les fichiers de build et dans le Makefile, la compilation a fonctionné. Le Linux embarqué PYNQ était fonctionnel.

Une fois la carte SD montée et insérée dans la Zybo-Z7-20, TeraTerm permet de communiquer entre le PC et la carte en UART. Cet outil a permis de voir tout le processus de démarrage, puis de pouvoir naviguer dans les fichiers de la Zybo.

En connectant le câble Ethernet à un routeur du réseau, toute l'équipe a eu accès au système de fichiers de la carte. C'est un avantage de PYNQ : lorsque la carte est connectée à un réseau, elle est accessible par n'importe quelle machine du réseau. En écrivant l'adresse IP dans un navigateur, une page Jupyter s'ouvre avec le système de fichiers de la carte dans lequel on peut naviguer, rajouter des fichiers, les modifier ou encore ouvrir un terminal. De plus, la carte se retrouvait connectée à Internet ce qui a été utile pour l'installation de packages supplémentaires. En plus de ces éléments, la distribution PYNQ a été pensée pour les cartes de développement. Elle conserve des propriétés de performances suffisantes pour une utilisation dans l'embarqué.

Ce PYNQ compilé présentait tout de même un défaut : le port USB de la carte n'était pas reconnu. Ce qui signifiait que le Dongle USB n'était pas accessible pour créer une communication sans fil. Le problème a été ignoré mais a empêché d'utiliser l'IHM prévue lors de la conception.

4.1.3 Un choix discutable

La compilation et le déploiement du Linux embarqué a demandé près de six semaines. Entre le temps de compilations, les problèmes de versions, les erreurs et les configurations, j'y ai passé beaucoup de temps. Ceci n'était pas prévu dans le planning et m'a laissé beaucoup moins de temps pour la suite du stage.

D'autres parties ont tout de même été avancées pendant les compilations mais mon choix d'utiliser un linux embarqué avait été pensé pour gagner du temps. Le but était de simplifier le travail à venir en permettant de développer et tester plus aisément et rapidement. Au final il se trouve que ce choix a fait perdre beaucoup plus de temps qu'il n'en a fait gagner.

Plusieurs choses auraient pu être mieux faites dans cette période. Tout d'abord, je n'ai pas cherché à savoir comment faire du multi-threading en bare-metal alors que c'était une option envisageable. Ensuite, en ce qui concerne la compilation j'aurais dû mieux me renseigner en amont sur les linux existants avant de choisir la Zybo-Z7-20 comme carte de développement. Effectivement, j'ai eu un Petalinux fonctionnel mais aucune idée de comment l'exploiter. Il est probable qu'une formation à Petalinux aurait demandé moins de ressources.

Malgré tout ces éléments, la mise en place d'un linux embarqué est une fierté personnelle importante. C'est une étape qui demande toujours beaucoup de temps et de persévérance dans un projet. Cette expérience autour du linux m'a permis d'apprendre beaucoup sur ce domaine. J'ai pu configurer du kernel, configurer le device-tree et découvrir les outils Xilinx. De plus, cette étape du projet est finalement un succès. Le fait d'avoir un linux fonctionnel a permis de tester les briques logicielles développées.

4.2 Environnement de programmation

4.2.1 Les outils utilisés

Pendant les compilations du linux embarqué, le développement du logiciel du robot a démarré. La figure 3.5 permet de visualiser l'architecture de ce logiciel. Pour programmer j'ai utilisé Visual Studio Code sous Windows et le package threading.

4.2.2 Architecture logiciel

Un fichier `__init__.py` a été mis en place, qui a permis l'initialisation et le lancement tout les threads un par un : détection, localisation, navigation, contrôle autonome et gestionnaire. Le Gestionnaire et le mode autonome ont d'abord été programmés. Les diagrammes d'états sont présents en 4.1 et 4.2. Ces deux diagrammes avec le mode manuel forment la base de l'architecture logicielle.

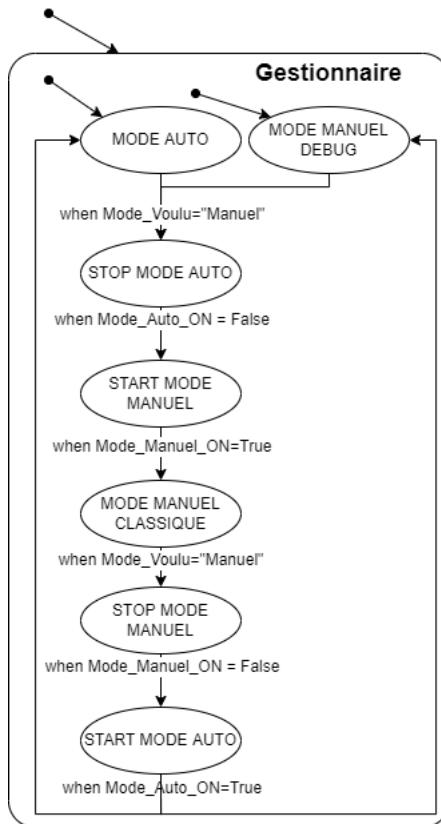


FIGURE 4.1 – Diagramme d'état du Gestionnaire

Un système d'interruption des threads a été ajouté. C'est à ce moment qu'est apparue une hiérarchie dans les threads : lorsque le gestionnaire demande le passage en mode manuel, il demande d'abord l'interruption du thread de contrôle autonome. Mais avant de s'interrompre, le contrôle autonome doit lui-même demander l'interruption du thread navigation et attendre qu'il soit bien dans un état interrompu.

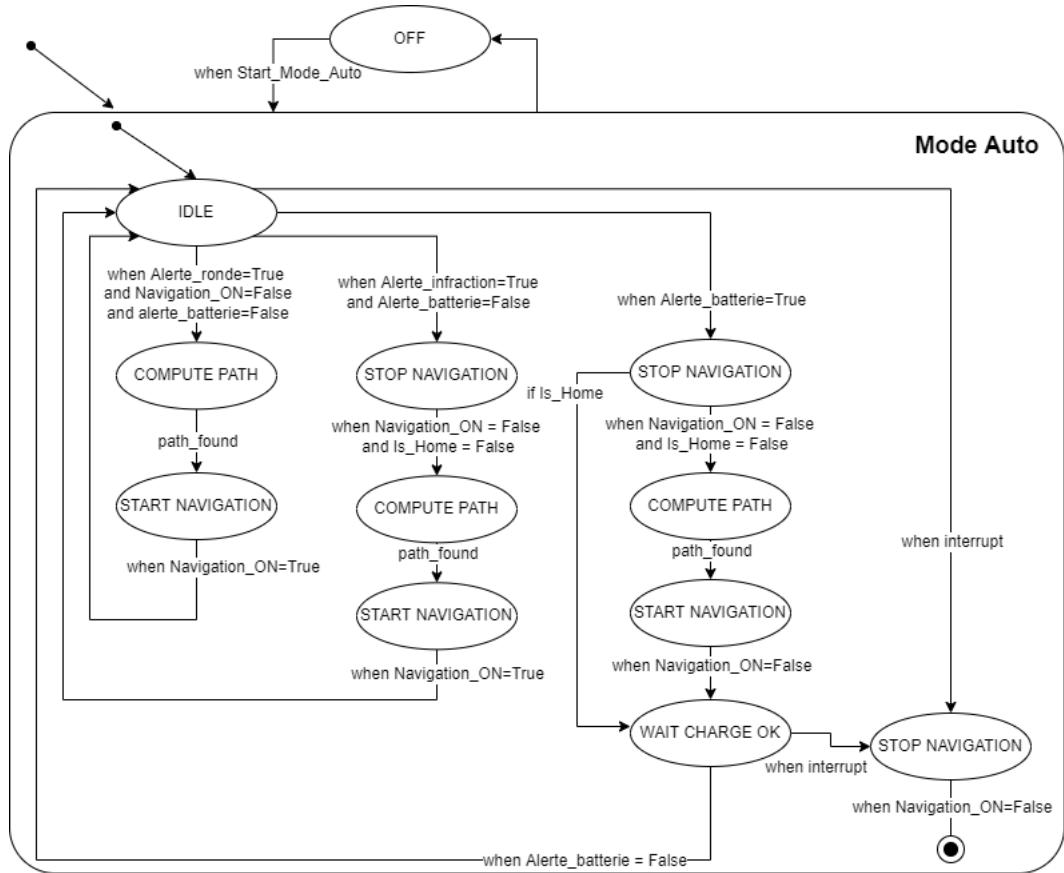


FIGURE 4.2 – Diagramme d'état du mode autonome

Il en va de même lorsque l'on passe du mode manuel au mode autonome. Le thread de contrôle manuel a en fait 2 états :

- L'état "mode manuel classique" où il traduit les entrées claviers en directions pour diriger le robot ainsi que l'entrée "m" qui permet de demander le passage en mode manuel.
- L'état "mode manuel debug" qui fonctionne en parallèle du mode autonome. Dans ce mode il n'interprète plus les déplacements mais uniquement l'entrée "m" pour repasser en mode manuel.

Un système robuste a été développé, capable de passer d'un mode à l'autre en vérifiant que chaque thread est bien interrompu avant le changement de mode. Cette sécurité sert de base et doit être solide pour pouvoir développer correctement le reste du logiciel sans problèmes.

4.2.3 Le mode manuel pour le debogage

Afin de pouvoir développer et tester les programmes, un environnement de test adéquat est nécessaire. Il n'est pas envisageable d'attendre que la dévelopeuse FPGA ait terminé de faire fonctionner tous les capteurs.

En mode autonome le thread de contrôle manuel continue de fonctionner mais n'interprète plus les commandes de déplacement. Il est donc possible de s'en servir à des fins de débogage pour simuler la détection par un capteur. Ceci peut convenir pour simuler :

- La détection d'intrusion par un balise
- La détection d'un tag NFC
- La détection d'un signal de batterie faible
- La détection d'un signal de ronde

Le développement du contrôle manuel du robot a donc été poursuivi en y ajoutant ces fonctionnalités. Le contrôle manuel attend une entrée clavier et l'interprète en commande de direction. L'ajout de ces entrées a aussi permis de tester le thread de mode autonome au complet et de le valider.

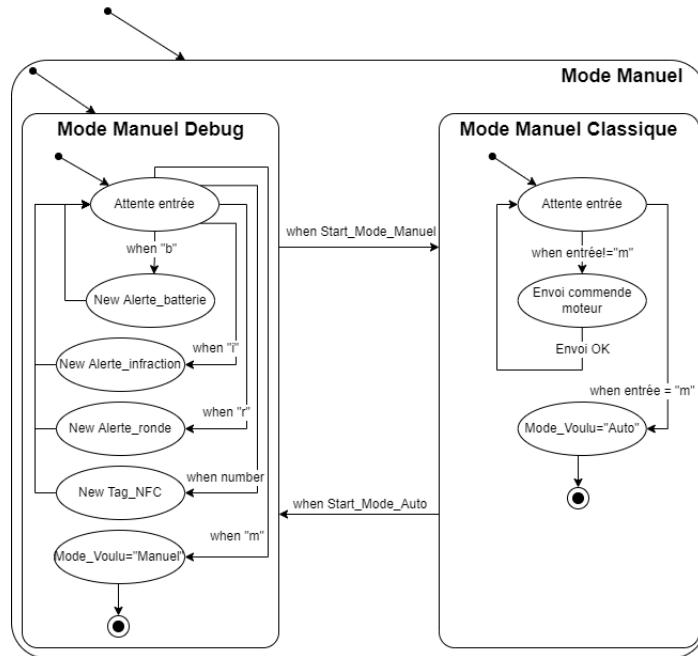


FIGURE 4.3 – Diagramme d'état du mode manuel

Ceci termine la mise en place de l'environnement de travail. Il est composé en premier lieu d'un linux embarqué et d'une base logicielle. Le linux sur la carte permet de programmer en python et d'accéder aux entrées et sorties ainsi que de communiquer avec le FPGA facilement. La première version de base du logiciel est prête à l'emploi, permet le passage de manuel à autonome de manière sûre et offre des outils pour le débogage des futurs blocs logiciels.

Chapitre 5

Localisation

Le premier bloc logiciel réalisé a été le bloc de localisation. Il a servi au repérage du robot dans l'espace. Ses données ont été essentielles, principalement la donnée d'orientation du robot pour la navigation. La première étape a été de réaliser un filtre de Kalman étendu (EKF) fusionnant les données odométriques et IMU [4] [3]. La seconde étape a consisté à tester et régler le filtre à l'aide de datasets, puis à l'implémenter avec les données réelles du capteur. Ce rapport n'entrera pas dans les détails du fonctionnement d'un filtre de Kalman pour se concentrer sur son paramétrage et son exploitation.

5.1 Le filtre de Kalman étendu

L'objectif du filtre de Kalman est de fusionner les données IMU et odométrique pour calculer la position du robot. C'est à dire ses coordonnées x et y dans le repère de l'agence, ainsi que son angle d'orientation θ . Les données IMU et odométriques sont récupérées selon le repère du robot (x' , y' , z'). Il faut donc les convertir en coordonnées de l'agence (x , y , z). 5.1

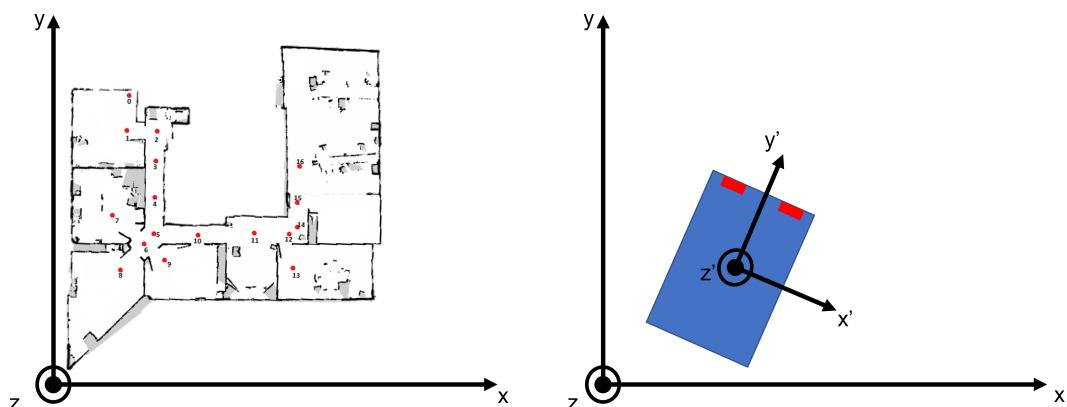


FIGURE 5.1 – Repères de l'agence par rapport à celui du robot

Les équations du filtre de Kalman sont les suivantes :

Predict :

$$\hat{X}_{k+} = A \cdot \hat{X}_k$$

$$P_{k+} = A \cdot P_k \cdot A^T + Q$$

Correct :

$$K_{k+1} = P_{k+} \cdot H_{k+1}^T \cdot (R_{k+1} + H_{k+1} \cdot P_{k+} \cdot H_{k+1}^T)^{-1}$$

$$X_{k+1}^a = X_{k+1}^+ + K_{k+1} \cdot (Z - h(X_k^+))$$

$$P_{k+1} = (Id - K_{k+1} \cdot H_{k+1})$$

X : Vecteur d'état

A : Matrice de transition

Q : Suite de vecteurs aléatoires de bruit d'état

R : Suite de vecteurs aléatoires de bruit d'observation

h : Matrice d'observation

\hat{X}_k : Vecteur d'état estimé

\hat{X}_{k+} : Vecteur d'état prédict

P_{k+} : Matrice de covariance de l'erreur de prédiction de l'état

P_k : Matrice de covariance de l'erreur d'estimation de l'état

Z : Vecteur de mesures

K_{k+1} : Gain de Kalman

Pour réaliser le filtre, les matrices A , H , X , Z , Q et R ont été déterminées.

L'IMU transmet 9 données, trois données d'accélération par l'accéléromètre, trois données de vitesse angulaire par le gyroscope et trois données magnétique par le magnétomètre. Sur toutes ces données, seules les accélérations en x' et en y' sont utiles, ainsi que la donnée de vitesse angulaire selon z' . L'odométrie transmet des données de vitesse de translation et de rotation. Sous forme matricielle, Z représente les entrées capteur et Y la sortie correspondante. Le vecteur d'états X regroupe toutes les variables d'états du système :

$$Y = \begin{bmatrix} x \\ y \\ \theta_z \end{bmatrix} \quad X = \begin{bmatrix} x \\ v_x \\ a_x \\ y \\ v_y \\ a_y \\ \theta_z \\ \dot{\theta}_z \end{bmatrix} \quad Z = \begin{bmatrix} v_{x'} \\ v_{y'} \\ a_{x'} \\ a_{y'} \\ \dot{\theta}_{z_{odo}} \\ \dot{\theta}_{z_{gyr}} \end{bmatrix}$$

La matrice de transition A est définie par le système d'équations suivant :

$$\left\{ \begin{array}{l} x^+ = x + v_x \cdot \Delta t + a_x \cdot \frac{\Delta t^2}{2} \\ v_x^+ = v_x + a_x \cdot \Delta t \\ a_x^+ = a_x \\ y^+ = y + v_y \cdot \Delta t + a_y \cdot \frac{\Delta t^2}{2} \\ v_y^+ = v_y + a_y \cdot \Delta t \\ a_y^+ = a_y \\ \theta_z^+ = \theta_z + \dot{\theta}_z \cdot \Delta t \\ \dot{\theta}_z^+ = \dot{\theta}_z \end{array} \right. \quad A = \begin{bmatrix} 1 & \Delta t & \frac{\Delta t^2}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & \frac{\Delta t^2}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

H est déterminé par l'équation $Z = h(X)$:

$$\left\{ \begin{array}{l} v_{x'} = v_x \cdot \cos \theta - v_y \cdot \sin \theta \\ v_{y'} = v_x \cdot \sin \theta + v_y \cdot \cos \theta \\ a_{x'} = a_x \cdot \cos \theta - a_y \cdot \sin \theta \\ a_{y'} = a_x \cdot \sin \theta + a_y \cdot \cos \theta \\ \dot{\theta}_{z_{odo}} = \dot{\theta}_z \\ \dot{\theta}_{z_{gyr}} = \dot{\theta}_z \end{array} \right.$$

$$H = \frac{\partial h}{\partial X} = \begin{bmatrix} 0 & \cos \theta & 0 & 0 & -\sin \theta & 0 & -v_x \cdot \sin \theta + v_y \cdot \cos \theta & 0 \\ 0 & \sin \theta & 0 & 0 & \cos \theta & 0 & -v_x \cdot \cos \theta - v_y \cdot \sin \theta & 0 \\ 0 & 0 & \cos \theta & 0 & 0 & -\sin \theta & -a_x \cdot \sin \theta + a_y \cdot \cos \theta & 0 \\ 0 & 0 & \sin \theta & 0 & 0 & \cos \theta & -a_x \cdot \cos \theta - a_y \cdot \sin \theta & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

La matrice de bruit d'observation R a été déterminée en prenant en compte l'erreur fournie par la datasheets de l'IMU. L'odométrie ne possède pas de datasheet, l'erreur a été estimée par plusieurs tests.

La matrice de bruit d'état Q a été fixée en observant le comportement du filtre à un état fixe. Le bruit obtenu en sortie correspond aux variances des variables d'état. Avoir une matrice Q non-nulle aide à mieux comprendre le système et permet de meilleures estimations.

$$R = \begin{bmatrix} \epsilon_x & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \epsilon_y & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \epsilon_{v_x} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \epsilon_{v_y} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \epsilon_{ax} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \epsilon_{ay} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \epsilon_\theta & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \epsilon_{\theta_z} \end{bmatrix}$$

$$Q = \begin{bmatrix} \sigma_{odom_x'} & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{odom_y'} & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{acc_x'} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{acc_y'} & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{odom_z'} & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_{gyr_z'} \end{bmatrix}$$

Une fois l'étude théorique du système terminée, spyder a été utilisé pour programmer, tester et ajuster les configurations du filtre, principalement Q, R et les données d'initialisation.

5.2 Comportement du filtre

La phase de test a été effectuée avec des données odométriques uniquement dans un premier temps. Les données IMU n'étaient pas encore disponibles à cet instant du stage et son utilisation a pris du retard à cause d'un défaut sur le capteur IMU reçu. Il a été commandé une seconde fois. Les matrices de la partie précédente ont été réduites pour ne prendre en compte que les données odométriques.

Les données utilisées pour les tests ont été récoltées en déplaçant le robot en mode manuel. Les données odométriques ont été récoltées toutes les 100ms et enregistrées dans un fichier. Plusieurs datasets ont été réalisés sur différents comportements du robot : immobile, mouvement rectiligne (dans 8 directions), rotation sur place, virage.

La premier test a été de vérifier le comportement du filtre avec un robot immobile. Les données récoltées sont toutes nulles, le filtre ne modifie donc pas les coordonnées. Le filtre ne diverge pas à l'arrêt.

Le second test a consisté à visualiser un mouvement rectiligne. Pour rappel, le robot est équipé de roues omnidirectionnelles. Il peut donc se déplacer dans 8 directions tout en gardant la même orientation. Les graphiques obtenus sur un déplacement rectiligne de 60 à 70 cm sont représentés 5.2 et 5.3. Ces graphiques représentent les coordonnées du robot.

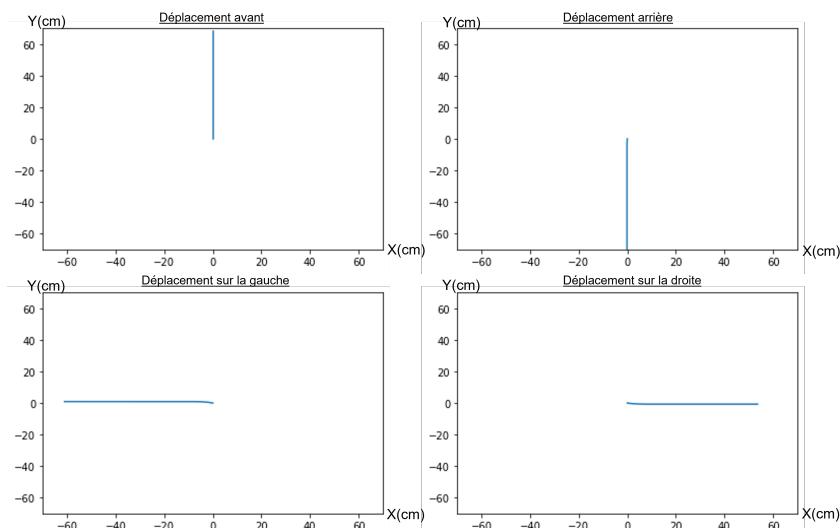


FIGURE 5.2 – Translation uniforme en x et en y

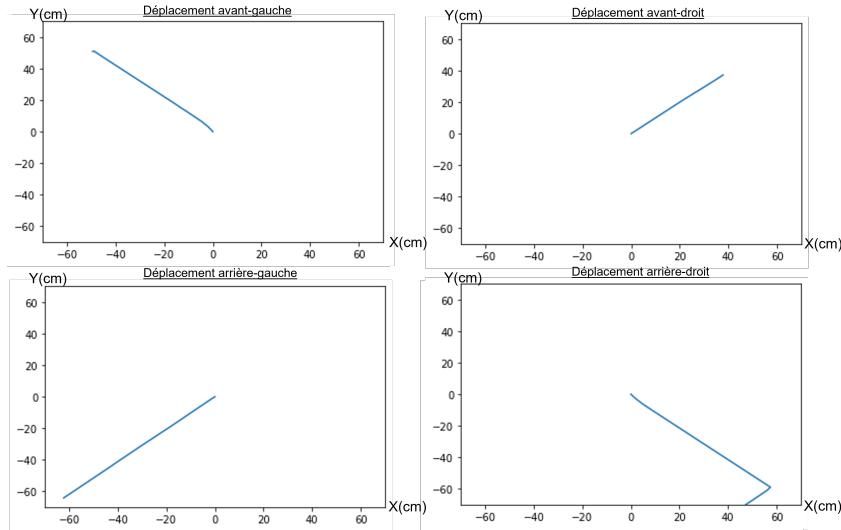


FIGURE 5.3 – Translation uniforme en diagonale

Sur le graphique représentant le mouvement arrière, un léger biais apparaît dans les premiers instants du déplacement. Il arrive que ce biais apparaisse également sur un déplacement vers l'avant. Son ordre de grandeur étant très faible il n'aura pas d'impacts sur le robot.

On observe que sur les mouvements latéraux le long de l'axe x , un décalage est détecté en y de l'ordre de 1cm. Ce comportement est imprévisible. Sur 3 mètres une erreur de 3cm est acceptable, mais sur des plus grandes distances le calcul n'est plus fiable.

Enfin les déplacements diagonaux montrent un glissement à la fin du mouvement qui fausse la localisation. Il peut aller de 1cm à une dizaine de centimètre comme observé sur le déplacement arrière-droit, ce qui représente une erreur bien trop importante dans ce cas précis. Cette erreur est due à deux éléments. D'abord, les roues du robot ont un effet de glissement plus important à la fin de ce type de mouvement que sur les autres mouvements. Ce glissement est amplifié par le filtre qui calcule un réel déplacement de plusieurs centimètres.

Des datasets plus importants et plus précis ont ensuite été analysés avec un mouvement rectiligne mesuré et chronométré, ainsi qu'un virage. 5.4 5.5

Le robot a parcouru 305 cm. Les coordonnées finales calculées par le filtre sont (311.13, 0.16). Cela correspond à un décalage de 6,13cm sur une distance de 305cm soit une erreur de 2% là encore c'est une erreur acceptable en prenant en compte le fait que la localisation sera corrigée par RFID tout les 2 mètres environ lors de la navigation.

Le filtre a un comportement proche de la réalité. Une correction fiable de la localisation est nécessaire tous les 3m environ sur des déplacements arrière, avant, droite et gauche. Les déplacements diagonaux ne sont utilisés que lors des phases de correction de trajectoire du robot, donc sur des très courtes distances afin de limiter l'erreur.

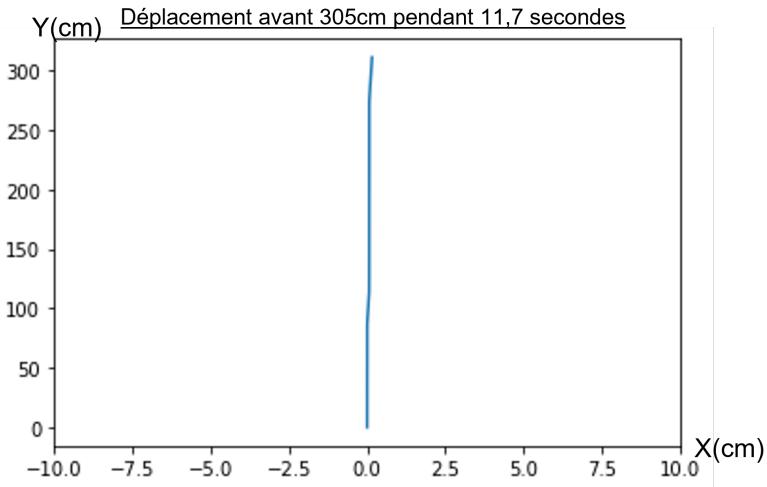


FIGURE 5.4 – Mouvement mesuré rectiligne

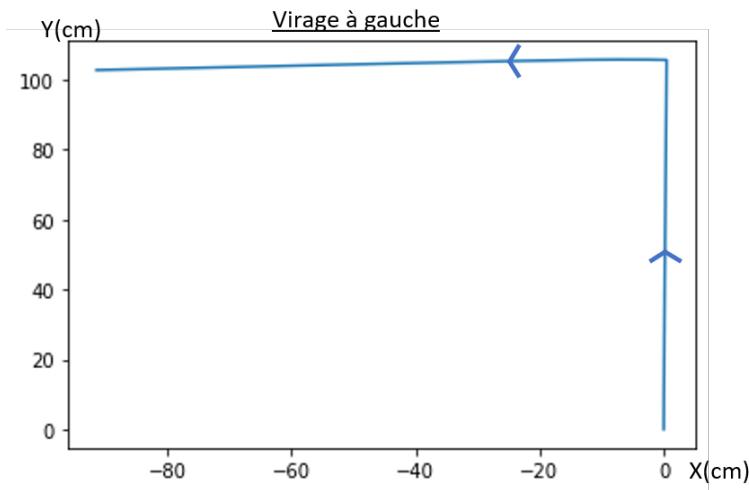


FIGURE 5.5 – Mouvement rectiligne avant sur 1 mètre, rotation à 90 degrés puis mouvement rectiligne avant sur 1 mètre

Une fois l'IMU prêt à être manipulé, de nouveaux datasets incluant à la fois les données odométriques et IMU ont pu être réalisés. Cependant le filtre a montré une divergence très importante avec l'ajout de ces nouvelles données comme observé en 5.6. Le mouvement réalisé est le même que précédemment : un mouvement rectiligne le long de Y sur 3 mètres. Les données sont absurdes et atteignent des valeurs de l'ordre de 10^7 .

Plusieurs tests ont été réalisés et tous ont mené à un résultat similaire : une forte divergence du filtre. Pour pallier à ce problème, la matrice de covariance de bruit d'état R a été modifiée et ajustée par essais-erreurs jusqu'à obtenir un comportement cohérent.

5.7

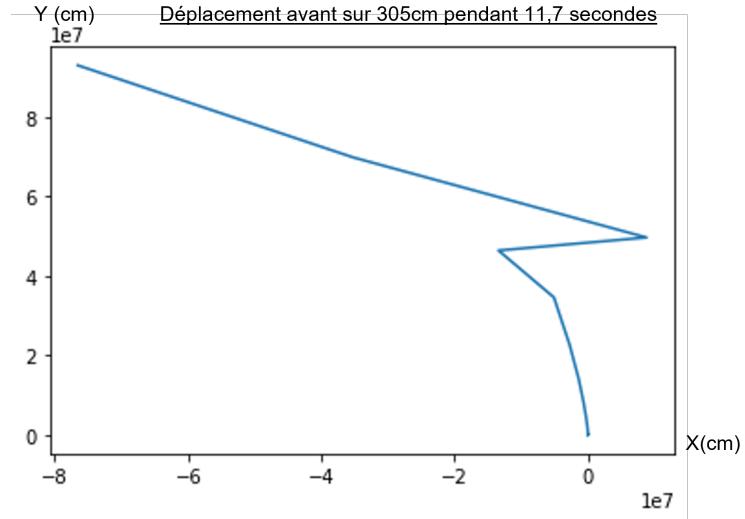


FIGURE 5.6 – Premier test du filtre complet

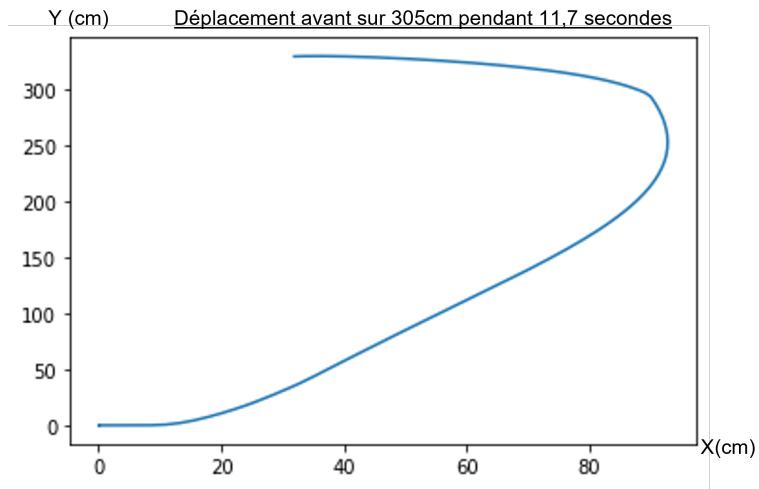


FIGURE 5.7 – Deuxième phase de test du filtre

Ce deuxième graphique est le meilleur comportement obtenu avec les données IMU. La distance parcourue en y est cohérente, mais on observe un déplacement important le long de l'axe x . Après une étude plus approfondie des données reçues, il s'est avéré que le capteur IMU montrait une précision bien inférieure à celle annoncée par le constructeur. La précision annoncée est de $1 \mu\text{m}/\text{s}^2$. Dans les faits, le bruit du capteur apporte une erreur beaucoup plus grande pouvant atteindre $0.1 \text{ m}/\text{s}^2$. Ceci rend les données reçues par ce capteur très peu fiables. Le décalage final est de 30cm. Un changement de capteur pour un modèle plus performant est nécessaire.

L'ajout des données IMU rendant le calcul moins fiable, il a été décidé de faire l'intégration du bloc de localisation avec le filtre sur les données d'odométrie uniquement.

5.3 Prise de recul

Cette partie localisation a été un réel défi sur ce stage. Malgré les notions en filtre de Kalman acquise par les cours de l'INSA, la mise en pratique en autonomie complète d'un filtre pour un projet concret a nécessité de nombreuses recherches complémentaires. Sur l'aspect théorique le filtre de Kalman étendu (EKF) n'avait été traité qu'en surface et sa mise en pratique est bien plus complexe que le filtre de Kalman (KF). Il a été nécessaire de comprendre le rôle de chaque matrice précisément. Grâce à cette étude, la partie développement du logiciel a été beaucoup plus rapide. L'analyse des résultats obtenus a manqué de précision. L'IMU n'a pas pu être déployé sur le robot à la date prévue ce qui a retardé les analyses des données. Le temps restant n'était pas suffisant pour la mise en place d'une solution de secours. La brique d'intégration a été priorisée.

La solution n'utilisant que les données d'odométrie reste cependant très fiable avec seulement 2% d'erreur. Elle a été intégrée au robot et a ensuite été exploitée par le bloc navigation.

Chapitre 6

Navigation

Lorsqu'une infraction est détectée, le lieu de la détection et l'emplacement actuel du robot sont récupérés. Avec ces deux éléments, la fonction Dijkstra programmée par le FPGA permet de déterminer un chemin à parcourir pour rejoindre le lieu de l'infraction. C'est dans ce but que le bloc de navigation a été développé.

6.1 La carte

Avant d'expliquer le fonctionnement de la navigation, il est essentiel de comprendre comment a été réalisée la carte des locaux et le calcul du chemin à parcourir. Sur la carte des locaux 6.1, 17 points de passages ont été fixés. Si aucun mur n'est traversé entre deux points, un arc est formé en liant ces points entre eux. Ce schéma de carte avec des points et des arcs est une carte **topologique**, à ne pas confondre avec une carte topographique.

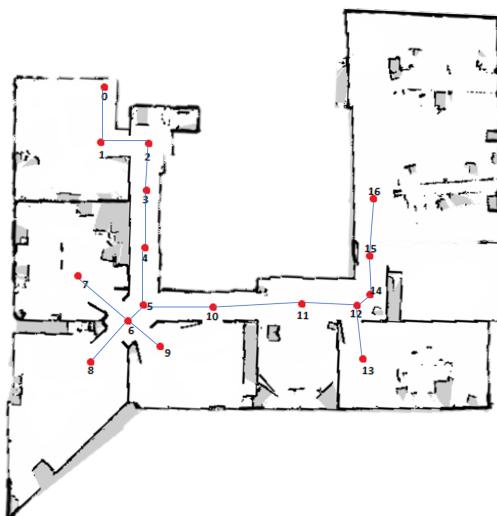


FIGURE 6.1 – Carte et topologie des locaux

Chaque point de cette carte topologique est associé à des coordonnées et chaque arc est associé à une distance à parcourir ainsi qu'une orientation.

Ce type de carte s'intègre parfaitement au projet. L'algorithme de Dijkstra sur FPGA se base sur cette carte, le chemin renvoyé par ce-dernier est en fait une suite de points correspondant aux points rouges sur la carte. Les étiquettes RFID qui sont placés aux différents points de passage sont en fait placés au niveau de ces points. Ils ont alors une double utilité : mettre à jour le calcul de la localisation, et vérifier qu'un point est bien atteint avant de passer au point suivant lors de la navigation.

L'ensemble de la carte a été pré-enregistrée manuellement dans la carte de développement. Les points et les arcs ont également été spécifiés à la main.

6.2 La navigation

Afin de contrôler la navigation le robot se base sur :

- Son orientation dans l'espace, l'angle θ calculé dans le bloc localisation
- Des étiquettes RFID disposées sur les points de passages du robot

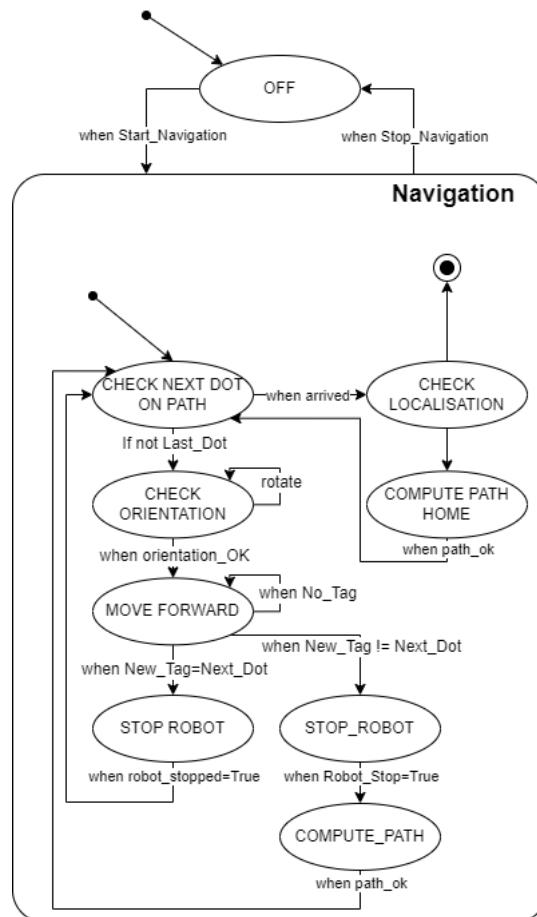


FIGURE 6.2 – Diagramme d'état de la navigation

Afin de tester ce programme le bloc de localisation par l'odométrie est disponible et peut être utilisé. La simulation de détection d'étiquettes se fait par clavier. Le comportement est satisfaisant avec un temps de réaction de l'ordre de 150 ms. En cas de détection d'une erreur telle qu'une mauvaise étiquette détectée, le robot s'arrête simplement. La fonction de reprise sur erreur est une idée d'amélioration du logiciel. Une fois une étiquette détectée, le robot est capable de s'orienter dans la bonne direction pour le point suivant avec une erreur de ± 2 degrés.

L'intégration de ce bloc a été très largement retardé par la livraison du capteur RFID. Commandé mi-avril, ce dernier est arrivé début août et n'a été intégré au système que mi-août. Les résultats des tests d'intégration sont les suivants :

- La vitesse de rotation du robot doit être réduite car un effet de glissement a lieu au niveau des roues à la fin d'une rotation ce qui provoque une plus grande erreur d'orientation. La vitesse a été réduite à 5 deg/s. Il paraît clair qu'un capteur plus précis est essentiel au bon fonctionnement du robot. Obtenir un filtre de Kalman intégrant des données IMU fiables semble être une bonne solution.
- La vitesse de translation du robot est correcte, les étiquettes étant lues toutes les 200ms.
- Le comportement général correspond aux attentes
- Après 3 ou 4 virages, l'erreur d'orientation devient importante. Un système de rectification devait être mis en place, cette tâche est confiée à la développeuse FPGA qui va calculer la déviation en fonction de l'étiquette détectée sur un point. L'image 6.3 permet de visualiser la solution envisagée.

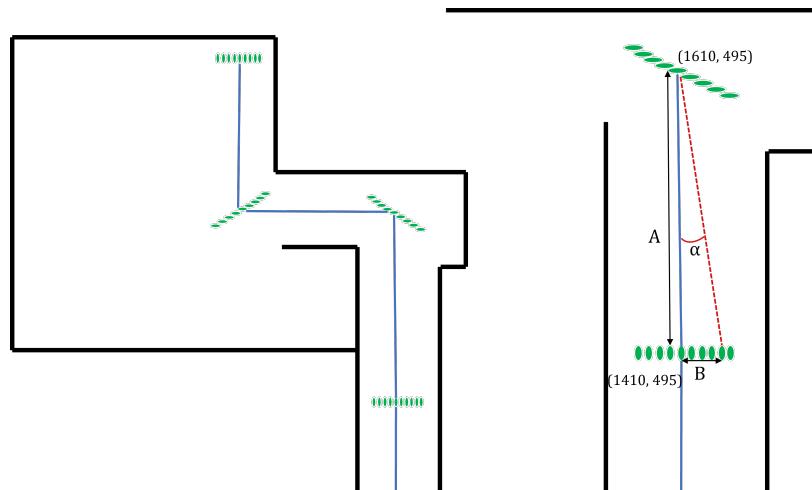


FIGURE 6.3 – Schéma de la correction de l'orientation par RFID

Comme chaque point de passage est constitué d'une ligne d'étiquettes RFID dont la position est connue, chaque étiquette donne à la fois le point qu'il représente sur la carte et la position dans la ligne. La figure précise ici les coordonnées du RFID central de chaque ligne. Dans cet exemple, le robot est parti de l'étiquette centrale du point 2 qui a comme coordonnées ($X_{P2\text{centre}} = 1610$, $Y_{P2\text{centre}} = 495$). Il devait rejoindre l'étiquette centrale du point 3 ($X_{P3\text{centre}} = 1410$, $Y_{P3\text{centre}} = 495$). Il a pris une trajectoire légèrement déviée de α . c'est cet angle qu'il est nécessaire de calculer. Il a détecté l'étiquette $P3_{t9}$ qui a comme coordonnées ($X_{P3_{t9}} = 1410$, $Y_{P3_{t9}} = 510$)

En fonction de l'étiquette détectée et de l'étiquette précédente, il est possible de connaître le décalage α du robot et par conséquent de corriger le décalage tant au niveau de la localisation selon x et y qu'au niveau de son orientation :

$$\alpha = \arctan \frac{B}{A} = \arctan \frac{Y_{P3\text{centre}} - Y_{P3_{t9}}}{X_{P2\text{centre}} - X_{P3\text{centre}}}$$

Ainsi, il est possible de corriger la 3ème donnée de localisation, l'orientation.

Cette partie est en cours de réalisation et fait partie des derniers objectifs de la fin du stage.

6.3 Analyse et prise de recul

La majeure partie du travail réalisé sur la navigation a eu lieu lors de la phase d'intégration. Il s'agit du bloc logiciel qui fait appel à presque la totalité des capteurs et actionneurs. Son fonctionnement est correcte et le robot suit sa trajectoire comme la carte le lui indique. C'est lors de son intégration que des problématiques inattendues sont apparues. La livraison des composants, principalement le lecteur RFID, a pris du retard ce qui a longtemps retardé la phase d'intégration complète. Pour le peu de temps de test avec le RFID, le travail effectué est conséquent et le résultat général est très satisfaisant. Nous avons su nous organiser pour faire en sorte de n'avoir qu'à brancher le module pour les tests. Au niveau logiciel, ma mission a consisté à créer des fonctions d'utilisation du module RFID sans réellement pouvoir les tester, puis intégrer ces fonctions aux blocs de localisation et de navigation. Des problèmes de conception (correction de l'orientation, vitesse de déplacement) ont été identifiés rapidement et des solutions ont été trouvées puis intégrées.

Chapitre 7

Détection d'infractions

La détection des infractions doit permettre au robot de savoir quand et où aller. Si une infraction est détectée dans l'open space des locaux, le robot doit savoir qu'il faut aller au point 16 de la carte 6.1. La solution utilisée a été reprise du projet 2021 : des balises de détection intégrant un détecteur de mouvement et un ESP32. Un ESP32 est également fixé sur le robot et sert de récepteur.

C'est principalement la développeuse FPGA qui a ajusté le fonctionnement des balises et qui traite la réception d'une alerte envoyée par une balise. Côté logiciel, l'alerte et le lieu de l'infraction sont communiqués par le FPGA directement.

L'objectif du logiciel a été de faire en sorte que l'infraction soit bien lue et traitée. Lorsque qu'une alerte intervient en mode autonome, c'est le bloc mode autonome qui traite l'alerte, récupère l'identifiant de la balise, détermine à quel point de la carte elle correspond, et lance la navigation vers ce point.

Certaines spécificités du fonctionnement ont tout de même été réfléchies. Par exemple, si une infraction au point 16 est reçue, le robot commence alors à se diriger vers ce point. Si une nouvelle infraction est détectée au point 13 alors que le robot est encore dans la phase de navigation vers le point 16. Quel doit être le comportement du robot dans ce cas ?

Initialement, le programme faisait en sorte de ne traiter que l'infraction la plus récente. Dans ce cas, la navigation vers le point 16 est interrompue pour prioriser le point 13.

Après quelques tests, il paraissait évident que ce n'était pas la bonne solution car le robot pourrait ne faire que s'interrompre si les points 13 et 16 détectent tout deux du mouvement en continu, comme s'il y avait une personne dans chaque pièce. Dans ce cas le robot ne ferait qu'interrompre sa navigation en boucle en voulant prioriser l'un puis l'autre. Le fonctionnement retenu est donc une FIFO où chaque infraction est traitée dans l'ordre d'arrivée.

Lors de l'intégration, les capteurs de mouvement ont affiché un comportement inattendu. Des mouvements étaient détectés continuellement. Chaque balise envoyait une alerte toute les 5 secondes environ. Un changement de capteur semble être la meilleure solution ici. Cela a été suggéré comme future amélioration du projet. Pour ce qui est de la réaction de logiciel, ce dernier arrive bien à traiter la balise et lance la navigation moins de 1s après avoir reçu l'alerte.

Chapitre 8

Blocs non-intégrés

8.1 IHM

Le module IHM a été en partie réalisé. L'objectif était d'utiliser un dongle WiFi se connectant à la carte de développement afin de pouvoir accéder à des informations à distance avec le robot : mission en cours, changement de mode, contrôle à distance, localisation, etc. Il s'agit d'un module très utile au debogage mais pouvait également être un confort pour l'utilisateur si l'interface avait été pensée pour une utilisation client.

Afin de pouvoir réaliser cette IHM, une formation en HTML, CSS et javascript a été suivie. Des recherches ont été menées afin de réaliser une interface capable de communiquer avec un programme python. C'est finalement l'outil Flask qui a été utilisé. C'est un package python qui offre la possibilité de réaliser une IHM web à partir de fichiers HTML, CSS et javascript. Il est possible de récupérer ou d'envoyer des données sur l'IHM aisément. Les simulations ont permis de vérifier que les données pouvaient correctement circuler. Les données de direction par exemple étaient correctement lues. Une visualisation de l'application réalisée est visible sur l'image 8.1. Un bandeau sur le dessus permet de passer du mode auto au mode manuel. En mode manuel, des boutons cliquables permettent d'envoyer des ordres de commande du robot.



FIGURE 8.1 – IHM développée sous flask

Le développement n'a pas été poussé plus loin car le port USB de la carte n'était reconnu par le linux embarqué sur la carte. Sans ce port il était impossible de connecter un dongle et de faire fonctionner l'IHM. De nombreuses modifications ont été apportées au device-tree, et au kernel pour tenter de trouver une solution mais aucune solution n'a résolu le problème. Comme l'IHM est un bloc de débogage utile uniquement à la partie logicielle, il a été décidé d'abandonner son développement.

8.2 Évitement d'obstacles

Le bloc évitement d'obstacles a été pensé dans l'architecture logicielle et matérielle lors de la phase de conception. Des ultrasons avaient été choisis avec un MUX/DEMUX sur la carte interface. Cependant c'est un planning très ambitieux qui avait été mis en place, la réalisation de l'évitement d'obstacles a donc été catégorisée comme optionnelle.

À la place, un des objectifs de la fin de stage est d'utiliser les ultrasons en ajoutant un bloc qui reçoit les commandes moteurs voulues par le mode manuel ou le bloc de navigation. Il filtre ces commandes en fonction de ce que voient les ultrasons. Si un obstacle est sur la trajectoire, le filtre coupe toute les commandes demandant un mouvement vers l'avant. Ce bloc n'a pas encore été réalisé, c'est un des objectifs de la fin de ce stage.

Chapitre 9

Architecture mécanique

Ce dernier chapitre présente l'architecture mécanique réalisée pour le système. La base roulante n'a pas de structure permettant la fixation de tous nos capteurs et cartes. Elle ne donne accès qu'à 3 petites vis pour fixer une structure sur le dessus.

9.1 Workflow

Pour réaliser cette structure, une imprimante 3D était à disposition : la Creality Ender 3. Un workflow simple mais suffisant a été mis en place pour la réalisation de la structure. Le logiciel de modélisation est Thinkercad, outil de créations 3D sur navigateur. Il est simple d'utilisation et permet de réaliser beaucoup de structures. Malgré le manque d'outils spécifiques comme une règle dans le logiciel, il a été suffisant pour les besoins du système. Ce logiciel a permis de créer des fichiers STL à partir d'une structure 3D. Il a ensuite été nécessaire d'utiliser un outil permettant de générer un fichier interprétable par l'imprimante. C'est ce qu'on appelle un slicer. Le slicer utilisé était Ultimaker Cura.

9.2 La structure

L'idée générale de la structure était de ne pas penser au système en entier immédiatement, mais de penser à une base solide sur laquelle de nombreux éléments pouvaient être ajoutés. Cette méthode permet beaucoup plus de flexibilité en cas d'imprévus comme un changement de modèle de capteur ou de carte de développement.

Pour des raisons pratiques, les cartes devaient être proches du centre du robot de manière à ce que les capteurs puissent être ajoutés autour de la structure. Le système réalisé se base sur le concept d'un rack pouvant accueillir des tiroirs. Sur une hauteur de 10cm, 6 emplacements de tiroirs ont été placés. Les tiroirs pouvaient prendre une forme libre, pouvant s'adapter à la carte supportée. De petites encoches permettaient de fixer les tiroirs de manière à ce que, une fois posés, ils ne puissent pas bouger sans être soulevés.

Sur le contour de cette structure, un système de puzzle est mis en place afin de pouvoir accueillir de multiple modules supplémentaire, principalement des support de capteurs.

9.1

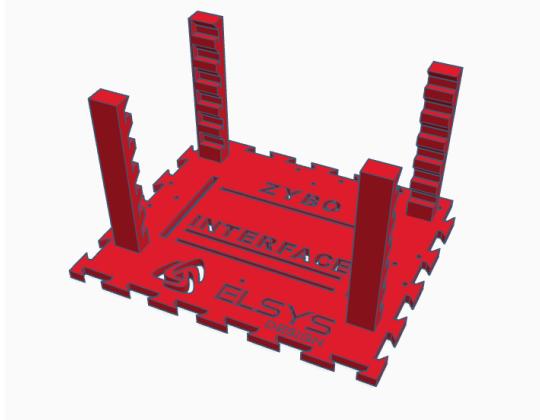


FIGURE 9.1 – Base de la structure mécanique

Une fois cette base imprimée, le support de la batterie et des capteurs ont été réalisés en faisant attention à la taille, l'accessibilité des pins et la position autour du robot si besoin. Par exemple, le module RFID devait se trouver sous le robot tout en étant fixé à cette base. Chaque ultrason a une position spécifique autour du robot. Vous trouverez une visualisation de la structure finale avec les capteurs 9.2.

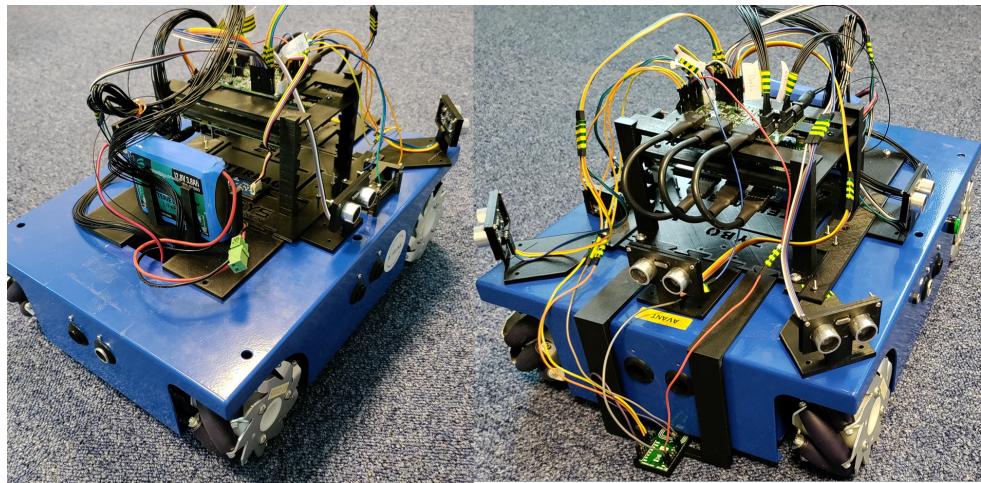


FIGURE 9.2 – Structure mécanique réelle

Cette structure modulable s'est révélée être très pratique pour ajouter facilement tous les composants une fois reçus. Il y a un capteur auquel il aurait fallu penser lors de la réalisation de la base : l'IMU. En effet, ce capteur a besoin de se trouver au centre du robot. Heureusement, plusieurs trous pouvant accueillir des vis avaient été laissées dans la structure. Le capteur a donc pu être fixé même s'il était difficile d'accès sous les cartes d'interface et de développement.

Chapitre 10

Conclusions et perspectives

En l'état, les modules réalisés et intégrés sont : la localisation, la navigation, l'utilisation de la base roulante, du module NFC, des balises et de l'IMU malgré le défaut de bruit. En s'appuyant la carte des locaux, le robot est capable de répondre à une détection d'anomalie et de naviguer en autonomie. D'ici la fin du stage, la correction par trigonométrie va permettre de corriger la composante d'orientation pour plus de fiabilité dans le temps, et l'utilisation des ultrasons va éviter les collisions.

Le cahier des charges de ce projet de robot de surveillance autonome était très ambitieux. Le travail réalisé durant ce stage est conséquent :

- La conception à partir du cahier des charges a permis d'avoir une idée claire du projet, des besoins et des priorités.
- La configuration et mise en place d'un linux embarqué
- La conception et la programmation de la structure logicielle
- La réalisation d'un filtre de Kalman fusionnant IMU et odométrie
- La programmation de ce même filtre et l'exploitation des données capteurs
- La programmation de la carte topologique décrivant les chemins pouvant être parcourus
- La conception et la programmation de l'algorithme de navigation manipulant la localisation, la base roulante et le lecteur RFID
- L'intégration de ces blocs sur le robot
- Les tests et corrections permettant d'obtenir un comportement proche de celui souhaité par le cahier des charges initial
- La conception de la structure mécanique du robot à l'aide d'une imprimante 3D
- La réalisation d'une IHM qui n'a pas été intégrée au robot

Une partie du cahier des charges a été ainsi rempli. L'évitement d'obstacle et le traitement de l'infraction doivent encore être développés.

Ce projet est en cours depuis 3 ans. Il ne sera donc pas reconduit en 2023 et restera en suspens. Un autre sujet de stage sera mis en place dont le fonctionnement sera similaire à notre stage : un stagiaire Hardware, un software et un FPGA réalisant un projet liant FPGA et logiciel sur une carte Hardware. Les travaux réalisés cette année pourront servir de bon point de départ pour la prise en main du projet et de la technologie des cartes de développement avec les outils Xilinx.

Cette période de 6 mois chez Elsys Design m'a permis d'évoluer tant au niveau technique qu'au niveau humain dans un environnement professionnel. Le logiciel pour l'embarqué est un domaine qui me passionne. Le fait de pouvoir faire fonctionner des systèmes en étant très bas-niveau, proche de Hardware, m'intéresse et me motive depuis le début de la quatrième année. Le projet de 4ème année avec la voiture autonome ainsi que ce stage n'ont fait que confirmer l'intérêt que j'y porte. Ce projet a mis en pratique toutes les compétences acquises lors de ma formation : conception, gestion de projet, robotique, filtrage, algorithme de plus court chemin, programmation, multithreading, manipulation de capteurs ou encore les OS pour les systèmes embarqués. Certains de ces enseignements ont même été complétés par des recherches supplémentaires lors du stage. J'ai également acquis de nouvelles compétences telles que la communication entre FPGA et logiciel, le fonctionnement de la technologie RFID, les outils Xilinx et le monde de l'IHM.

Le projet dans sa globalité correspond parfaitement avec mes intérêts de développement logiciel bas-niveau, au sein d'un environnement industriel instructif et formateur tant par les compétences techniques demandées que par la gestion humaine et professionnelle du projet.

Annexes

Annexe 1 : Présentation de la société

Elsys Design est une société d'ingénierie spécialisée dans la conception des systèmes embarqués. Elle a été créée en 2000 par deux ingénieurs en électronique. Aujourd'hui, l'entreprise compte près de 700 ingénieurs dans 11 implantations : 8 en France, ainsi que deux filiales en Serbie et aux États unis. 10.1



FIGURE 10.1 – Implantations d'Elsys Design

Le but d'Elsys Design est de concevoir et de valider des systèmes électroniques et logiciels de pointe. Elle accompagne les projets à forte valeur ajoutée de grandes sociétés internationales, de PME et de start-up.

Conjuguant expertise technique et savoir-faire métier, ses ingénieurs interviennent sur l'ensemble du cycle de développement de systèmes complexes, au sein de nombreuses industries telles que l'aérospatial, l'automobile, la défense, l'énergie, le ferroviaire, l'IoT, le médical ou encore les semi-conducteurs.

ELSYS Design est l'une des filiales d'Advans Group. Trois sociétés composent le groupe :

- ELSYS Design, spécialisée dans les systèmes embarqués
- AVISTO, spécialisée dans le développement logiciel
- MECAGINE, spécialisée dans la mécanique des structures et des systèmes.

Annexe 2 : Roues omnidirectionnelles

Une roue **omnidirectionnelle**, aussi appelée roue **mecanum** ou encore roue **holonomic**, est une roue permettant de se diriger dans toutes les directions. Vous en trouverez une image 10.2.



FIGURE 10.2 – Roue Mecanum

Quand un robot embarque 4 de ces roues, il obtient une liberté de déplacement très importante : il peut se déplacer dans toutes les directions sans changer d'orientation et il peut pivoter sur lui-même, donc changer son orientation en restant sur place. Dans le cadre d'un robot en intérieur, ce type de roue est préféré aux roues classiques.

Bibliographie

- [1] Digilent. Zybo z7 petalinux demo. <<https://digilent.com/reference/programmable-logic/zybo-z7/demos/petalinux>>, 2017. [En ligne ; accédé le 28-Avril-2022].
- [2] Digilent. Petalinux-zybo-z7-20. <<https://github.com/Digilent/Petalinux-Zybo-Z7-20>>, 2021. [En ligne ; accédé le 2-Mai-2022].
- [3] Scott Lobdell. How to implement an inertial measurement unit (imu) using an accelerometer, gyro, and magnetometer. <<http://youtube.com/user/slobdell3>>, 2017. [En ligne ; accédé le 18-Mai-2022].
- [4] Cot Léa. Filtres de kalman. <<http://www.moodle.insa-toulouse.fr>>, 2021. [En ligne ; accédé le 16-Mai-2022].
- [5] Dorfell Parra. Pynq 2.7 for zybo-z7. <<https://gitlab.com/dorfell/>>, 2022. [En ligne ; accédé le 27-Mai-2022].
- [6] PYNQ. Pynq. <<https://github.com/Xilinx/Pynq>>, 2021. [En ligne ; accédé le 8-Mai-2022].
- [7] Sebastian Thrun, Michael Montemerlo, Daphne Koller, Ben Wegbreit, Juan Nieto, and Eduardo Nebot. Fastslam : An efficient solution to the simultaneous localization and mapping problem with unknown data. *Journal of Machine Learning Research*, 4, 05 2004.
- [8] Xilinx. Petalinux tools documentation : Reference guide (ug1144). <<https://docs.xilinx.com/>>, 2022. [En ligne ; accédé le 2-Mai-2022].