UQAC
UNIVERSITÉ DU QUÉBEC
À CHICOUTIMI

ESSAI

PRÉSENTÉ À

L'UNIVERSITÉ DU QUÉBEC À CHICOUTIMI

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INGÉNIERIE

PAR

MAXIME VAIDIS

CONCEPTION, RÉALISATION ET ÉTUDE D'UN ESSAIM DE ROBOTS AUTONOME PROTEGEANT UN GROUPE DE PERSONNES MUNIES DE SEMELLE INTELLIGENTE

DECEMBRE 2017

RÉSUMÉ

Ce projet porte sur le problème de la protection d'un convoi de personnes munies de semelle intelligente par un essaim de robots.

De nos jours, il y a beaucoup de flux de population qui nécessite d'être protégés dans des zones à risques (familles syriennes, irakiennes...). L'essaim de robots est une solution qui permettrait de les protéger sans exposer d'autres personnes au danger. Celui-ci devra suivre le groupe de personnes et éviter toutes les perturbations externes dans le but de réduire les erreurs de positionnement des robots.

La semelle intelligente portée par les gens, élaborée à partir de plusieurs capteurs, donnera les informations sur leur orientation et leur vitesse de marche. Les robots pourront être munis de capteurs de distance et de centrale inertielle afin de détecter les obstacles environnant et de se déplacer autour du groupe de personnes. Un drone fournira également des informations visuelles sur l'environnement autour des personnes. Le système est entièrement basé sur un réseau de modules WiFi (WBAN : Wireless Body Area Network) qui communiqueront toutes les données recueillies.

Un serveur se chargera de collecter toutes les données reçues par les robots et les semelles. Celles-ci seront traitées par différents algorithmes qui dirigeront les robots de manière autonome autour des personnes.

REMERCIEMENTS

Je tiens à remercier toutes les personnes qui ont contribué au succès de ma maîtrise et de mon stage de fin d'étude.

Tout d'abord, j'adresse mes remerciements à mon maître de stage, Mr Martin Otis professeur au département des sciences appliquées de l'Université du Québec à Chicoutimi, qui m'a beaucoup aidé dans l'élaboration de mon projet de stage. Ses conseils m'ont permis de développer mon travail et de trouver en ce stage un domaine de recherche en totale adéquation avec mes attentes.

Je remercie également tous mes collègues du laboratoire LAIMI (Laboratoire d'Automatisme et d'Interaction 3D Multimodale Intelligente) pour leurs conseils et les nombreux échanges constructifs que l'on a eu.

Enfin, je tiens à remercier ma famille, en particulier mon père, ma mère et ma sœur pour leur soutien lors de mes études. Je remercie également mes amis qui ont contribué de près ou de loin à ce travail.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

CHAPITRE 1

INTRODUCTION

Un grand nombre de personne se déplacent aujourd'hui pour des raisons économiques, vis-à-vis de la situation politique de leur pays, ou encore étant donné l'émergence des pays en développement, ce qui conduit ces millions de personnes sur les routes. Cependant dans certaines régions du monde, le fait de voyager est dangereux, surtout dans les pays en guerre. Là-bas, les gens se déplacent majoritairement à pied avec peu de moyens de protection, s'exposant ainsi à des situations dangereuses. Dans le but de protéger ces personnes, nous devrions être capables de collecter des informations sur leur déplacement ainsi que sur l'environnement les entourant. Plusieurs études ont été menées avec l'idée de récupérer ces informations via différents réseaux de capteurs [1-4]. Ces réseaux de capteurs utilisent dans la plupart des cas une communication WiFi [1-3], mais ils peuvent utiliser d'autres protocoles comme la communication Bluetooth ou le protocole NFC (Near Field Communication). Ces modes de communications sont relativement simples à mettre en œuvre et peuvent être très utiles pour transmettre des données entre plusieurs systèmes robotiques. Les données recueillies peuvent par la suite être utilisées pour protéger le groupe de personnes par des robots. L'automatisation d'un tel système permettrait d'améliorer son efficacité et de réduire la présence humaine dans des situations à risques, ce qui sur de longues distances est un atout majeur dans ce type de projet.

Dans des contextes assez proches de ce projet, beaucoup d'études ont été réalisées sur l'interaction d'un groupe de robots mobiles avec des facteurs externes et sur leurs impacts dans la prise de décision collective [5-6]. Nous aurons beaucoup de personnes au sol et nos robots doivent être capables de traquer toutes ces personnes, dans le but de trouver une disposition s'adaptant au groupe de personnes et de les suivre pour les protéger avec un maximum de mesures de sécurité tout au long de leur trajet. Le système devra agir en trois étapes : analyser les données sur le déplacement des personnes et donner une position cible à atteindre pour chaque robot, planifier la trajectoire du robot vers cette position, et donner un vecteur de déplacement à chaque robot conforme au mouvement du groupe de personnes. Pour réaliser cela, nous utiliserons des semelles intelligentes qui nous donnerons des informations sur le déplacement de

chaque personne, dans le but de déduire un vecteur moyen de déplacement que permettra aux robots de suivre le groupe et de le superviser.

Le but de cet essai est de présenter notre réseau de module WiFi (WBAN : Wireless Body Area Network) utilisé avec nos semelles intelligentes et les différents algorithmes crées pour contrôler l'essaim de robots. Avec les données collectées de ce réseau, nous souhaitons identifier certains patterns fiables qui nous permettront de suivre un groupe de personnes dans le but de mettre en place une protection autour d'elles durant leur trajet, anticipant ainsi les situations à risques ou pour éviter la mobilisation de militaire dans des zones dangereuses. Le système que nous présentons fonctionne sous Ubuntu 14.04 avec l'utilisation de ROS (Robot Operating System) [7]. L'organisation de cet essai est le suivant : le chapitre 2 décrit l'état de l'art des différents algorithmes existants pour la création d'enveloppe convexe à partir de nuage de points, mais également tous les algorithmes permettant de planifier la trajectoire d'un robot avec ou sans évitement d'obstacles en mouvement. Une présentation des différents systèmes de réseau de capteurs sera aussi effectuée. Le chapitre 3 portera sur la conception du système, notamment en décrivant le matériel utilisé ainsi qu'en expliquant en détail le fonctionnement des différents algorithmes. Nous expliquerons également comment l'on peut détecter différents états du robot et comment le système réagi en fonction de ceux-ci. Le chapitre 4 montrera les différents résultats obtenus à travers divers expérimentations. Nous finirons sur une conclusion de nos travaux dans le chapitre 5.

CHAPITRE 2

ÉTAT DE L'ART

2.1 INTRODUCTION

En robotique, il existe de nombreux domaines qui utilisent des algorithmes afin de contrôler des robots. Les principaux domaines sont la robotique industrielle, la robotique domestique, la robotique médicale, la robotique militaire et la robotique scientifique. La robotique industrielle possède un grand nombre de robots sur des chaînes d'assemblage. Ceux-ci sont le plus souvent à base fixe et reproduisent uniquement une seule tâche. En robotique domestique, nous retrouvons les robots effectuant des tâches ménagères comme par exemple les robots aspirateurs ou ceux de surveillance. Dans le domaine médical, ce sont majoritairement des robots chirurgiens qui existent sur le marché. Ils servent d'assistance lors des opérations. Les militaires utilisent également beaucoup de robots. Les drones et les robots autonomes servent énormément pendant des opérations militaires. Ils peuvent récolter des informations sur l'ennemi ou le terrain, afin de fournir un avantage tactique aux soldats. La robotique scientifique permet l'exploration de notre planète (fonds océaniques, forêts,…) mais également de notre système solaire avec l'envoi de sondes spatiales et de robots comme par exemple sur la Lune et sur Mars.

Notre projet s'inscrit dans le domaine militaire puisque notre système doit pouvoir être opérationnel dans des endroits à risques, et dans des zones de guerres. La plupart des systèmes robotiques militaires servent à la collecte d'informations. Seul une minorité est programmée pour le combat, étant donné qu'ils ne sont pas encore totalement opérationnels et que la législation de l'ONU n'a pas encore traité ce sujet afin de valider de telles armes de destruction fonctionnant de manière autonome. Ces robots autonomes ont la capacité de se mouvoir dans un environnement assez incertain et peuvent s'adapter à des conditions difficiles. Cependant, aucun n'est encore capable de protéger un groupe de personnes dans des zones de combats.

Dans cette partie, nous décrirons les recherches déjà menaient sur lesquelles nous nous sommes basées pour notre projet. Nous ferons principalement un bilan des recherches effectuées sur l'algorithmique géométrique d'un ensemble de points dans le but d'en extraire son enveloppe convexe, ainsi que sur les différentes façons de planifier la trajectoire d'un robot. La dernière partie portera sur les travaux réalisés sur les essaims de robots.

2.2 ALGORITHMIQUE GEOMETRIQUE POUR ENVELOPPE CONVEXE

En algorithmique géométrique, il existe plusieurs manières d'avoir l'enveloppe convexe d'un ensemble de point. Des algorithmes existent pour la 2D, la 3D, ou d'autres dimensions. Dans le cadre de notre projet, nous nous intéresserons uniquement au cas 2D. Les parties qui suivent présenteront les différentes méthodes utilisées. Nous vous présenterons ici les plus connues.

2.2.1 MARCHE DE JARVIS (GIFT WRAPPING ALGORITHM)

L'algorithme de la marche de Jarvis est un algorithme utilisé seulement en 2D, publié en 1973 par R. A. Jarvis [8]. Il possède une complexité d'ordre $O(N.H)$, où N est le nombre de points et H est le nombre de points sur l'enveloppe convexe. En comparaison des autres algorithmes pour le traitement temps réel, cet algorithme est préférable pour un N faible ou quand H est petit devant N. En règle générale, on utilise peu celui-ci car les autres possèdent de meilleures performances.

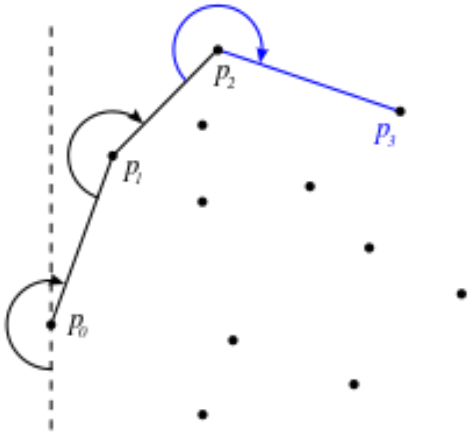


Figure 1. Fonctionnement de l'algorithme de la marche de Jarvis [47]

L'algorithme fonctionne pour un nombre minimum de points supérieur ou égal à trois, tant que les points ne sont pas tous colinéaires entre eux. Il démarre à partir d'un point connu pour être sur l'enveloppe convexe, le point le plus à gauche de l'ensemble de point de départ p_0. Il choisit le point suivant p_{i+1} en fonction des angles polaires des autres points, angles qui sont définis à partir d'un repère polaire dans lequel le point p_i en est le centre. L'algorithme continue ainsi jusqu'à atteindre le point de départ.

2.2.2 LE PARCOURS DE GRAHAM (GRAHAM SCAN ALGORITHM)

L'algorithme du parcours de Graham a été publié en 1972 par Ronald Graham [9]. Il est d'une complexité en $O(N.log(N))$ où N est le nombre de points de l'ensemble de départ. La première étape de cet algorithme est de trouver le point le plus en bas à gauche p_0 de l'ensemble de point. Ensuite l'ensemble des point est trié en fonction de l'angle que forme chaque point avec l'axe des abscisses relativement par rapport à p_0. Ce tri est mémorisé dans un tableau T. Une fois le tri réalisé, l'algorithme prend successivement la séquence de trois points du tableau T et forme deux couples avec. Par la suite, il cherche à savoir si le fait de passer d'un couple à l'autre est le fait de 'tourner à gauche' ou 'tourner à droite'. Si c'est le fait de 'tourner à droite', le point pivot (deuxième des trois points) est retiré de T car il ne fait pas partie de l'enveloppe convexe. Ce processus s'arrête quand on revient au point initial. Le tableau T contiendra alors tous les points de l'enveloppe convexe dans le sens trigonométrique.

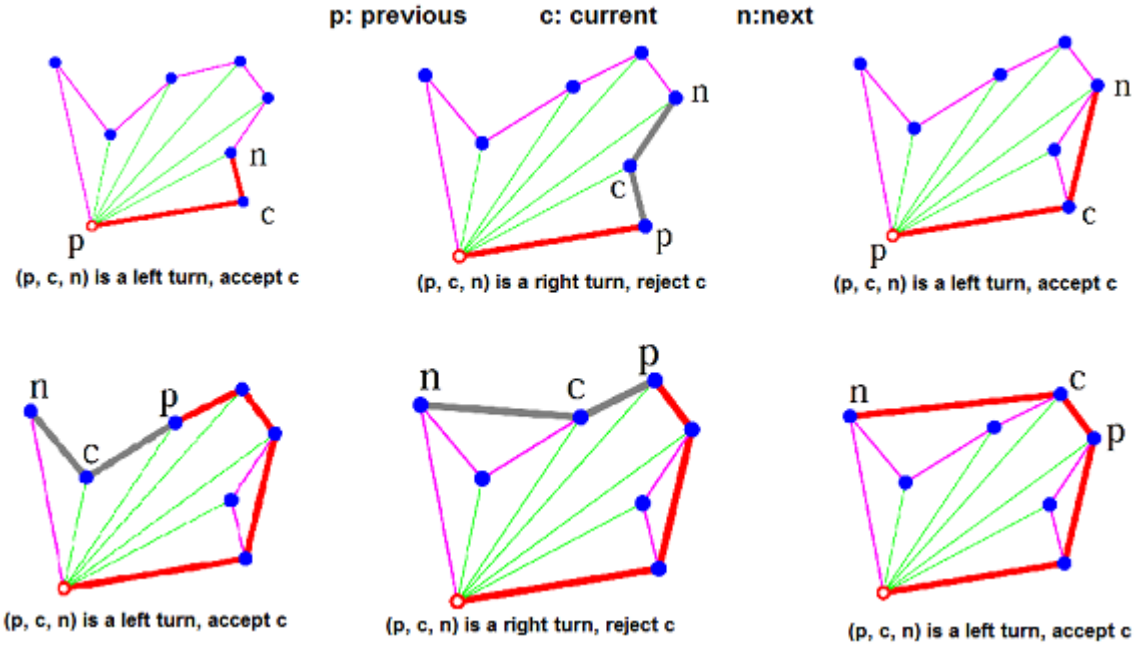


Figure 2. Fonctionnement de l'algorithme du parcours de Graham [48]

2.2.3 L'ALGORITHME DE CHAN

L'algorithme de Chan est un algorithme qui permet de calculer l'enveloppe convexe d'un ensemble de point en 2D ou en 3D, publié en 1996 [10]. Sa complexité est $O(N.logH)$ où N est le nombre de point de l'ensemble et H le nombre de point sur l'enveloppe convexe. En 2D, il utilise un algorithme de complexité $O(N.logN)$ (par exemple l'algorithme du parcours de Graham) avec la Marche de Jarvis.

L'algorithme commence par partitionner l'ensemble de point Q en $1 + \frac{N}{m}$ sous ensemble Q_i avec au plus m points dans chaque sous-ensemble. Le nombre m est idéalement le nombre de point situé sur l'enveloppe convexe H si on le connaît d'avance. Si l'on ne le connait pas, on peut commencer par prendre $m = 2$ puis continuer à chaque itération avec des valeurs de m au carré sans dépasser le nombre N (2, 4, 16, 256, …). Sur chaque sous-ensemble Q_i, on calcule leur enveloppe convexe avec l'algorithme du parcours de Graham par exemple. Une fois que cela est fait, on applique la Marche de Jarvis à l'ensemble des enveloppes convexes trouvées. Si $m < H$, le calcul ne sera pas fini. Il faudra changer m et refaire une itération. Si $m > h$, on obtient l'enveloppe convexe.

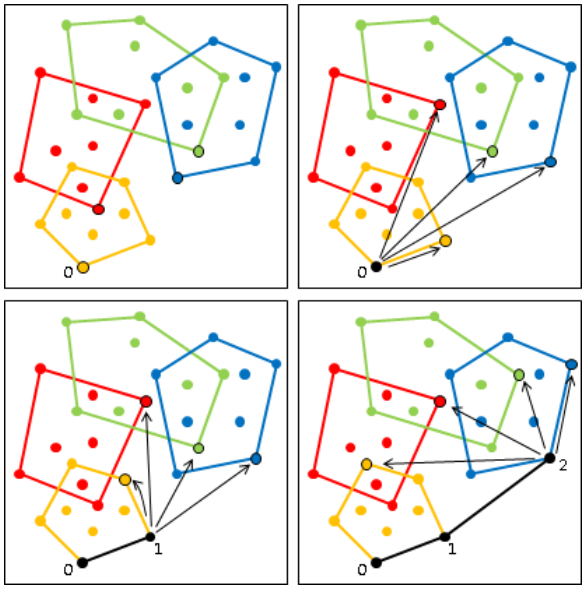


Figure 3. Fonctionnement de l'algorithme de Chan [49]

2.2.4 L'ALGORITHME QUICKHULL

L'algorithme Quickhull est une autre méthode qui permet de trouver l'enveloppe convexe d'un nombre fini de points dans un plan mais aussi dans des dimensions supérieures, publié en 1996 [11]. Il utilise la méthode 'diviser pour mieux régner'. Sa complexité est considérée comme $O(N.logN)$ où N est le nombre de point de l'ensemble de départ. En général l'algorithme possède de bonne performance. Cependant si l'ensemble de point possède beaucoup de symétrie ou se rapproche d'un cercle, son temps d'exécution peut augmenter énormément.

L'algorithme s'effectue en plusieurs étapes. Tout d'abord on trouve les points se situant sur le minimum et maximum des abscisses. Ces deux points seront toujours situés sur l'enveloppe convexe. On forme ensuite une droite à partir de ceux-ci. Cela divisera l'ensemble de points en deux, qui seront traités de façon récursive. On cherche ensuite le point d'un côté de la ligne qui aura le plus de distance vis-à-vis de la droite. Ce point est lui aussi situé sur l'enveloppe convexe. Les trois points forment un triangle. Les points qui se situent dans ce triangle ne font pas partie de l'enveloppe convexe. Pour trouver les autres points y appartenant, on poursuit le processus de recherche du point le plus distant avec les nouvelles droites crées. L'algorithme s'arrête quand il n'y a plus de point à traiter.

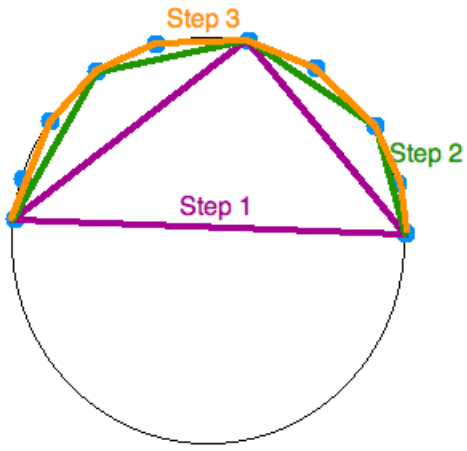


Figure 4. Fonctionnement de l'algorithme Quickhull [50]

2.3 PLANIFICATION DE TRAJECTOIRE

La planification de chemin/trajectoire d'un robot est un problème assez récurrent en robotique. Il existe plusieurs approches en fonction de la situation rencontrée. La plus simple des planifications de chemin se fait dans le cadre d'un environnement statique. Le problème est ainsi réduit à un aspect géométrique dans lequel toute trajectoire ne présentant pas de collision avec l'environnement du robot est une solution acceptable. Dans ce contexte, généralement il faut calculer le chemin le plus court entre le point de départ et le point à atteindre. Deux grandes familles de méthodes de résolution se présentent :

- L'approche globale (déterministe) : le but est de trouver un chemin complet du robot vers sa cible avant le début de son déplacement. Pour cela, nous nous basons sur une connaissance aussi complète que possible de l'environnement du robot.
- L'approche locale (réactive) : ce cas se présente lorsque nous ne disposons pas d'assez d'informations sur l'environnement du robot. Il n'est alors pas possible de concevoir un trajet complet du robot vers sa cible avant son déplacement. Les méthodes de planification doivent donc prendre en compte de façon dynamique les données nouvelles sur l'environnement de travail et construire au fur et à mesure la trajectoire de déplacement du robot.

Nous présenterons ci-dessous quelques méthodes principalement utilisées en robotique, d'abord dans les cas où les obstacles sont statiques puis ensuite dynamiques.

2.3.1 APPROCHES GLOBALES DANS UN ENVIRONNEMENT STATIQUE

2.3.1.1 LES GRAPHES

Ces méthodes de résolutions visent à analyser la topologie des espaces sans obstacles dans le but de simplifier le problème de recherche d'un plus court chemin dans un graphe entre le point de départ initial et le point de destination final. Elles s'effectuent donc en deux étapes : la première étape consiste à créer le graphe de l'environnement à analyser. La seconde étape est de parcourir le graphe afin de trouver le plus court chemin. Celle-ci utilise des algorithmes assez connus comme ceux de Bellman et Dijkstra. Dans cette partie nous présenterons les différents algorithmes de construction du graphe de l'environnement à étudier.

2.3.1.1.1 GRAPHE DE VISIBILITE

Le graphe de visibilité est une idée de Nilsson publié en 1969 [12], qui est fondé sur le principe que le chemin le plus court à effectuer dépend des sommets des obstacles polygonaux situés dans l'environnement de travail à étudier. Le graphe est construit de manière à relier chaque sommet d'un obstacle entre eux par un segment. Cette méthode est peu employée puisqu'elle autorise le contact entre le robot et les obstacles.

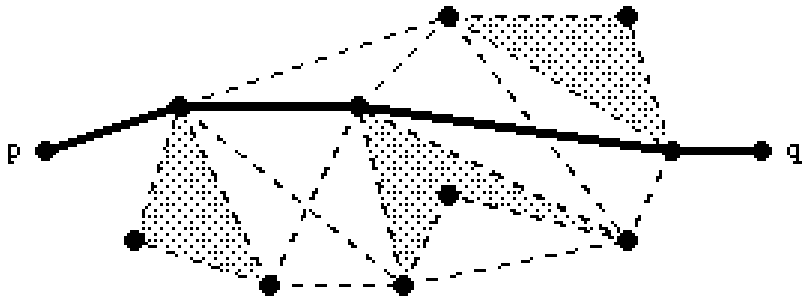


Figure 5. Exemple de graphe de visibilité avec p le point de départ et q le point d'arrivé [51]

2.3.1.1.2 DIAGRAMME DE VORONOÏ

Un autre moyen de construire un graphe est d'utiliser un diagramme de Voronoï (1989) [13], qui par rapport aux graphes de visibilité permettent d'éviter tout contact avec les obstacles. Le graphe se construit en traçant des segments d'égales distances aux obstacles et aux bords de l'environnement connu. Les segments tracés sont ensuite reliés au point initial et au point de destination.

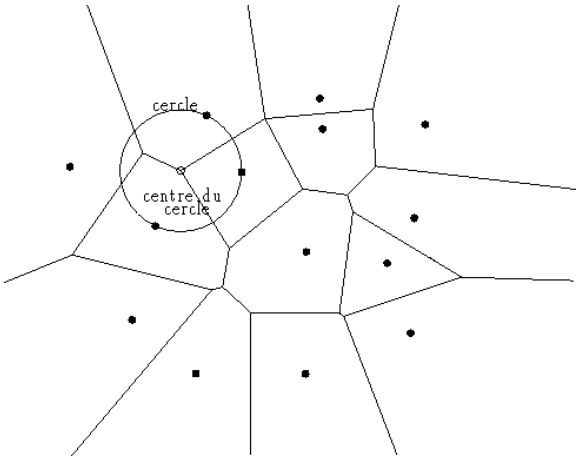


Figure 6. Diagramme de Voronoï [52]

2.3.1.1.3 DECOMPOSITION CELLULAIRE

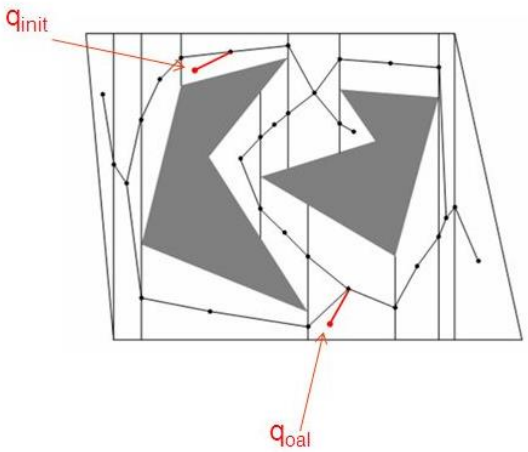


Figure 7. Exemple de décomposition cellulaire [59]

La méthode de décomposition cellulaire s'effectue en découpant l'environnement connu en un ensemble de cellules adjacentes. Deux catégories de découpages existent : les méthodes exactes et les méthodes approchées. Les méthodes exactes font un découpage qui recouvre entièrement l'environnement connu tandis que les méthodes approchés tentent de se rapprocher de celui-ci avec des formes simples comme des triangles par exemple. Une fois le découpage effectué, le barycentre de chaque cellule est calculé et ceux-ci sont reliés entre eux pour former le graphe à étudier.

2.3.1.1.4 RAPIDLY-EXPLORING RANDOM TREES (RRT)

Cette méthode de construction de graphe a été proposée par Lavalle en 1998 [14]. C'est l'une des méthodes les plus utilisées au cours des dernières années pour élaborer des GRAPHE. Le graphe est construit à partir d'un arbre qui prend comme racine initiale le point de départ du robot. L'arbre 'grandit' progressivement dans l'environnement connu autour du robot. Cette croissance se fait par pas temporel et en choisissant de manière arbitraire un chemin non bloqué par un obstacle. Ce processus continue jusqu'à ce que le point de destination soit atteint.

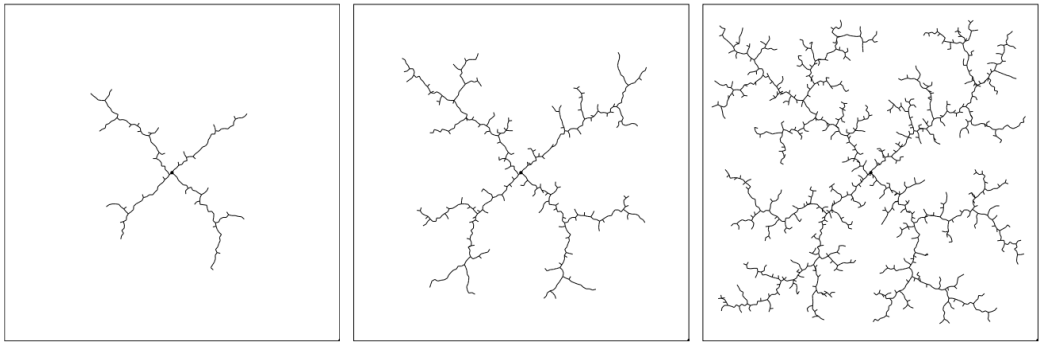


Figure 8. Exemple de développement d'un arbre de l'algorithme RRT [53]

2.3.1.2 LES METHODES HEURISTIQUES

2.3.1.2.1 ALGORITHMES GENETIQUES

Cette méthode a été créée dans les années 60 par John Holland. Elle est encore très utilisée de nos jours [15]. Elle se base sur le processus d'évolution naturelle suivant le modèle de Darwin. Ils sont très utilisés dans le domaine de l'optimisation et la planification de mouvement. L'algorithme fonctionne en simulant des solutions à chaque itération. Celles-ci sont représentées par une population d'individus possédant des gènes. A chaque itération, seul les gènes qui optimisent la fonction coût du trajet sont transmis à la nouvelle population et ainsi de suite jusqu'à ce qu'il reste uniquement les gènes de la fonction coût minimale. Cette population sera alors le trajet recherché.

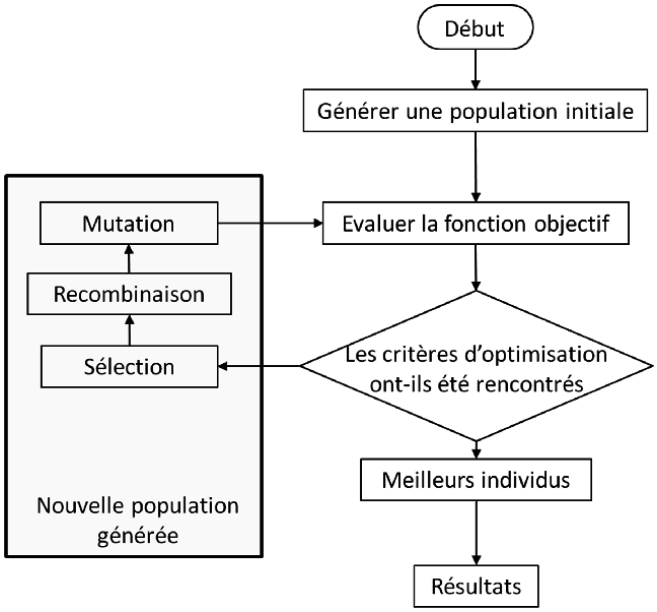


Figure 9. Organigramme d'un algorithme génétique [54]

2.3.1.2.2 RECUIT SIMULE

Cette méthode stochastique a été créée par Kirkpatrick et al. en 1983 [16]. Elle reprend le principe du recuit des matériaux solides. Le réchauffement à très haute température et le refroidissement lent d'un matériau réorganise les atomes de celui-ci en un état plus stable, i.e d'énergie plus faible. L'algorithme commence par le déplacement d'une particule contenant une solution possible du problème. Au début lors du chauffage, la particule possède un chemin quasi-aléatoire allant vers la solution, le coût de celle-ci étant alors très élevée. Cela permet d'explorer l'environnement à analyser. Au fur et à mesure que la température baisse, le chemin devient de moins en moins aléatoire et la particule se dirige vers la solution avec un coût minimum. Cet algorithme permet d'éviter les minimas locaux et de converger vers le minimum global. Cependant il n'existe pas de solution exacte au problème. On obtient uniquement une solution approchée avec une vitesse de convergence plus faible près du minimum global. Cette méthode est souvent utilisée pour compléter d'autres algorithmes.

2.3.2 APPROCHES LOCALES DANS UN ENVIRONNEMENT STATIQUE

2.3.2.1 CHAMP DE POTENTIEL

Cette méthode se base sur l'approche des champs de potentiels en physique. Ce sont ceux-ci qui guideront le robot vers sa destination finale dans son environnement. Cet algorithme a été initialement proposé par Oussama Khatib en 1985 [17]. Il fonctionne de la manière suivante : les obstacles sont représentés par des champs de potentiels répulsifs afin d'éviter tout contact avec ceux-ci. L'objectif est lui représenté par un champ de potentiel attractif. Le robot est vu comme une particule plongée dans un champ de potentiel équivalent à la somme du champ répulsif et du champ attractif. Le déplacement du robot est calculé de manière itérative. Celui-ci s'effectue suivant une descente du gradient. Cette méthode est très simple d'utilisation et son coût de calcul est faible. Cependant elle possède plusieurs limitations. Le robot peut être pris dans des minima locaux qui le paralyseront. Des mouvements d'oscillations peuvent aussi apparaître et l'algorithme ne permet pas de passer entre deux obstacles assez proches. Afin de corriger ces défauts, plusieurs méthodes complémentaires ont été mises en place comme par exemple le suivi des murs pour éviter le problème d'oscillations.

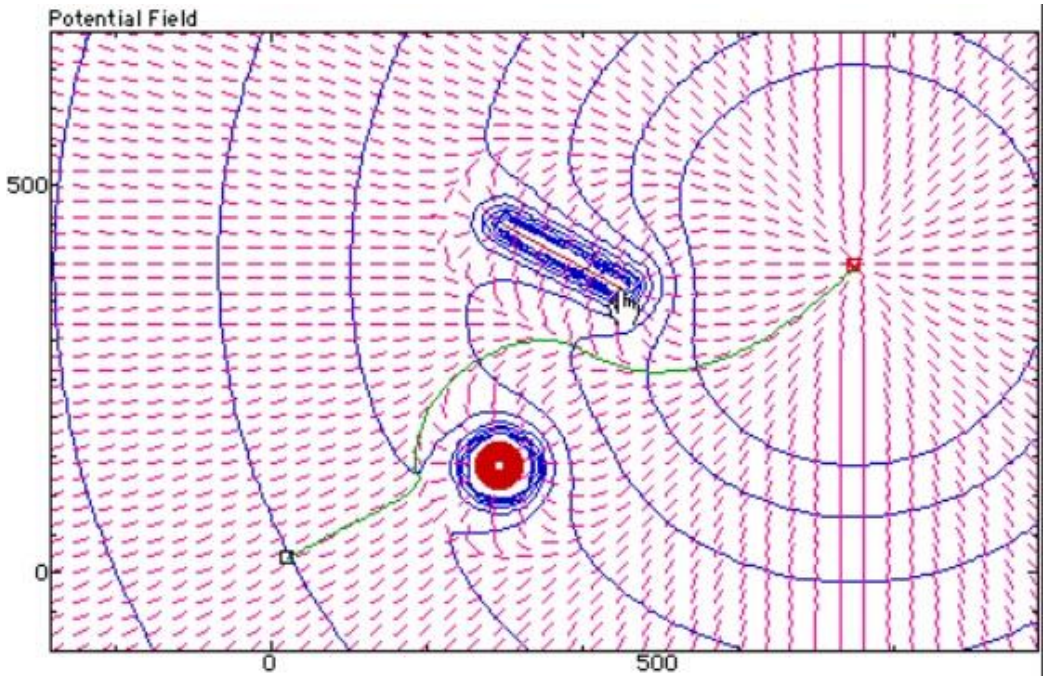


Figure 10. Exemple d'application du champ de potentiel [55]

2.3.2.2 FENETRE DYNAMIQUE

Cette méthode fut proposée par Dieter Fox et al. en 1997 [18]. Elle permet l'évitement d'obstacles en temps réel grâce à la configuration des commandes du robot. Elle travaille sur le domaine des vitesses possibles du robot, celles qui permettent d'éviter les obstacles. Dès que l'environnement du robot est analysé, l'algorithme procède à une optimisation des commandes des moteurs en fonction de différents éléments comme le temps de parcours, l'optimisation de la vitesse, la distance aux obstacles, etc. Le résultat est un couple de valeur (V ;W) correspondant à la vitesse de translation et de rotation du robot. L'avantage de cette méthode est qu'elle prend en considération la cinématique du robot. Néanmoins, elle est difficilement intégrable dans un système multi-robot et ne fonctionne pas pour des obstacles en mouvement.

2.3.2.3 DIAGRAMME DE PROXIMITE

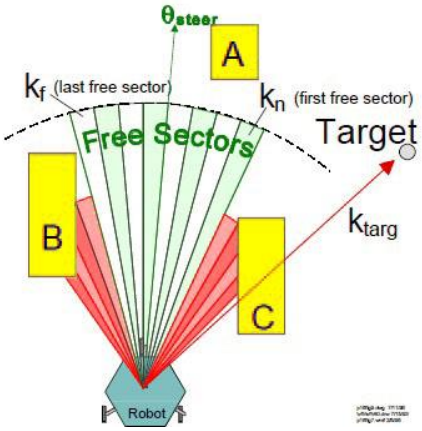


Figure 11. Diagramme de proximité [56]

Cette méthode de planification de trajectoire crée par Minguez et Montano (2000) [19] porte sur un diagramme de proximité des obstacles mis à jour continuellement lors du déplacement du robot. La trajectoire choisit est celle où il n'y a pas d'obstacles détectés et également la distance la plus proche par rapport à l'objectif à atteindre. Plusieurs comportements du robot peuvent alors être implémentés. Il peut avancer en ligne droite vers l'objectif, passer entre deux obstacles ou les contourner. La vitesse donnée dépendra de la distance par rapport à

l'obstacle. Cette méthode peut être adaptée à la morphologie du robot. Cependant elle ne considère pas la cinématique du robot et la convergence vers la destination n'est pas assurée.

2.3.2.4 LOGIQUE FLOUE

La logique floue est un concept qui a été posé en 1965 par Lot Fi Zadeh. Elle se base sur la théorie des ensembles mathématiques flous qui parlent du concept d'ensembles définies de manières imprécises. Elle convertie des grandeurs physiques en des variables linguistiques et procède à des calculs sur celles-ci. Par exemple, lorsque nous parlons d'une distance à un obstacle en physique celle-ci est donnée en mètres. En logique floue, c'est nous qui définissons des variables telles que 'proche de l'obstacle', 'loin de l'obstacle', etc. La planification de trajectoire va se baser sur ces variables linguistiques via des règles précises pour générer une commande qui sera par la suite reconvertie en variables physiques. L'algorithme de logique floue est composé de quatre blocs : un bloc de fuzzification qui transforme les variables physiques en variables linguistiques. Un deuxième bloc qui se compose d'une bibliothèque de règles à appliquer sur ces variables linguistiques. Un autre bloc se charge de donner la commande en fonction de la bibliothèque des règles. Le dernier bloc converti la commande linguistique en variables physiques.

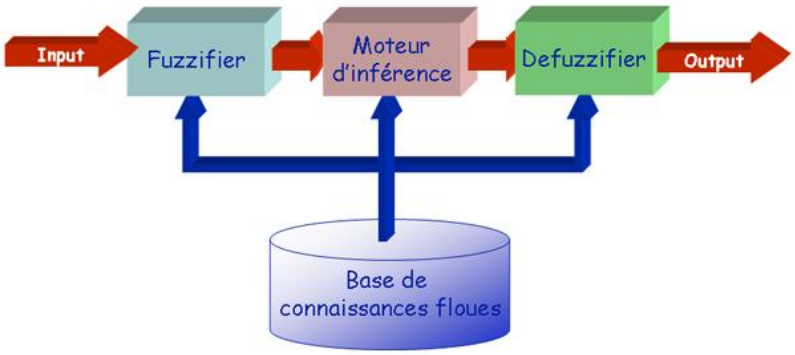


Figure 12. Fonctionnement d'un algorithme de logique floue [57]

Cette méthode possède plusieurs avantages. Elle permet de traiter des informations imprécises et de les exploiter en temps réel. De nombreux comportements du robot peuvent être implantés dans le système via la bibliothèque des règles, en majorité des décisions du type perception/action. L'implémentation est également relativement facile. Néanmoins la convergence globale vers la destination n'est pas assurée. En effet des problèmes de minima locaux peuvent intervenir et ainsi que le fait de ne pas avoir toutes les informations complètes sur l'environnement du robot.

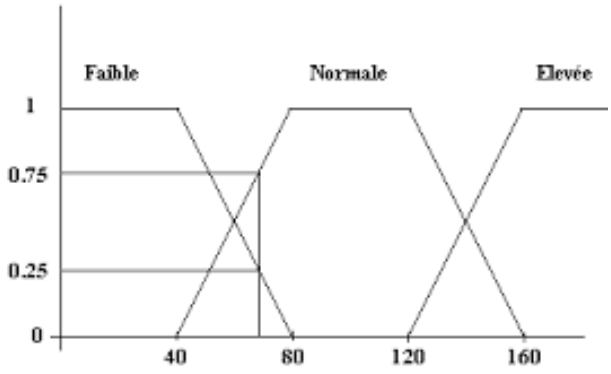


Figure 13. Exemple de variable linguistique utilisée en logique floue [58]

2.3.3 NAVIGATION AVEC TRAJECTOIRE DE REFERENCE DANS UN ENVIRONNEMENT DYNAMIQUE

2.3.3.1 APPROCHE BASEE SUR L'ESPACE DES ETATS-TEMPS

Cette méthode des états-temps a été mise au point par Erdmann et Lozano-Perez en 1987 [20]. Le principe de celle-ci est de rajouter une dimension temporelle à l'environnement du robot. De ce fait les régions possédant des obstacles dynamiques se transforment en régions statiques qui seront évitées par le robot. L'ajout de cette dimension temporelle augmente beaucoup la complexité de l'algorithme de planification de chemin ainsi que la difficulté de trouver le bon chemin sur le graphe.

2.3.3.2 BANDES ELASTIQUE DE KHATIB

Khatib a proposé initialement cette solution pour des robots holonomes en 1993 [21]. Pour fonctionner, l'algorithme a besoin d'une trajectoire de référence initialement calculée. Celle-ci est représentée par un ensemble de disques en 2D ou de boules en 3D. Cela permet de considérer la trajectoire comme étant élastique, sur laquelle des forces répulsives seront générées par les obstacles en mouvement, mais aussi des forces attractives entre chaque disques/boules afin de maintenir la bande tendu. Pour ce faire, la taille et la position des différents disques peuvent changées. L'une des limitations de cette méthode est que l'on ne connait pas toujours la distance entre le système robotique et les obstacles.

2.3.3.3 DEFORMATION VARIATIONELLE DE LAMIRAUX

Cette méthode d'évitement de collision avec des obstacles dynamiques fut proposée par Lamiraux et al. en 2004 [22]. Son principe est de modifier la trajectoire déjà planifiée du robot pendant son déplacement pour éviter les obstacles. L'algorithme change les entrées du système de façon à simuler un potentiel répulsif lorsque le robot est proche d'un obstacle et d'en diminuer la force lorsqu'il s'en éloigne.

2.3.3.4 ROADMAP ELASTIQUE

Cette méthode crée par Yang et Brock en 2006 [23] est appliquée pour les robots mobiles de types manipulateurs. Une roadmap de l'environnement du robot est créé puis sa trajectoire est calculée par l'algorithme A* (algorithme de recherche d'un plus court chemin dans un graphe) pour planifier un chemin entre le point initial et le point final. Ce chemin est contrôlé par la suite afin de vérifier qu'il n'y a pas de collision avec des obstacles. Lors du déplacement du robot, les nœuds du graphe près des obstacles sont mis à jour et le chemin est planifié une nouvelle fois.

2.3.3.5 DEFORMATION DE TRAJECTOIRE DE KURNIAWATI ET FRAICHARD

Les méthodes que l'on a vu précédemment se basent toutes sur une déformation de la trajectoire suivi par le robot, à l'exception de la déformation de Lamiraux dont les vitesses et accélérations peuvent être manipulées. Néanmoins les travaux de Lamiraux sont assez coûteux en temps de calcul. Une limitation de ces méthodes est qu'elles créent des trajectoires parfois impossibles à faire suivre au robot du fait de la dynamique de l'obstacle. Les contraintes cinématiques peuvent être trop importantes (par exemple l'angle de braquage maximal d'un robot peut être dépassé). Il est parfois préférable de laisser passer un obstacle devant le robot puis de continuer son chemin derrière lui. En se basant sur ces problèmes, Kurniawati et Fraichard ont proposé leur solution en 2007 [24]. Le robot peut alors adapter sa cinématique en anticipant le déplacement des obstacles. Malgré tout, cet algorithme ne peut respecter des contraintes non-holonomes et se limite donc à être appliqué seulement à des systèmes de type de masse ponctuel.

2.3.3.6 DEFORMATION DE TRAJECTOIRE DE DELSART ET FRAICHARD

Cette méthode est la continuité de la précédente, qui a pour but de l'adapter au système robotique avec des contraintes cinématiques et dynamiques. Ce travail a été réalisé par Delsart et Fraichard en 2008 [25]. Une étude approfondie de la dynamique de l'obstacle permet de mieux anticiper ses mouvements et donc de mieux les éviter.

2.3.4 NAVIGATION SANS TRAJECTOIRE DE REFERENCE DANS UN ENVIRONNEMENT DYNAMIQUE

2.3.4.1 METHODE DE NAVIGATION BASEE SUR LES ETATS DE COLLISIONS INEVITABLES

Cet algorithme permet de sécuriser le déplacement d'un robot dans un environnement dynamique, sans qu'il y ait besoin de calculer une trajectoire de référence. Il a été conçu par Fraichard et développé par la suite (2003 [26], 2007, 2008 et 2009). Un état de collisions inévitables est décrit comme étant un état dans lequel quel que soit la trajectoire future du robot, celui-ci entrera en collision avec un obstacle. Pour sécuriser son chemin, il faut alors éviter que celui-ci entre dans un état de collision, mais également éviter les états qui créent des états de collisions inévitables. Cette méthode dépend très fortement des prévisions de déplacement des obstacles dans l'environnement du robot.

2.3.4.2 CHAMPS DE POTENTIEL ET RECUIT SIMULE

La combinaison des champs de potentiels et du recuit simulé est une approche créée par Zhang et al. en 2004 [27]. Elle propose une solution au problème de la non-convergence vers la destination finale lorsque la destination finale est située près des obstacles, ainsi que pour résoudre le problème des minima locaux.

2.3.4.3 THEORIE DES ECOULEMENTS A POTENTIEL DE VITESSE ET OPTIMISATION PAR ESSAIMS PARTICULAIRES

Cet algorithme se base sur l'optimisation par essaims particulaires et la théorie des écoulements pour contrôler le robot dans son environnement. Il a été publié par Hu et al. en 2007 [28]. Il permet d'éviter des obstacles sans chemin de référence. Cependant l'algorithme ne prend pas en compte les changements soudains de vitesse des obstacles.

2.3.4.4 CHAMPS DE POTENTIEL ET ALGORITHME D'EVOLUTION BACTERIEN

Les limitations dues à la méthode du champ de potentiel peuvent être réduites fortement par un algorithme couplant cette méthode et un algorithme génétique. C'est ce que les travaux de Montiel et al. ont montré en 2015 [29]. Ce nouvel algorithme peut trouver les valeurs optimales des différents champs de potentiels attractifs et répulsifs, permettant d'éviter notamment le problème des minima locaux. Néanmoins les contraintes cinématiques et dynamiques du robot ne sont pas prises en compte.

2.4 ESSAIM DE ROBOTS

Les unités de mesures inertielles (combinaisons d'accéléromètres, gyroscopes et magnétomètres) sont assez largement utilisées pour la mesure de déplacement et de tracking, permettant après analyse d'obtenir la position et la cinématique de systèmes robotique. Des recherches ont été menées dans le but d'aider un opérateur à sélectionner un robot en particulier à distance, amenant ainsi une simplification de l'interaction entre cet opérateur et l'essaim de robots [30]. Dans ce cas nous devons choisir avec attention le moyen de communication pour avoir de meilleurs résultats [31]. Cependant dans le but de contrôler et simplifier les interactions entre l'opérateur et les robots, nous pouvons programmer deux types de robot. L'un aurait le rôle de leader du groupe, donnant ainsi les ordres et récoltant les informations sur les autres robots. Les autres exécuteraient les ordres donnés par le leader et lui transmettraient les informations dont il aurait besoin pour contrôler l'essaim [32-35].

Le contrôle de l'essaim de robots est possible en utilisant le mouvement de l'opérateur ce qui permet de sélectionner un robot un par un, chaque robot pouvant effectuer des tâches plus ou moins compliquées en fonction de son degré de liberté. Certains robot pourront suivre une

direction, faire des actions simples, tandis que les autres auront d'autres objectifs assignés [36-37].

Comme nous l'avons vu, beaucoup de recherches ont été effectuées sur le management des robots en essaim, surtout sur leur évolution dans des environnements complexes comme par exemple des montagnes, des vallées et avec la présence de nombreux obstacles qui pourraient entrer en collision avec l'essaim de robots [5-6]. De plus si l'un des robots est endommagé, cela n'aurait pas d'impact négatif sur l'ensemble de l'essaim, permettant ainsi à l'opérateur de poursuivre sa mission sur le terrain ce qui peut être très utile en terrain inconnu. Les algorithmes prennent en compte plusieurs paramètres comme la physique de déplacement du robot pour avoir de meilleurs résultats lors des simulations ou sur le terrain [38]. La dispersion d'un essaim de robots peut fournir un avantage important lors des opérations militaires du fait de la mise en place d'actions coordonnées décentralisées en utilisant uniquement une communication WiFi, ce qui ne cause pas de problème si l'un des robots est détruit [39-43].

2.5 CONCLUSION

Nous avons présenté dans ce chapitre l'état de l'art de la plupart des algorithmes de création d'enveloppe convexe, des différentes méthodes de planification de chemin pour un robot dans différentes situations, et également l'état des recherches sur les essaims de robots. Nous avons pu constater la très grande diversité d'algorithmes qui existe et dont nous pouvons nous servir dans le cadre de notre projet. Chacun des algorithmes présentés a des inconvénients et des qualités que nous devons prendre en compte suivant le contexte de développement de notre système. Cependant, en ce qui concerne la planification de trajectoire, aucune approche n'élimine entièrement les problèmes rencontrés. C'est pourquoi ce domaine est un domaine de recherche encore très actif à l'heure actuel et de nouvelles méthodes de résolution sont publiés assez fréquemment.

Pour pallier à ce problème, nous présenterons dans le chapitre 3 une combinaison d'algorithmes qui planifiera le chemin des robots de notre essaim de manière à éviter le plus possible les problèmes que nous avons vus précédemment.

CHAPITRE 3

MÉTHODOLOGIE ET CONCEPTION DU SYSTEME

3.1 INTRODUCTION

Le chapitre 2 nous a permis de voir qu'il existe de nombreuses méthodes afin de contrôler un essaim de robots de manière autonome. Chacune de ces différentes méthodes permet de résoudre un problème en particulier, mais elles possèdent généralement plusieurs inconvénients. Nous devons donc être en mesure de pouvoir faire nos choix de conception de notre système en fonction de nos besoin et de minimiser le plus possible les conséquences des inconvénients.

Dans le cadre de ce projet, plusieurs objectifs doivent être atteints. Le système doit être capable de repérer en temps réel plusieurs personnes et plusieurs robots (position, orientation dans l'espace). La communication des données doit donc être rapide. Il doit aussi être en mesure de prendre en compte les obstacles présents dans l'environnement proche du groupe de personnes. Le nombre de semelles intelligentes et de robots étant variable, le système doit s'adapter à cette contrainte. Afin de pouvoir mettre en place ce système permettant de contrôler un essaim de robots, il faudra procéder en plusieurs étapes et simplifier les conditions de tests du système. Nous l'avons réalisé en intérieur afin de suivre plus facilement la position des personnes et des robots. Le repère utilisé est en 2D. Nous ne prenons pas en compte l'altitude.

Les différentes parties qui suivent décrivent les étapes de la conception du système. La première partie de ce chapitre présente le matériel utilisé afin de répondre aux exigences du cahier des charges. La seconde partie décrit en détail le fonctionnement des différents algorithmes mis en place pour contrôler de façon autonome les robots. La dernière partie décrit le système au complet, en expliquant les différentes interactions existantes entre chaque élément du système.

3.2 MATERIEL UTILISE

Dans le but de faire fonctionner notre système, nous utilisons plusieurs outils qui sont présentés dans les parties ci-dessous.

3.2.1 ROS (ROBOT OPERATING SYSTEM)

Dans notre projet, nous utiliserons ROS. C'est un logiciel libre d'utilisation permettant d'écrire des programmes informatiques en vue de faire de la robotique. Il possède de nombreux outils et librairies qui simplifient la programmation de système robotique complexe dans de nombreux langages (C, C++, Python…). Dans notre projet nous utilisons la version Indigo de ROS dans l'environnement d'Ubuntu 14.04. Un serveur qui y sera installé agira comme une intelligence artificielle, prenant les décisions pour l'essaim de robots, collectant leurs informations à propos de leurs états et leur environnement, et les commandera à distance de manière autonome.

3.2.2 LE ROBOT

Le robot utilisé est un modèle Lynxmotion Aluminum A4WD1 Robot. Il est composé de deux servomoteurs, l'un commandant sa vitesse d'avancée et l'autre commandant sa vitesse de rotation. Le robot est programmable par une carte Botboarduino qui est un microcontrôleur compatible avec une carte Arduino Duemilanove. Sur chaque robot sera disposé un module WiFi pour pouvoir recevoir et envoyer des données, ainsi qu'une centrale inertielle afin de suivre l'état du robot (vitesse, orientation, détection d'événements). Ses dimensions sont de 32 cm de longueur, 35 cm de largeur et 15 cm de hauteur. Le schéma électrique des composants sur le robot est présenté en Annexe D.

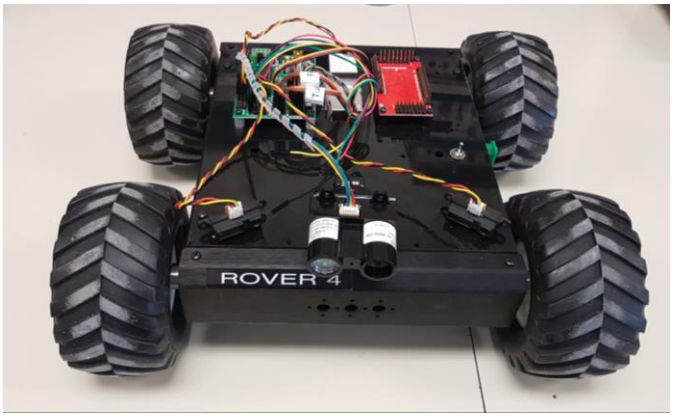


Figure 14. Robot utilisé

3.2.3 CENTRALE INERTIELLE

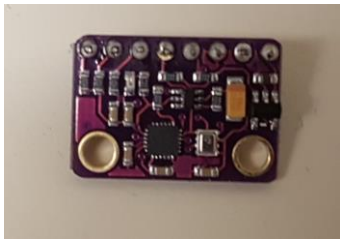


Figure 15. MPU9250

Dans notre projet, nous utilisons un MPU9250. Cette centrale inertielle possède un accéléromètre, un gyroscope ainsi qu'un magnétomètre. Chaque élément possède trois axes (X, Y et Z). L'accéléromètre et le gyroscope serviront à détecter les différents états dans lequel se trouve le robot (en train d'avancer, de chuter dans un trou…). Le magnétomètre lui servira à donner l'orientation de la personne ou du robot.

3.2.4 LA SEMELLE

Dans le but de suivre le mouvement d'un groupe de personne, nous utilisons des semelles intelligentes qui nous fournissent des données. Chaque personne serait équipée d'une semelle. Celle-ci possède des capteurs de pression permettant de savoir quand une personne fait un pas ainsi qu'une centrale inertielle qui nous donne des indications sur la vitesse et l'orientation de la personne. Dans notre projet, nous ne nous occupons pas de la conception de la semelle. Nous l'utiliserons uniquement pour recevoir et traiter les données qu'elle nous envoi.

Figure 16. Semelle intelligente

3.2.5 WBAN

Afin d'établir un protocole de communication entre nos semelles, ROS et les robots, nous avons choisi d'utiliser la technologie WiFi via l'ESP-8266 qui est un petit microcontrôleur pour la communication WiFi. Chaque robot est connecté à un ESP dans le but de créer un réseau qui communique beaucoup d'information dans un lapse de temps très court. Cette technologie est assez petite et peut facilement être connectée aux robots. C'est pourquoi nous l'avons choisi dans le cadre de notre projet.

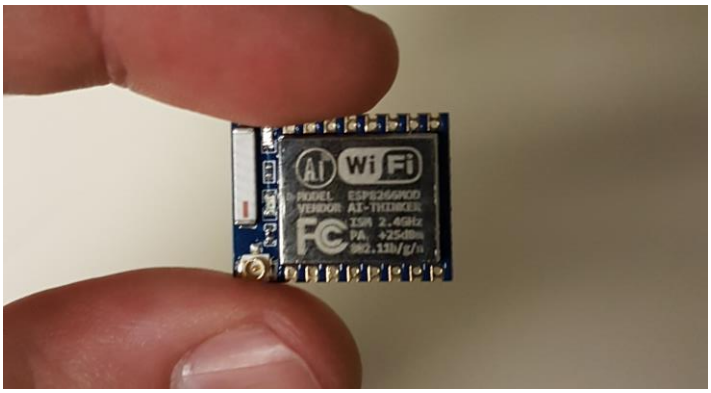


Figure 17. ESP8266

3.2.6 LE SYSTEME DE CAMERA OPTITRACK

Afin de trouver la position des personnes et des robots, nous utilisons un système de caméra Optitrack de Natural Point. Celui-ci se compose de 8 caméras infrarouges pouvant atteindre 100fps. Le système détecte des marqueurs réfléchissants passifs de 11mm posés sur la jambe de

la personne et sur le dessus du robot. Il est possède de détecter plusieurs marqueurs simultanément et d'avoir leur position au millimètre près, ainsi que leur orientation dans le repère que l'on a défini pour l'expérimentation.

Figure 18. Caméra Optitrack

Les caractéristiques des caméras sont les suivants :

- 46° de champ de vision
- connexion à l'ordinateur via un câble USB
- une résolution de 0.3 MP
- une fréquence de 100 FPS
- une précision submillimétrique
- un temps de latence de 10 ms
- 26 LED infrarouge

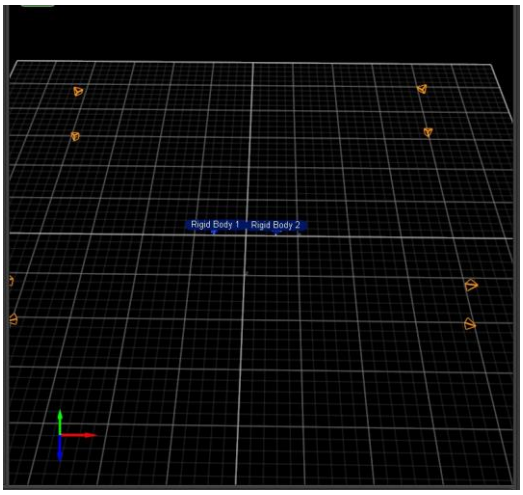
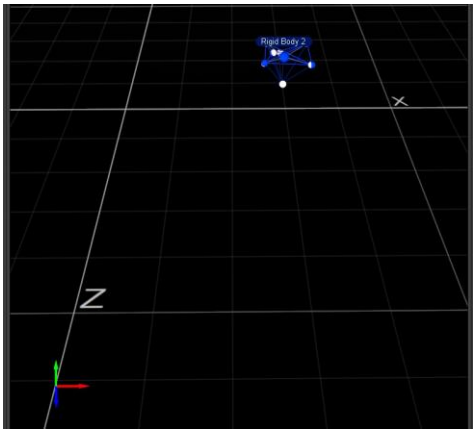


Figure 19. Repère de travail des caméras **Figure 20. Marqueur utilisé**

Avant de faire la détection des marqueurs, il a fallu calibrer le système de caméras (triangulation 3D). Cet espace ou volume de détection permet par la suite de couvrir un maximum de surface suivant la disposition des caméras. Le logiciel permettant d'acquérir les données des caméras ne fonctionne que sous Windows. Cependant nous pouvons envoyer les données via des sockets à notre serveur sous ROS. Ces données seront par la suite réutilisées par l'algorithme de positionnement des robots, présenté dans la partie suivante.



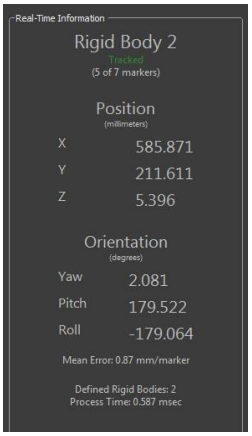


Figure 21. Tracking du robot par les caméras **Figure 22. Données sur le robot**

3.3 LES ALGORITHMES

Une fois que les personnes sont équipées de semelles intelligentes, nous devrions être en mesure de traiter les données que l'on reçoit. Afin de protéger ce groupe de personnes à l'aide de

nos robots, nous devons procéder en 3 étapes : premièrement nous devons fixer une position à atteindre pour chaque robot autour du groupe de personne. Ensuite nous devons planifier la trajectoire du robot jusqu'à cet objectif. Enfin le robot doit pouvoir effectuer ce trajet planifié. Les parties ci-dessous présentent les 3 différents algorithmes qui réalisent ces différentes étapes.

3.3.1 POSITIONEMENT DES ROBOTS

Cet algorithme réalise le positionnement des robots autour du groupe de personnes. Dans l'état de l'art du chapitre 2, nous avons vu qu'il existe de nombreux algorithmes pour trouver l'enveloppe convexe d'un ensemble de point. Nous avons décidé d'utiliser l'enveloppe convexe dans notre projet afin d'encercler complètement le groupe de personne par les robots. Cela aurait pu ne pas être le cas avec une enveloppe concave. Ce type d'enveloppe nous permet également de mieux répartir les robots autour des personnes.

L'algorithme va chercher dans la base de données les informations sur les personnes du groupe (vitesse, orientation, position). Ces données serviront à définir le vecteur de déplacement du robot lorsqu'il atteindra sa nouvelle position et à s'y adapter en temps réel. Les différentes étapes sont présentées dans la Figure 24. L'algorithme fonctionne sous ROS dans un nœud différent des semelles et des robots. Cela permet à celui-ci de ne pas dépendre et de s'adapter au nombre de personne ou de robot connecté au système.

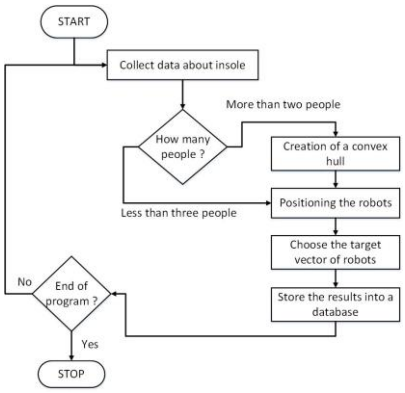


Figure 23. Organigramme de l'algorithme de positionnement des robots

Une fois les données récupérées, on regarde le nombre de personne qui sont présentes. Pour moins de trois personnes, l'algorithme va placer directement les robots sur un cercle autour de celle(s)-ci. Le centre du cercle est la moyenne de la position des deux personnes ou la position de l'unique personne s'il y en a qu'une. Une distance entre la personne et les robots est définie afin de laisser assez d'espace entre le robot et la/les personne(s). Les robots sont placés sur le cercle

de manière à former des sections angulaires uniformes, et d'avoir toujours un robot en face de la position du déplacement moyen du groupe. Par exemple pour trois robots et une personne, l'angle formé entre deux robots sur le cercle entourant la personne sera de 120°, et un robot sera positionné devant la personne lorsqu'elle marchera. Le vecteur de déplacement de chaque robot sera celui de la personne la plus proche de lui.

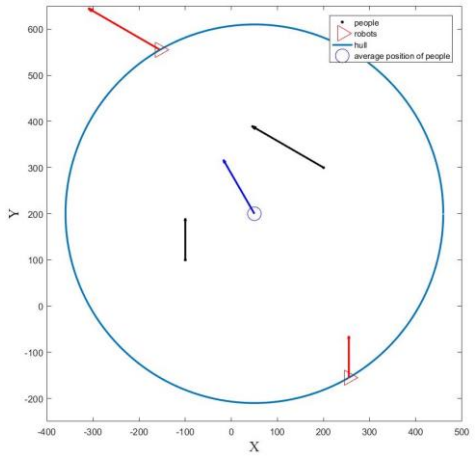


Figure 24. Exemple de résultat avec deux personnes et deux robots

Si elles sont plus de deux, on utilise l'algorithme de la Marche de Jarvis pour trouver les personnes dessinant l'enveloppe convexe du groupe. Une fois que l'enveloppe convexe du groupe est formée, on cherche à dilater celle-ci grâce à une distance fixe entre les personnes et les robots.

On obtient alors une deuxième enveloppe convexe sur laquelle les robots seront positionnés. Les robots sont toujours placés de manière à former des sections angulaires uniformes, et de façon à avoir toujours un robot en face de la position du déplacement moyen du groupe. Une fois la position fixé pour chaque robot, on leur attribue leur vecteur de déplacement grâce au maximum de la valeur du produit scalaire entre le vecteur des personnes présentes proches de la position à atteindre et le vecteur normale à l'enveloppe convexe secondaire de la position à atteindre. Le vecteur pour lequel cette valeur est maximale est le vecteur de déplacement choisi. Si jamais la personne est à plus de 3 mètres de la position à atteindre, la valeur du vecteur de déplacement donnée au robot est divisée par deux. Cela permet au robot d'anticiper le fait qu'une personne va sortir de l'enveloppe convexe, modifiant ainsi l'enveloppe convexe, et donc modifier la disposition des robots. Dans le cas où personne n'est situé à moins

de 6 mètres de la position cible, la valeur du vecteur de déplacement est celle du vecteur moyen de déplacement du groupe. Un exemple est donné par la Figure 26.

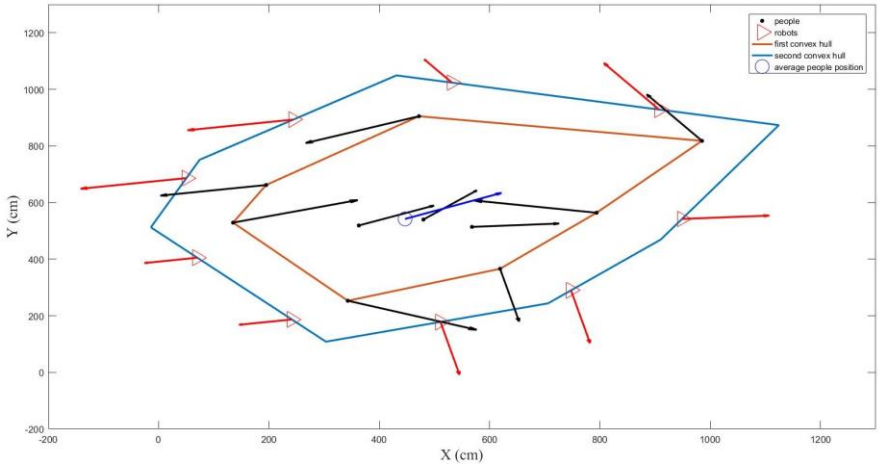


Figure 25. Exemple avec 9 personnes et 9 robots

Pour faire le lien entre la position cible et le robot, nous associons chaque résultat au robot le plus proche de la position à atteindre. Ces résultats sont ensuite envoyés dans la base de données pour être utilisés par l'algorithme de planification de trajectoire.

3.3.2 PLANIFICATION DE TRAJECTOIRE

Comme nous l'avons vu dans le chapitre 2, il existe de nombreuses méthodes de planification de chemin en robotique pour une utilisation en temps réel. Dans le cadre de notre projet, nous avons choisi d'utiliser la méthode de planification locale du diagramme de proximité (VFH : Vector Field Histogram), méthode qui nous permet de traiter les informations en temps réel en évitant certains problèmes d'oscillation de trajectoire de certaines méthodes comme celle des champs de potentiel (VFF : Virtual Force Field). Afin de réduire encore les problèmes inhérents au diagramme de proximité, nous couplerons cette méthode avec un bloc de logique floue qui commandera les moteurs du robot dans le but d'éviter la collision avec des obstacles [44]. Nous développerons cette méthode dans la partie suivante.

L'algorithme de planification de trajectoire fonctionnera sur le nœud de chaque robot sous ROS, permettant le traitement en parallèle des trajectoires des différents robots connectés composant l'essaim. Un organigramme de ce nœud est donné en Annexe A. Initialement,

l'algorithme va construire une carte de proximité autour du robot en allant chercher les coordonnées des obstacles dans une base de données. Cette carte a une dimension de 2.1x2.1 mètres et est de forme carrée. Elle est représentée dans le programme par une matrice carrée de 21 colonnes et 21 lignes. Nous avons choisi de compresser par un facteur 10 les données sur la position des obstacles afin de réduire le temps de calcul du processus. Ainsi chaque case de la matrice correspond à une zone de 10x10 cm autour du robot. Si un obstacle est présent dans l'une de ces cases, la valeur de celle-ci passe à 1. Sinon elle reste à 0. Les algorithmes que nous avons créés ne prennent en considération que des points pour représenter les personnes, les robots ou les obstacles. Ils ne prennent pas en compte la forme de ceux-ci ce qui peut causer des problèmes de collision lors de la planification de trajectoire. Pour pallier à ce problème, nous rajoutons un contour de 20 cm (2 cases) autour des différents objets ou personnes sur la carte de proximité.

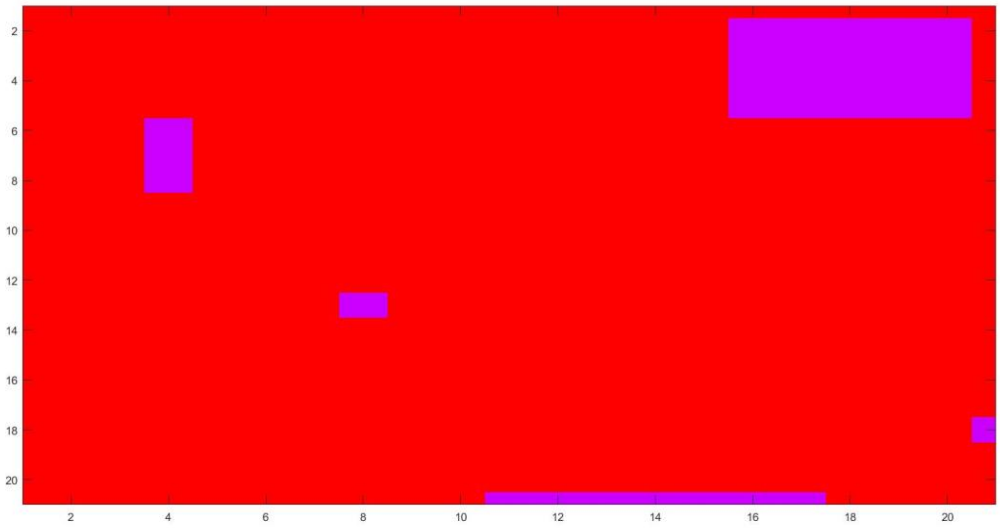


Figure 26. Exemple de carte de proximité avec les obstacles en rose

Afin d'optimiser le trajet du robot, nous avons voulu utiliser une méthode de diagramme de proximité qui prenne en compte la morphologie du robot. Celui-ci possède une longueur d'au maximum 40 cm en prenant une petite distance de sécurité. Le diagramme de proximité doit donc être en mesure de détecter les espaces assez étroits dans lesquels le robot pourrait passer. Pour cela nous diviserons de manière uniforme la carte de proximité en 15 sections rectangulaires de 40 centimètres de largeur. L'angle de rotation entre chaque section serait alors de 24°.

Ce découpage est effectué en début du programme pour éviter de perdre du temps de traitement de données. Les cases de la matrice correspondant à chaque section sont stockées dans des tableaux pour être utilisées par la suite. Un exemple de découpage est montré à la Figure 28.

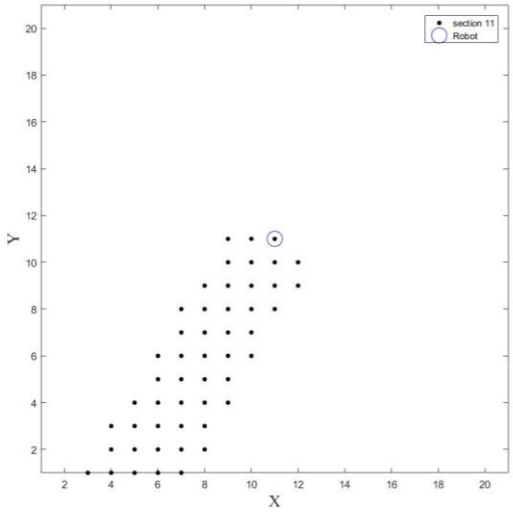


Figure 27. Exemple de section de la carte de proximité calculée

Cette façon de prendre en compte la morphologie du robot permet de mieux adapter la solution à l'environnement du robot. Cependant elle est un peu plus compliquée à mettre en œuvre que la solution traditionnelle qui consiste à prendre pour section des arcs de cercle uniformément répartis. Nous avons donc comparé les deux solutions au niveau du temps de calcul logiciel afin de voir si la différence est importante. Pour ce faire nous avons utilisé une classe de Timer qui permet de calculer ce temps sans 'time overhead' pour un programme en C++ sous Windows. Comme l'algorithme tourne sous ROS, on peut s'attendre à ce que les résultats trouvés soit meilleurs puisqu'il utilise un OS de type Unix. Les résultats sont montrés à la Figure 29, Figure 30 et le Tableau 1.

Tableau 1. Résultats obtenus sur un échantillon de 1000 essais

	Moyenne	Écart-type
Section circulaire	4,41 ms	0,67 ms
Section rectangulaire	13,66 ms	2,53 ms

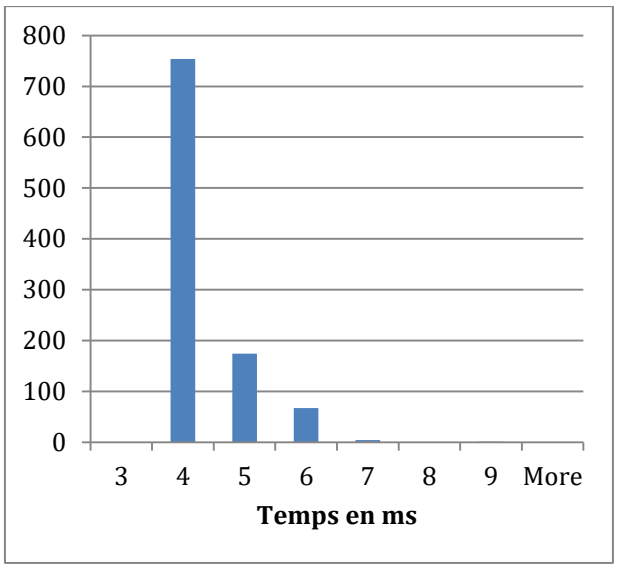


Figure 28. Performance Section Circulaire sur 1000 essais

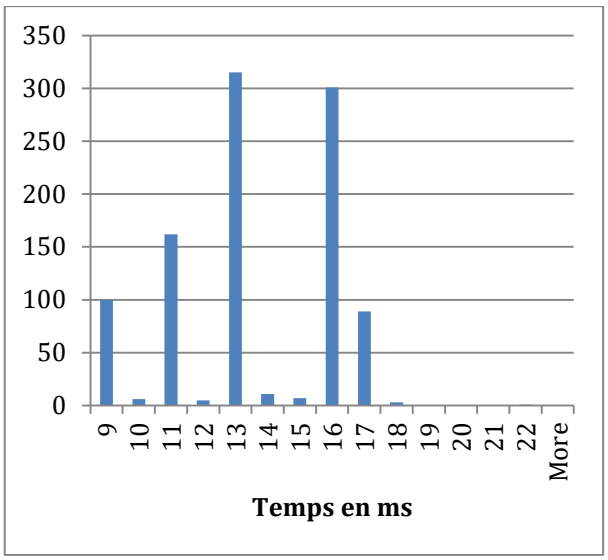


Figure 30. Performance Section Rectangulaire sur 1000 essais

Les résultats montrent que l'utilisation des sections rectangulaire multiplie par environ 3 le temps de calcul. Malgré tout, celui-ci est assez faible pour pouvoir être utilisé sur un système en temps réel.

Une fois que les sections sont créées, l'algorithme vérifie la présence d'un obstacle dans chacune d'entre elles. Celles qui ne possèdent pas d'obstacles sont mémorisées et sont vu comme une potentielle solution pour la planification de trajectoire. Par la suite deux cas ce présente : la position finale donnée par l'algorithme de positionnement se trouve dans la carte de proximité, ou elle se trouve à l'extérieur.

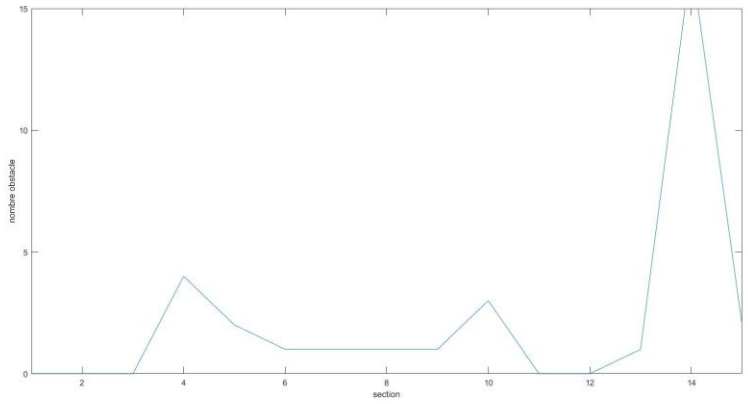


Figure 29. Détection de la présence d'obstacles dans chaque section

Dans le premier cas, si la section contenant cette position ne possède pas d'obstacles ou si l'obstacle est situé derrière celle-ci, alors la solution donnée est directement la position à atteindre. Sinon c'est la position moyenne du bord de la section ne contenant pas d'obstacle et étant la plus proche de la position finale qui est sélectionnée. De même si la position finale n'est pas dans la carte de proximité.

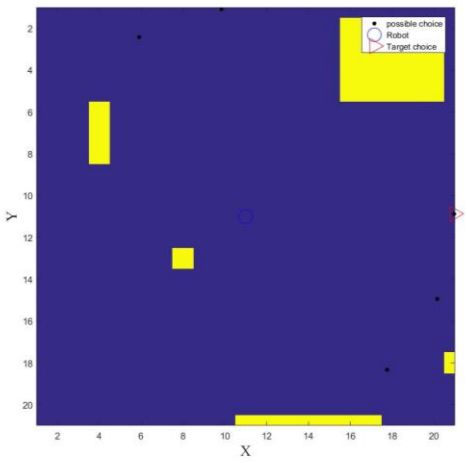


Figure 30. Résultat de l'algorithme pour un objectif situé en (30 ; 8)

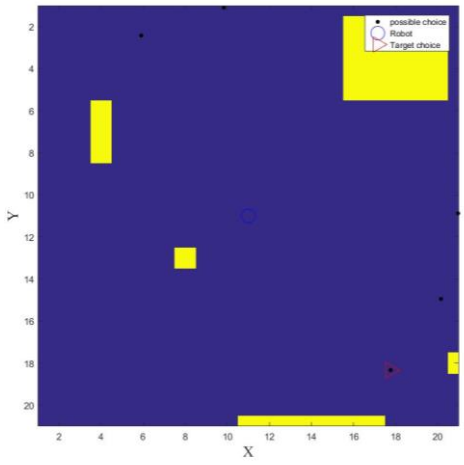


Figure 31. Résultat de l'algorithme pour un objectif situé en (12 ; 30)

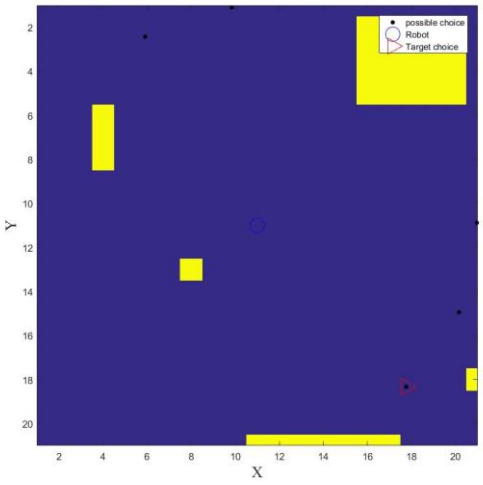


Figure 32. Résultat de l'algorithme pour un objectif situé en (4 ; 18)

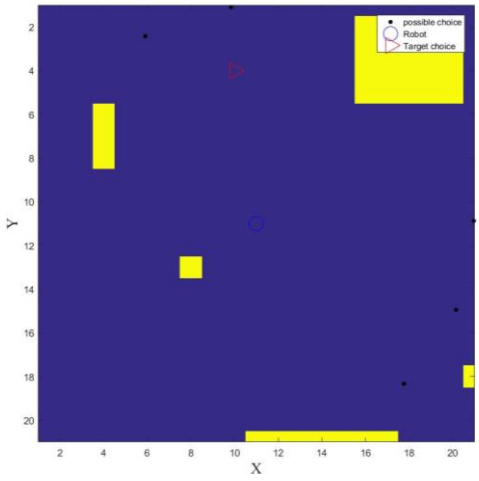


Figure 33. Résultat de l'algorithme pour un objectif situé en (10 ; 4)

La solution est ensuite traitée pour donner des indications aux robots afin de préparer la commande des moteurs. Grâce à celle-ci, nous pouvons calculer la distance restante du robot par rapport à la solution ainsi que la différence d'orientation entre le robot et la position à atteindre. Le sens de la rotation à effectuer est également calculé. De plus durant la création de la carte de proximité, nous avons effectué un calcul de la plus courte distance par rapport à un obstacle. Ce sont ces quatre données qui sont transmissent au robot, pour lequel l'algorithme de logique floue va créer la commande des moteurs.

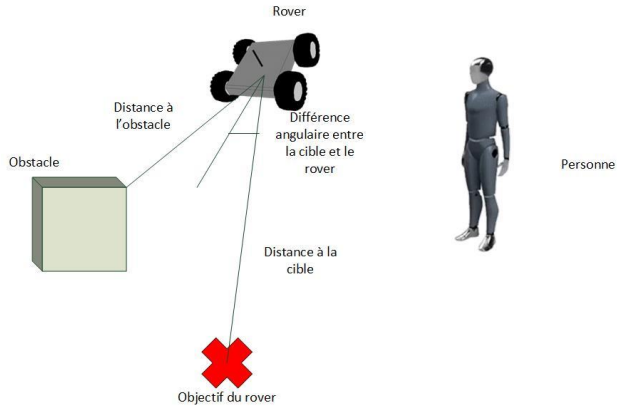


Figure 34. Schéma de la signification des données transmisses au robot

3.3.3 COMMANDES DES MOTEURS

Une fois les données calculées, elles sont envoyées aux robots par le serveur via leur nœud respectif. L'ESP8266 présent sur chaque robot les reçoit et exécute l'algorithme qui créera la commande des moteurs. Le robot ne possède pas un contrôle linéaire de la vitesse de ces moteurs (commande par palier via des valeurs données aux servomoteurs). C'est pour cela que nous avons choisi de commander les moteurs par un bloc de logique floue pour résoudre ce problème [45-46]. Un organigramme de ce code est présenté en Annexe B et C.

Ce bloc possède 3 entrées : la distance restante entre la position du robot et la position cible à atteindre, la différence d'angle entre l'orientation du robot et la position cible, ainsi que la distance minimale par rapport à un éventuel obstacle autour du robot. Deux sorties seront créées par l'algorithme : la commande en vitesse du robot en translation et celle en rotation.

La distance entre la cible et le robot est définie en centimètre. Dans notre cas elle ne dépassera pas 142 cm du fait de la construction de la carte de proximité. L'angle de différence sera lui compris entre 0 et 180°. La distance maximale par rapport à un obstacle est limitée à 142cm du fait également de la carte de proximité. Ce cas se produit s'il n'y a aucun obstacle autour du robot. En ce qui concerne les commandes des moteurs, les valeurs à transmettre à chacun des servomoteurs vont de 60 à 130. La valeur d'arrêt est de 95. Pour la commande en vitesse du robot, celui-ci avancera en ligne droite pour une valeur comprise entre 95 et 130, 130 étant la valeur de la vitesse maximale du robot. La plage de valeur de 60 à 95 correspond à une phase de recul du robot, 60 étant la valeur maximale de sa vitesse. Pour ce qui est de la vitesse de rotation du robot, la plage de valeur de 95 à 130 correspond à une rotation dans le sens horaire, 130 étant la vitesse maximale. La plage de valeur de 60 à 95 correspond à une rotation dans le sens trigonométrique, 60 étant sa valeur maximale. Dans le cadre de notre projet, nous ne prendrons pas en compte la phase de recul du robot dans le but de simplifier le problème. La sortie correspondant à la commande de la vitesse de rotation aura une plage de valeur de 0 à 37 et plus. Ce résultat sera ajouté ou soustrait en fonction de la valeur du sens de déplacement calculé par le serveur et transmise au robot. Du fait de la plage de valeur de toutes ces variables, nous avons alors défini les fonctions d'appartenance de chaque entrées et sorties du bloc de logique floue, présenté en Figure 36, 37, 38 ,39 et 40.

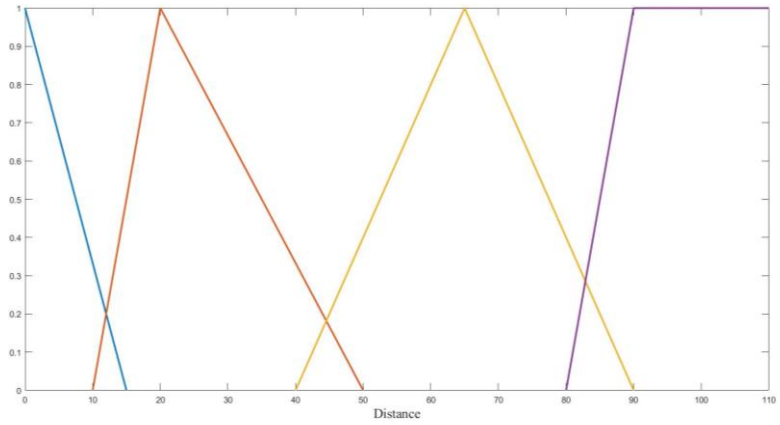


Figure 35. Fonction d'appartenance de la distance entre la cible et le robot (zeroDistance, procheDistance, moyenDistance et loinDistance)

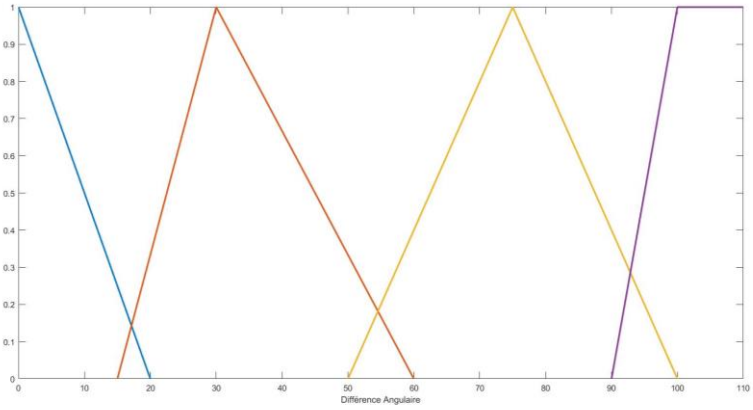


Figure 36. Fonction d'appartenance de la différence angulaire entre le robot et la cible (zeroAngle, procheAngle, moyenAngle et loinAngle)

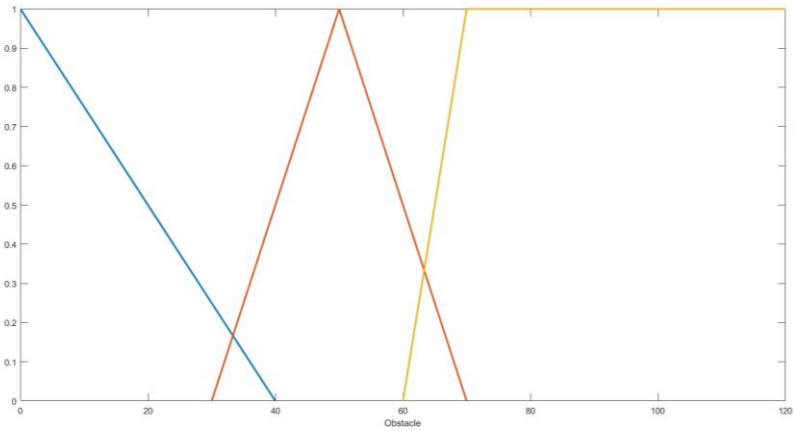


Figure 37. Fonction d'appartenance de la distance à un obstacle

(procheObstacle, moyenObstacle et loinObstacle)

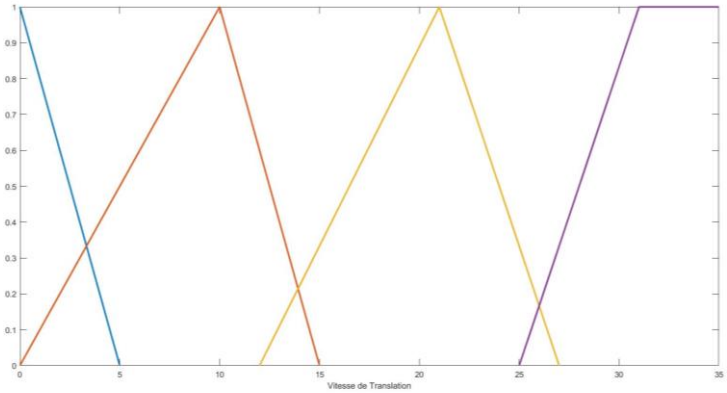


Figure 38. Fonction d'appartenance de la vitesse de translation du robot (pasVelocity, peuVelocity, avanceVelocity et viteVelocity)

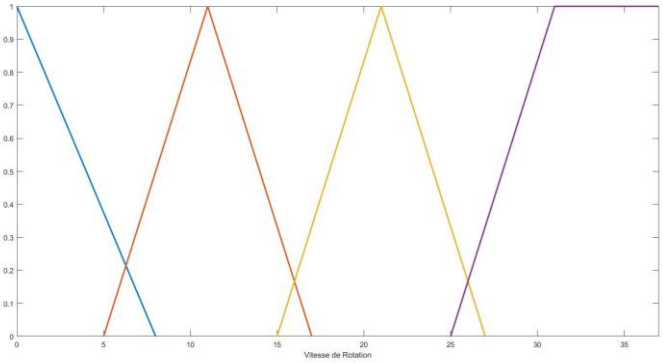


Figure 39. Fonction d'appartenance de la vitesse de rotation du robot (pasOrientation, peuOritentation, avanceOrientation et viteOrientation)

Du fait de cette définition des différents états de nos variables, il existe 1200 règles possibles pour ce système. Afin de réduire le temps d'exécution du programme, nous en avons sélectionné 28 qui nous permettent de contrôler le robot dans toutes les situations réalistes pouvant se produire. Ces règles donnent les valeurs des variables de sorties en se basant sur les fonctions d'appartenance précédemment défini pour reconnaitre la situation dans laquelle se trouve le robot. Un exemple de règle floue est présenté ci-dessous :

RULE 13 : IF ((distance EST faible) ET (diff_angle EST faible)) ET ((obstacle EST loin) OU (obstacle EST proche)) THEN ((vitesse EST faible) ET (rotation EST faible))

Ces différentes règles sont présentées dans le Tableau 2.

Tableau 2. Liste des règles de la bibliothèque de logique floue

numéro	Distance	Différence Angulaire	Obstacle	Vitesse Translation	Vitesse Rotation
1	zeroDistance	zeroAngle	procheObstacle OU moyenObstacle OU loinObstacle	pasVelocity	pasOrientation
2	procheDistance	zeroAngle	procheObstacle OU moyenObstacle OU loinObstacle	peuVelocity	pasOrientation
3	moyenDistance	zeroAngle	procheObstacle	peuVelocity	pasOrientation
4	moyenDistance	zeroAngle	moyenObstacle OU loinObstacle	avanceVelocity	pasOrientation
5	loinDistance	zeroAngle	procheObstacle	peuVelocity	pasOrientation
6	loinDistance	zeroAngle	moyenObstacle	avanceVelocity	pasOrientation
7	loinDistance	zeroAngle	loinObstacle	viteVelocity	pasOrientation
8	zeroDistance	procheAngle	procheObstacle OU moyenObstacle OU loinObstacle	pasVelocity	peuOrientation
9	zeroDistance	moyenAngle	procheObstacle OU moyenObstacle OU loinObstacle	pasVelocity	avanceOrientation
10	zeroDistance	loinAngle	procheObstacle OU moyenObstacle OU loinObstacle	pasVelocity	viteOrientation
11	procheDistance	procheAngle	moyenObstacle OU loinObstacle	peuVelocity	peuOrientation

12	procheDistance	moyenAngle	procheObstacle	peuVelocity	peuOrientation
13	procheDistance	moyenAngle	moyenObstacle OU loinObstacle	peuVelocity	avanceOrientation
14	procheDistance	loinAngle	procheObstacle OU moyenObstacle OU loinObstacle	pasVelocity	viteOrientation
15	moyenDistance	procheAngle	procheObstacle	peuVelocity	peuOrientation
16	moyenDistance	procheAngle	moyenObstacle OU loinObstacle	avanceVelocity	peuOrientation
17	moyenDistance	moyenAngle	procheObstacle	peuVelocity	avanceOrientation
18	moyenDistance	moyenAngle	moyenObstacle OU loinObstacle	avanceVelocity	avanceOrientation
19	moyenDistance	loinAngle	procheObstacle OU moyenObstacle	pasVelocity	viteOrientation
20	moyenDistance	loinAngle	loinObstacle	peuVelocity	viteOrientation
21	loinDistance	procheAngle	loinObstacle	viteVelocity	peuOrientation
22	loinDistance	procheAngle	moyenObstacle	avanceVelocity	peuOrientation
23	loinDistance	procheAngle	procheObstacle	peuVelocity	peuOrientation
24	loinDistance	moyenAngle	loinObstacle	viteVelocity	avanceOrientation
25	loinDistance	moyenAngle	moyenObstacle	avanceVelocity	avanceOrientation
26	loinDistance	moyenAngle	procheObstacle	peuVelocity	avanceOrientation
27	loinDistance	loinAngle	procheObstacle	peuVelocity	viteOrientation
28	loinDistance	loinAngle	moyenObstacle OU loinObstacle	avanceVelocity	viteOrientation

Une fois les variables de sorties défuzzifier, elles sont envoyées vers la carte BotBoarduino via une communication I2C avec la valeur du sens de rotation. Le programme sur la carte effectue les ordres et le robot se déplace.

3.4 BASE DE DONNEES

Afin de transmettre les données entre les différents nœuds de ROS et de garder une trace de ce qu'il s'est passé dans le système, nous avons créé une base de données sous PostgreSQL et

relié celle-ci à ROS. Le choix de choisir PostgreSQL comme base de données a été motivé par la rapidité d'exécution des requêtes ainsi que par sa simplicité d'installation. Les données enregistrées concernent les différentes semelles et robot présents sur le terrain, les résultats de l'algorithme de positionnement et de planification de trajectoire, ainsi que les obstacles présents dans l'environnement des robots. Une représentation des différentes tables des données est présentée ci-dessous.

Tableau 3. Table des obstacles

Obstacle	
Id	integer
X	Double precision
Y	Double precision

L'Id permet de différencier les différents obstacles. Le X et le Y sont les coordonnées de celui-ci.

Tableau 4. Table du résultat de l'algorithme des sections

Ordre	
Id	integer
RX	Double precision
RY	Double precision
CX	Double precision
CY	Double precision
VC	Double precision
OC	Double precision
VR	Double precision
Obstacle	integer

L'Id est le numéro de l'adresse IP du robot. Ainsi on peut facilement retrouver l'ordre associé à celui-ci. Le RX et le RY sont les coordonnées du robot. Le CX et le CY sont les coordonnées de sa cible à atteindre. VC et VR sont respectivement la vitesse à avoir sur la cible et la vitesse du robot. OC est l'orientation à avoir sur la cible et Obstacle est la distance entre le robot et l'obstacle le plus proche.

Tableau 5. Table d'une semelle intelligente

Personne	
Id	integer
X	Double precision
Y	Double precision
Orientation	Double precision
Vitesse	Double precision
Temps	Double precision

L'Id est l'adresse IP de la semelle intelligente. Le X et le Y sont les coordonnées de celle-ci. L'Orientation est l'orientation de la semelle sur le sol. La Vitesse est la vitesse de la semelle et le Temps est le temps à laquelle les données ont été mises en mémoire.

Tableau 6. Table d'un robot

Robot	
Id	integer
X	Double precision
Y	Double precision
Orientation	Double precision
Vitesse	Double precision
Temps	Double precision
État	integer

L'Id est l'adresse IP du robot. Le X et le Y sont les coordonnées de celle-ci. L'Orientation est l'orientation du robot sur le sol. La Vitesse est la vitesse du robot. Le Temps est le temps à laquelle les données ont été mises en mémoire et l'Etat est l'état dans lequel le robot se trouve (0 : pas encore connecté au système ; 1 : opérationnel ; 2 : déconnecté du système ; etc…).

Tableau 7. Table du résultat de l'algorithme de positionnement

Algorithme positionnement	
Id	integer
X	Double precision
Y	Double precision
Orientation	Double precision
Vitesse	Double precision

L'Id est l'adresse IP du robot qui va recevoir la position de la cible. Le X et le Y sont les coordonnées de la cible à atteindre. L'Orientation et la Vitesse sont le vecteur de déplacement que doit avoir le robot lorsqu'il atteint la cible.

3.5 DETECTION D'EVENEMENTS LIES AUX ROBOTS

Notre système évoluant dans un environnement assez incertain et de façon autonome, nous devons être en mesure d'avoir assez d'éléments permettant de définir l'état de chacun des robots. Ce contrôle permet de chercher un quelconque problème dans le fonctionnement du robot et de chercher à trouver une solution pour l'opérateur ainsi que pour l'essaim. C'est le travail que nous vous présentons dans cette partie.

3.5.1 BIBLIOTHEQUE D'EVENEMENTS

Dans le cadre de ce projet, nous avons proposons de détecter une série d'évènements qui pourraient avoir lieu lors de la mise en marche de l'essaim. Voici une liste des évènements qu'il serait intéressant de contrôler :

- Une panne de batterie
- Un robot coincé dans un tas de branches
- Une perte de signal/déconnexion du robot
- Le basculement d'un robot dans un trou
- L'explosion d'un robot
- Un ennui mécanique empêchant le robot de se déplacer
- Un robot qui se retournera

- Une collision éventuelle entre le robot et un obstacle

Certains de ces évènements ne pouvant pas être testé dans notre laboratoire, nous avons décidé de travailler sur la détection de quatre évènements : la perte de signal d'un robot (état 2), le robot qui tombe dans un trou (état 3), l'enlisement d'un robot dans du sable (état 4), et le robot qui entre en collision avec un obstacle non détecté (état 5).

3.5.2 DETECTION DES EVENEMENTS AFFECTANT L'ESSAIM DE ROBOTS

3.5.2.1 DETECTION D'UNE DECONNEXION

Afin de détecter une déconnexion venant du serveur ou du robot, nous devons effectuer une vérification de la bonne transmission des données via les sockets.

Dans le cas de la déconnexion venant du robot, le nœud de celui-ci le remarquera au bout de 5 secondes (duré fixée par nous-même à l'aide d'un Timer) puisqu'il attend les données venant du la centrale inertielle. S'il ne reçoit pas les données avant ce délai, le serveur fait passer l'état du robot de 1 à 2 indiquant ainsi à l'algorithme de positionnement des robots de ne plus prendre en compte celui déconnecté. Le nœud du robot ferme le socket puis en ouvre une nouvelle en attendant une éventuelle reconnexion. Le robot pourra alors se reconnecter et refaire partie de l'essaim.

Dans le cas où c'est le serveur tout entier qui a un problème et déconnecte tous les robots, ceux-ci vont détecter le fait de ne plus recevoir de données du serveur et vont mettre à l'arrêt leurs moteurs. Ils vont essayer de se reconnecter à leur nœud pour recevoir de nouveau leurs ordres.

3.5.2.2 DETECTION D'UNE CHUTE

Afin de détecter une chute du robot avec la centrale inertielle, nous avons réalisé une cinquantaine de prise de mesure d'une chute de celui-ci afin de trouver des caractéristiques de celle-ci. Pour avoir des éléments de comparaison, nous avons également réalisé des mesures de données sur un trajet normal sans chute. Un exemple de données reçues sur chacun des axes lors d'une chute est montré à la Figure 41.

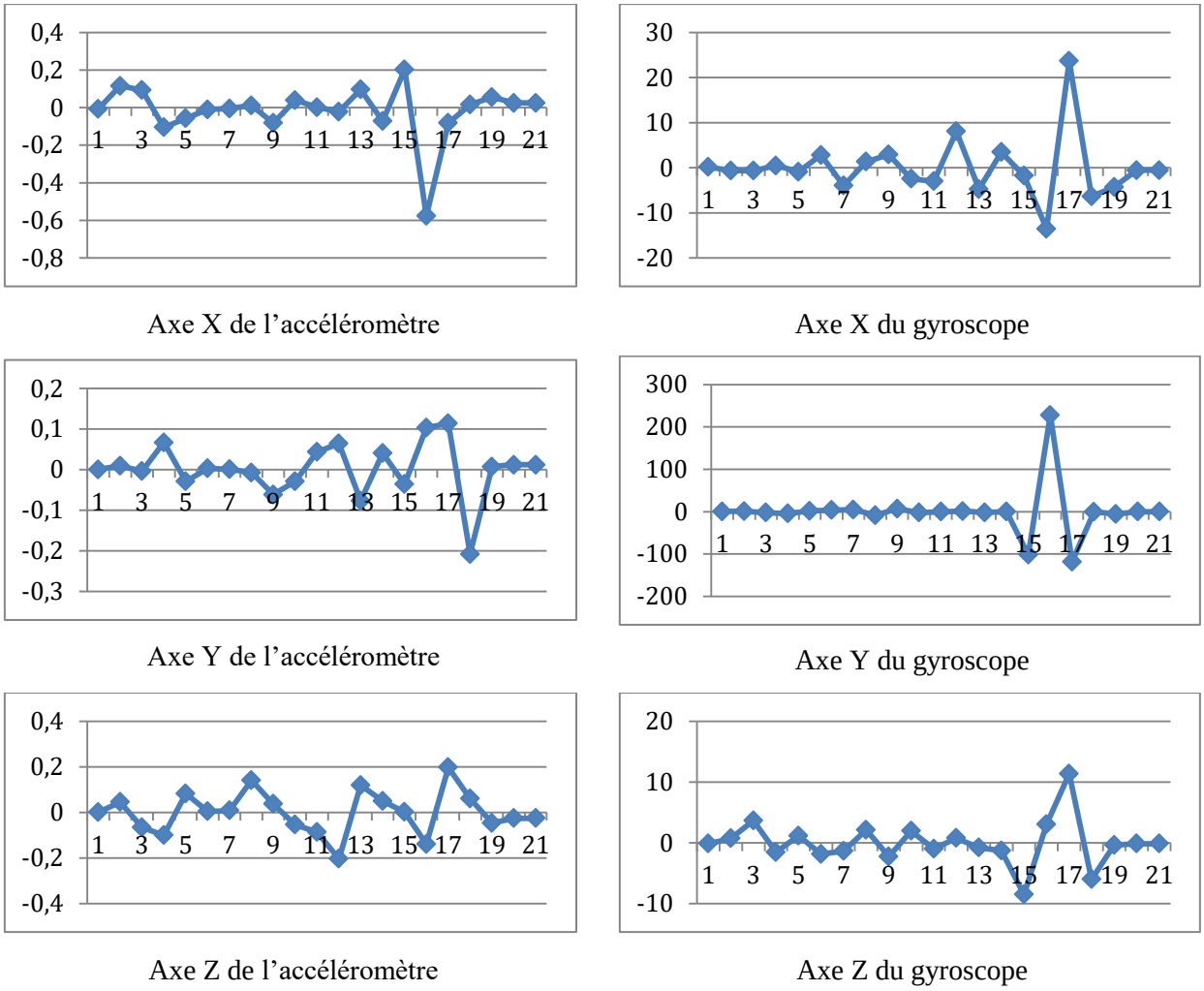


Figure 40. Exemple de données reçu lors d'une chute d'un robot (temps n°16 des abscisses)

Comme nous pouvons le constater, tous les axes sont plus ou moins impactés par cette chute. Afin de traiter les données en temps réel reçu pour la détecter, nous avons choisi de prendre une fenêtre dynamique de huit points sur chacun des axes. Une mesure de l'écart-type a été réalisée sur un échantillon de 81 fenêtres de chute et 429 fenêtres d'une avance normale. Les résultats sont présentés dans le Tableau 8.

Tableau 8. Caractéristiques de l'écart-type obtenu sur les échantillons

	Avance normale			chute		
	min	max	moyenne	min	max	moyenne
Ax	0.0224	0.3541	0.0998	0.1841	2.1664	1.0660
Ay	0.0305	0.6044	0.1139	0.1039	1.0295	0.4515
Az	0.0591	1.2871	0.2442	0.1039	2.8615	0.9917
Gx	0.9170	31.5917	6.3440	5.0755	315.3793	28.3540
Gy	1.8020	36.5200	10.8687	35.7923	666.7851	130.2606
Gz	0.4241	19.2819	3.1982	5.9463	310.6607	45.5385

Nous pouvons constater que certains axes sont plus importants pour la détection que d'autres. C'est le cas de l'axe des X de l'accéléromètre et de l'axe des Y du gyroscope.

Plusieurs méthodes ont été envisagées pour détecter la chute. La première fut un réseau de neurones qui prenait en entrée les 8 écart-type de chacun des axes. Il parvenait à détecter sans problème les cas de chute. Néanmoins il n'arrivait pas à faire la différence entre une avance normale et une chute, les données d'entrées étant normalisées entre -1 et 1. Cela modifiait la signification de celle-ci. La méthode des seuillages aurait pu être utilisée. Cependant au vu des résultats du Tableau précédent, trop de faux positifs ou de vrais négatifs auraient pu être détectés.

Nous avons alors pris la décision d'attribuer un coefficient d'importance à chacun des axes, chaque coefficient étant compris entre 0 et 1 et la sommes de ceux-ci vaut 1. Ils multiplieront l'écart-type de leur fenêtre de leur axe et nous obtiendrons alors une valeur à laquelle nous réaliserons un seuillage. La valeur des coefficients choisis est présentée dans le Tableau ci-dessous.

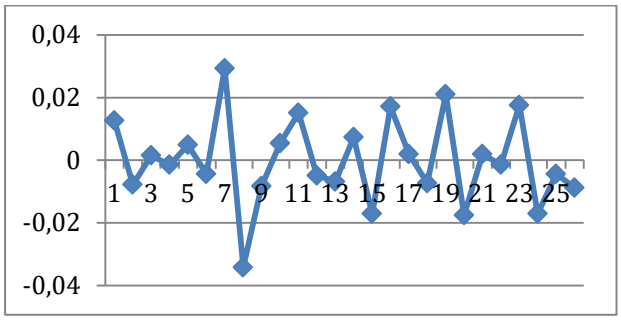
Tableau 9. Coefficient choisis pour détecter une chute

Ax	Ay	Az	Gx	Gy	Gz
0.3	0.1	0	0.1	0.45	0.05

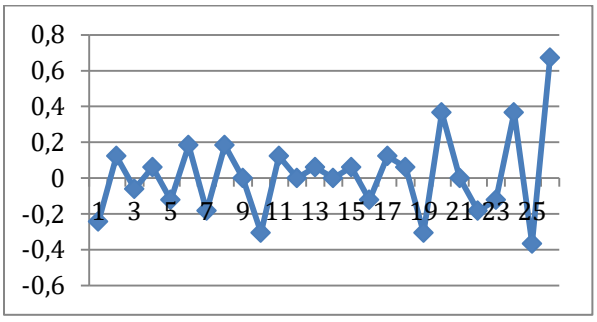
En comparant les valeurs obtenues pour chacune des fenêtres de chute et les fenêtres d'avance normale, nous trouvons que la valeur de seuillage est de 18. Cette valeur nous permet de différencier les deux états. En plus du seuillage, nous réaliserons également un contrôle sur la position du robot. Si celui-ci ne bouge plus alors que les commandes des moteurs lui demandent le contraire, nous aurons alors bien la confirmation qu'il est coincé.

3.5.2.3 DETECTION D'UN ENLISAGE DANS DU SABLE

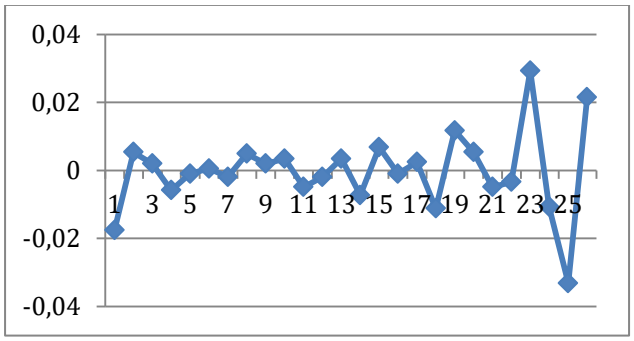
Dans le but de détecter le patinage des roues du robot dans du sable, nous avons également réalisé une dizaine de prises de données correspondant à ce cas. Un exemple de données reçues sur chacun des axes lors d'un enlisage est montré à la Figure 42.



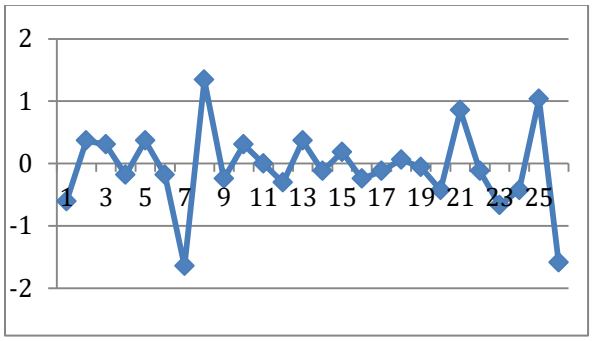
Axe X de l'accéléromètre



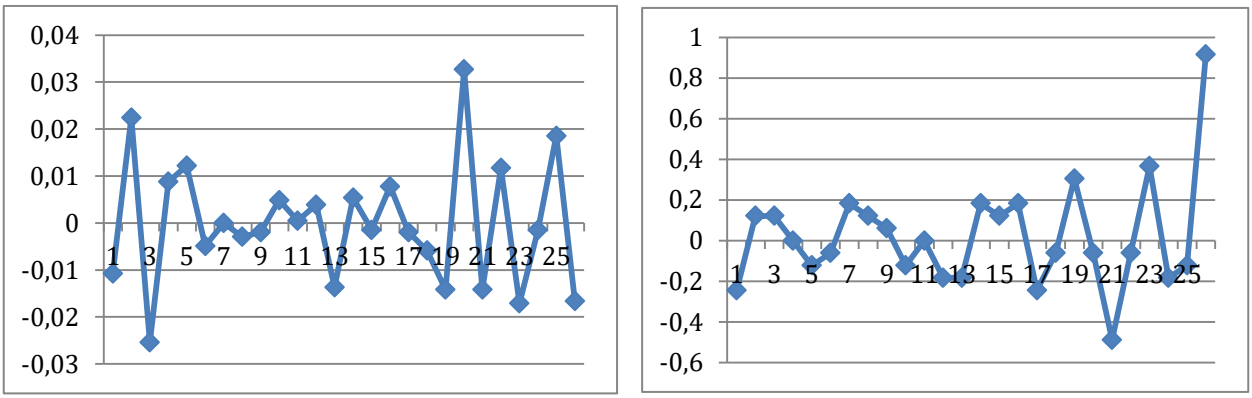
Axe X du gyroscope



Axe Y de l'accéléromètre



Axe Y du gyroscope



Axe Z de l'accéléromètre

Axe Z du gyroscope

Figure 41. Exemple de données reçu lors d'un enlisage du robot

Nous pouvons constater que les valeurs obtenues sur chacun des axes est très faible du fait que le robot bouge peu. Nous avons choisi d'analyser ces données de la même manière que dans le cas d'une chute. Nous avons donc repris une fenêtre de 8 données sur chacun des axes et mesuré leur écart-type. Les résultats sont présentés dans le Tableau 10.

Tableau 10. Caractéristiques de l'écart-type obtenu sur les échantillons

	Avance normale			sable		
	min	max	moyenne	min	max	moyenne
Ax	0.0224	0.3541	0.0998	0.0071	0.3339	0.0587
Ay	0.0305	0.6044	0.1139	0.0032	0.5570	0.0763
Az	0.0591	1.2871	0.2442	0.0054	0.8055	0.1833
Gx	0.9170	31.5917	6.3440	0.0795	131.0523	8.4243
Gy	1.8020	36.5200	10.8687	0.2235	289.4420	18.0981
Gz	0.4241	19.2819	3.1982	0.1054	95.5526	6.7905

Les résultats obtenus sont assez proche d'une avance normale du robot. Ils sont aussi très différents du cas d'une chute du robot. Cela nous permettra facilement de différencier ces deux cas. Nous prenons la même méthode que précédemment avec les valeurs des coefficients présentés dans le Tableau 11.

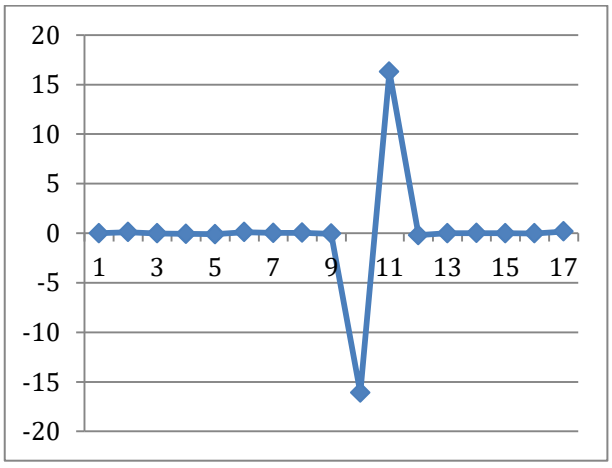
Tableau 11. Coefficient choisis pour détecter un enlisage

Ax	Ay	Az	Gx	Gy	Gz
0.2	0.2	0.1	0.2	0.2	0.1

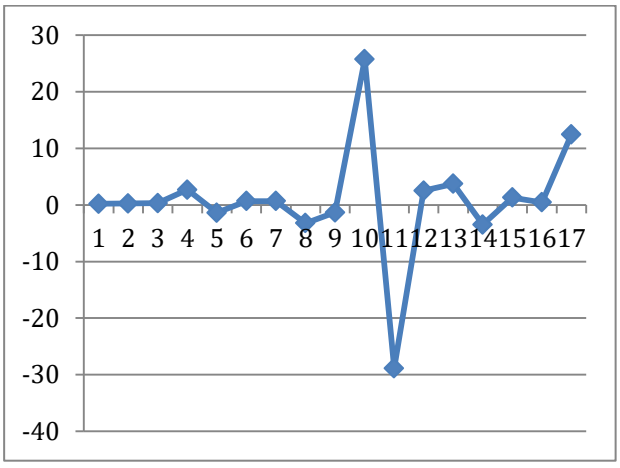
Ces coefficients ont été choisis en fonction du degré d'importance de leur axe. L'axe des Z présente un intérêt moindre puisque l'on travaille dans le plan XY. Ils ne permettent que de distinguer le cas d'un enlisage par rapport à une chute, avec un seuillage effectué sur la valeur obtenue. Si celle-ci vaut moins que 10, on considère qu'il n'y a pas de chute ou d'autre évènements. En plus de ce seuillage, nous devons donc réaliser un autre moyen de différencier le cas d'une avance normale et celui-ci d'un enlisage. Pour cela nous contrôlons la position du robot. Si celui-ci ne bouge plus alors que les commandes des moteurs lui demandent le contraire, nous saurons que les roues sont dans un cas de patinage.

3.5.2.4 DETECTION D'UNE COLLISION AVEC UN OBSTACLE

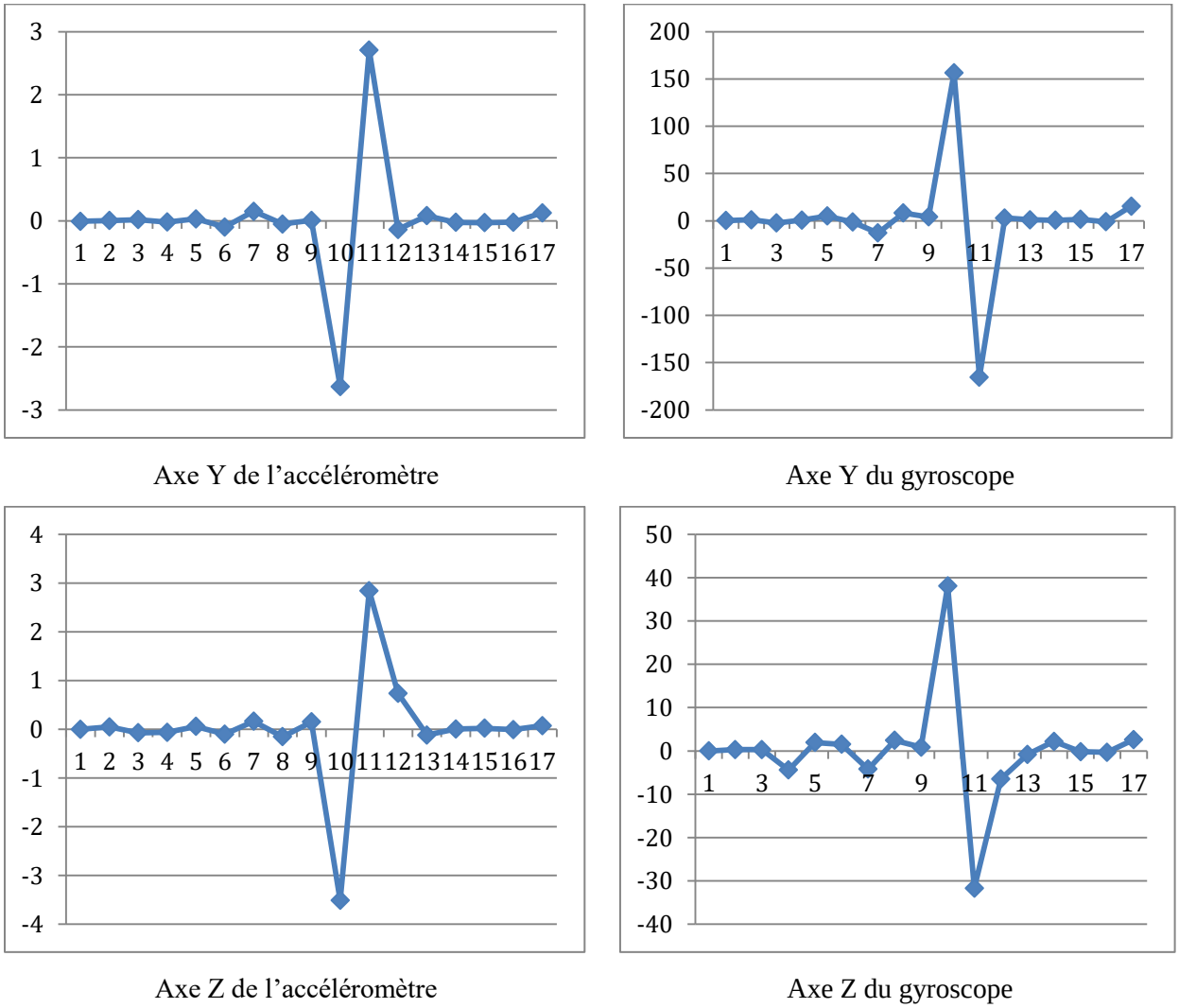
Le dernier test de détection d'événement que nous avons réalisé est le cas d'une collision avec un obstacle. Pour cela nous également réalisé une dizaine de prises de données correspondant à ce cas. Un exemple de données reçues sur chacun des axes lors d'une collision est présenté à la Figure 43.



Axe X de l'accéléromètre



Axe X du gyroscope



Axe Y de l'accéléromètre

Axe Y du gyroscope

Axe Z de l'accéléromètre

Axe Z du gyroscope

Figure 42. Exemple de données reçu lors d'une collision du robot avec un obstacle

Chacun des axes de l'IMU est impacté par la collision, facilitant ainsi sa détection. Cependant le signal de collision étant très bref, il est possible de ne pas le capter du fait du temps de communication des données entre le serveur et le robot. Pour comparer ces résultats à ceux d'une chute, nous avons repris la même méthode que précédemment. Les caractéristiques des écart-types obtenues par la fenêtre sont présentées dans le Tableau 12.

Tableau 12. Caractéristiques de l'écart-type obtenu sur les échantillons pour la collision

	chute			collision		
	min	max	moyenne	min	max	moyenne
Ax	0.1841	2.1664	1.0660	1.0736	8.6609	4.5547
Ay	0.1039	1.0295	0.4515	0.0517	1.4298	0.5954
Az	0.1039	2.8615	0.9917	0.1021	1.7340	1.0250
Gx	5.0755	315.5793	28.3540	1.7210	48.0934	20.5070
Gy	35.7923	666.7851	130.2606	4.1718	124.5035	68.6067
Gz	5.9463	310.6607	45.5385	1.4967	18.9951	9.3133

Les résultats obtenus sont très proches de ceux d'une chute. Il sera donc difficile de travailler avec des écart-types. Nous avons donc décidé de travailler avec uniquement un seuillage sur l'axe X de l'accéléromètre et sur l'axe Y du gyroscope pour détecter une collision frontale. La valeur devrait être bien supérieure à celle d'un cas de chute. Le seuil a été fixé à 2.5 pour l'axe X et 200 pour l'axe Y. Si la valeur reçue est supérieure sur X et inférieur au seuil sur Y, on sera alors dans le cas d'une collision. Dans le cas où on ne détecterait pas cette collision, le robot serait coincé en avance normale contre le bord de l'obstacle. Il ne pourrait plus bouger et c'est cela que l'on chercha à détecter comme dans le cas de l'enlisage.

3.6 CONCLUSION

Dans cette partie, nous avons présenté les différentes étapes nécessaires au bon fonctionnement de notre système. Nous avons justifié nos choix vis-à-vis du matériel employé et de la manière dont nous avons procédé pour parvenir à atteindre les objectifs du cahier des charges. Chacune des parties du système a été élaborées de manière indépendante ce qui a permis de les tester séparément. Ces simulations ont montré qu'il était possible de mettre ce système. Dans la partie suivante, nous vous présenterons les expérimentations menées dessus ainsi que les résultats obtenus.

CHAPITRE 4

EXPERIMENTATIONS

4.1 PRESENTATION GENERALE DU SYSTEME

Dans cette partie, nous vous présenterons les différents tests effectués sur notre système. Celui-ci fonctionne tel que présenté dans la Figure 44.

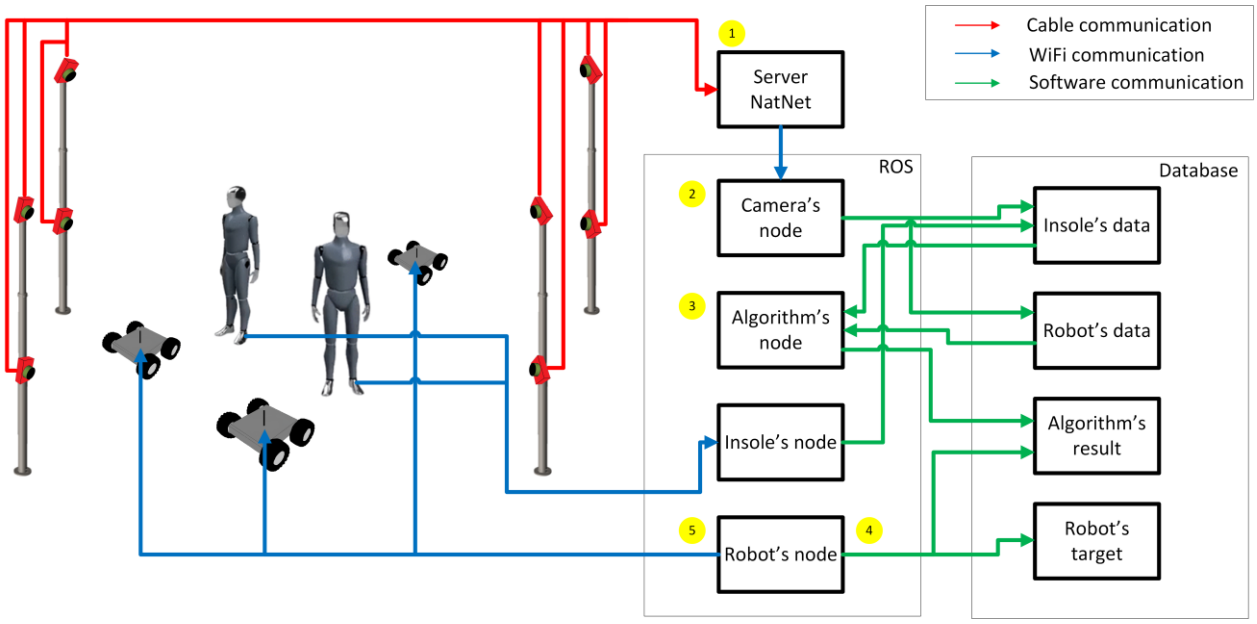


Figure 43. Schéma de fonctionnement du système

Les caméras récupèrent la position des personnes et des robots (1). Ces données sont réceptionnées sur le serveur grâce à un programme (2). Celles-ci sont envoyées dans une base de données. Parallèlement à cela, l'algorithme de placement des robots va chercher les positions des personnes et des robots (3). Il calcule pour chaque robot une position à atteindre et va la stocker dans la base de données. Chaque robot possède sa propre connexion au serveur. Sur cette connexion, l'algorithme de planification de trajectoire va générer un ordre de déplacement et le stocké dans la base de donné (4). Cette ordre est ensuite envoyé via la WiFi au robot dans lequel l'algorithme de commande des moteurs va le diriger (5). Le système peut être testé avec plusieurs robot et personnes. Les différents tests effectués sont présentés dans la partie suivante.

Pour les réaliser, nous avons utilisé une surface faite de planches en bois de 4.5m sur 3.2m représenté à la Figure 45. Les 8 caméras sont réparties aux quatre coins de la zone de test, 2 par coins l'un étant situé à 1 mètre en dessous de l'autre. Elle remplace le drone puisque nous faisons nos tests en intérieur. Ne disposant pas de semelle intelligente pour faire les tests, nous avons représenté le pied d'une personne par une planche en bois munie d'une dizaine de marqueurs pour le tracking. Le robot en possède 7 marqueurs. Ils sont illustrés à la Figure 46 et Figure 47. La distance entre les robots et les personnes a été fixé à 75cm afin de laisser suffisamment d'espace de manœuvre pour les robots.

Figure 44. Zone de test du système

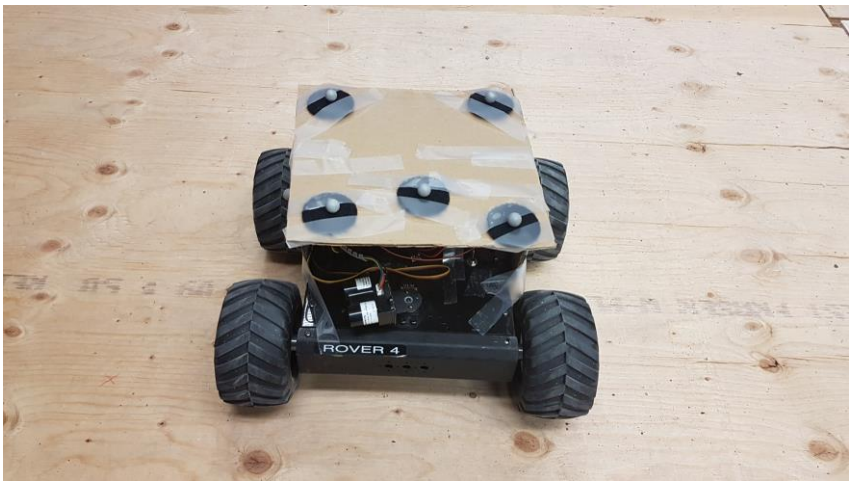


Figure 45. Robot muni de 7 marqueurs

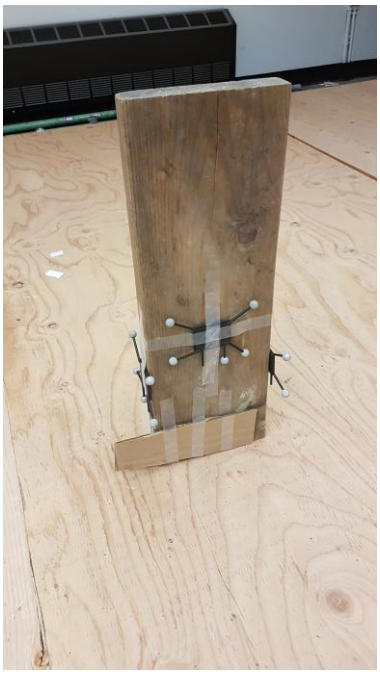


Figure 46. Planche simulant la jambe d'une personne

4.2 MOUVEMENT DU ROBOT

Afin de tester le système, nous avons mis en place différentes situations nous permettant de contrôler le bon déroulement de nos algorithmes. Ces situations sont présentées par la Figure 48. Le robot est représenté par un carré vert, la personne par un carré bleu et l'obstacle par un carré rouge.

Tous ces tests ont été réalisés avec succès. De ceux-ci, nous avons trouvé quelques caractéristiques de notre système :

- Le temps écoulé entre deux transmissions de données vers le robot est d'environ 100ms. Ce temps dépend de l'algorithme de planification de chemin, de l'envoi des données via les sockets par WiFi et surtout du temps de traitement de l'algorithme de logique floue qui peut prendre jusqu'à 50ms. Ce délai doit être pris en compte dans la mesure du possible pour anticiper des mouvements brusques ou éviter des obstacles.

- Le temps de réponse du système pour détecter le changement de déplacement d'une personne est d'environ de la seconde. De ce fait les déplacements effectués pour les tests n'avaient pas de vitesse très élevés (20 cm/s)
- Comme prévu par l'algorithme de logique floue, le robot se déplace lentement s'il détecte un obstacle assez proche de lui. Néanmoins nous ne lui avons pas fait comprendre qu'il pouvait accélérer dans certains cas, notamment quand l'obstacle n'était pas devant lui. Il ralentit donc beaucoup sa vitesse de déplacement à l'approche de la position cible car la personne n'y est pas très loin.
- Le robot atteint sa cible avec plus ou moins 10cm de précision, ce qui est acceptable pour un déploiement dans un environnement assez vaste.

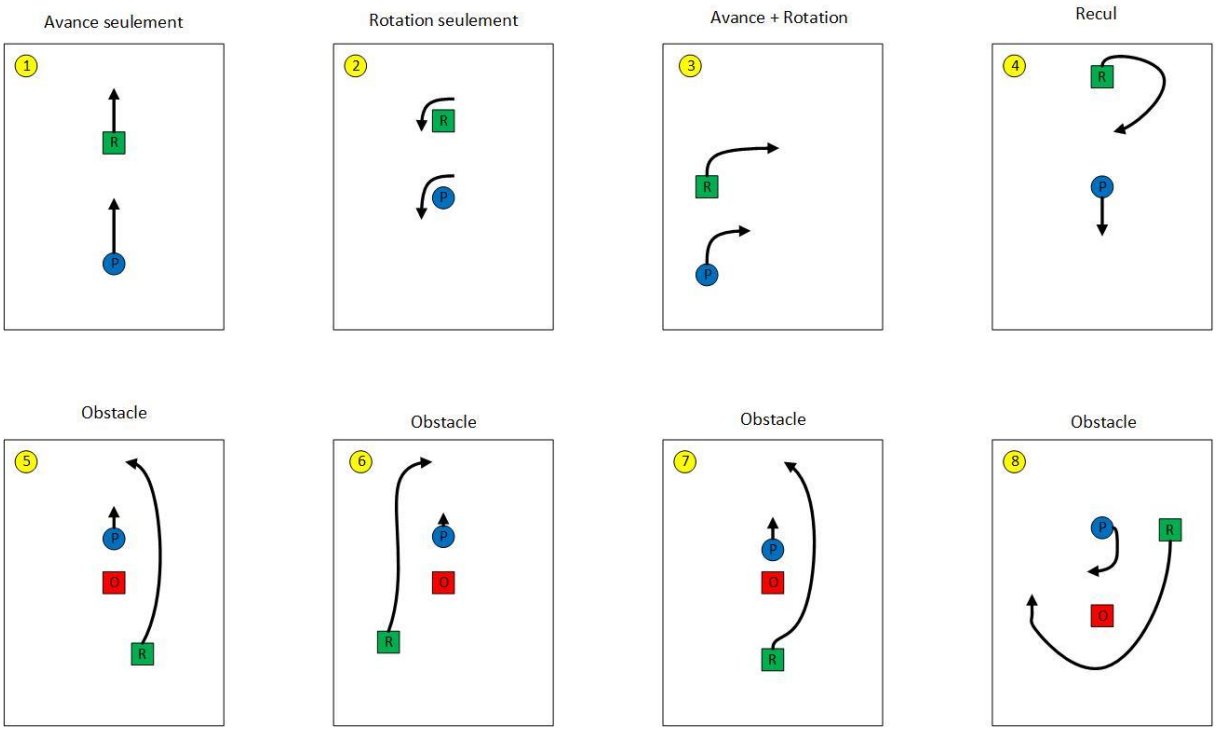


Figure 47. Différents tests réalisés sur le système

Voici un exemple détaillé des différentes commandes envoyées, des positions atteintes et orientation suivis par le robot pour rejoindre une position cible avec une translation et une rotation.

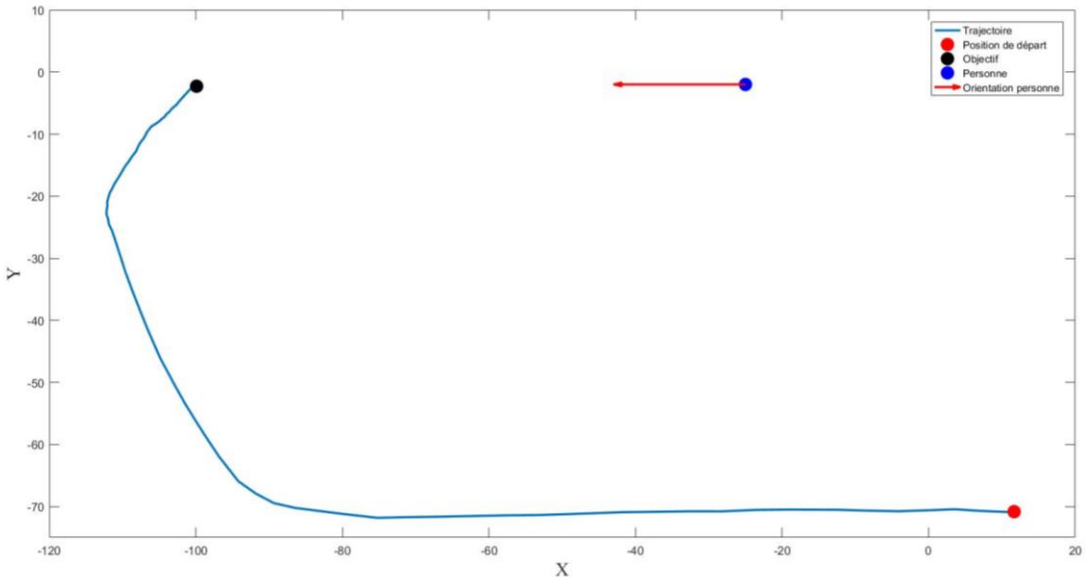


Figure 48. Exemple de trajectoire du robot

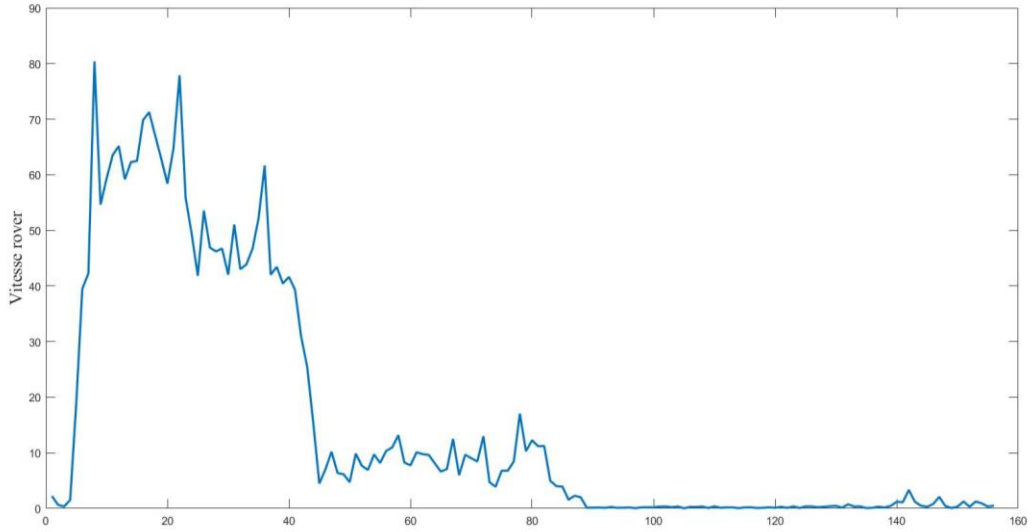


Figure 49. Vitesse du robot en cm/s lors de son déplacement

Le robot a été placé à la gauche de la personne. Afin de la protéger, il devait se placer devant celle-ci. Il a donc cherché à tourner dans la bonne direction pour atteindre la position cible désigner, situé à 75 cm de la personne. Ci-dessous sont présentés les différents paramètres calculés par le serveur puis transmis au robot afin que l'algorithme de logique floue donne une commande aux moteurs.

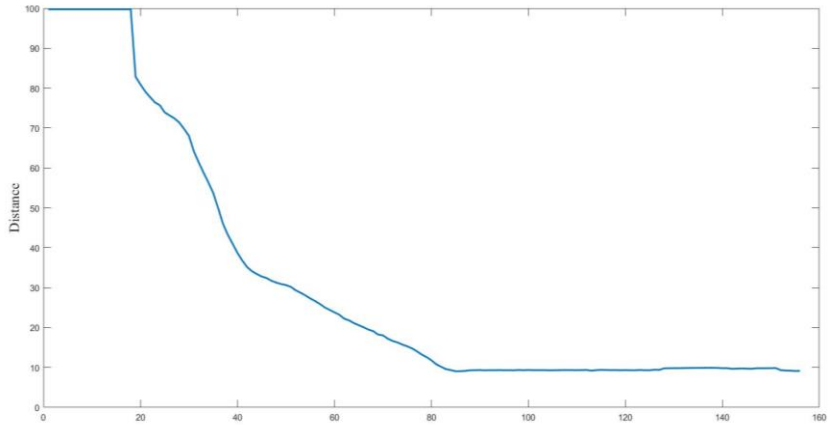


Figure 50. Distance par rapport à la position cible donné par l'algorithme de positionnement

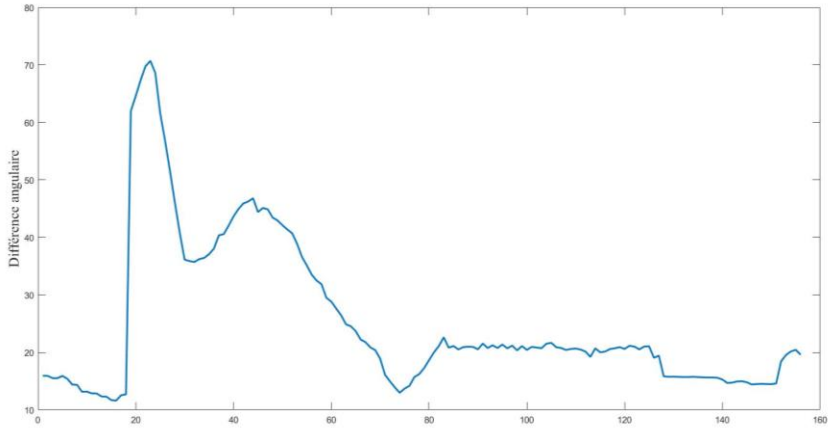


Figure 51. Différence angulaire entre la position du robot et la position cible

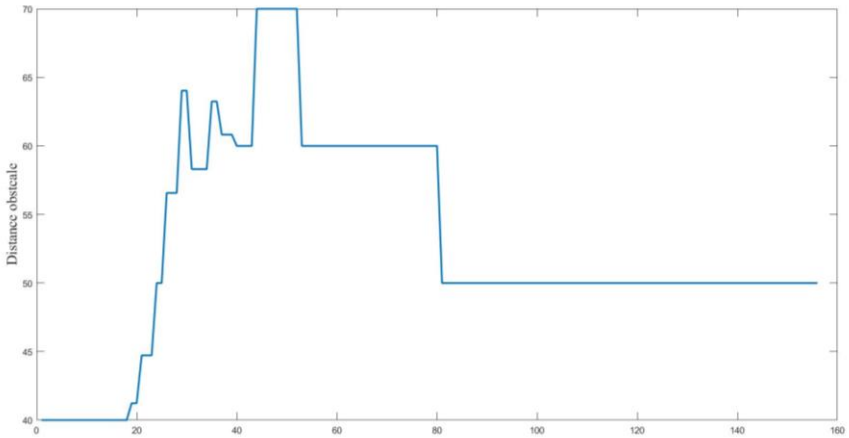


Figure 52. Distance minimale par rapport à un obstacle (dans ce cas la personne)

Ces trois graphiques nous permettent de mieux comprendre le comportement du robot. Au départ, celui-ci a détecté la personne comme étant un obstacle. Il a donc choisit de commencer son trajet en ligne droite pour l'éviter par la suite. Voyant que la différence angulaire entre sa position et la cible devenait trop importante, il a pris la décision de tourner. Comme il se rapprochait de plus en plus de la position cible, il a commencé à ralentir puis il s'est arrêté à moins de 10 cm de celle-ci. La distance finale par rapport à l'obstacle (ici la personne) est bien de 75cm environ, puisque la distance à l'obstacle est d'environ 50 cm à laquelle il faut rajouter 20cm pour modéliser la forme de la personne ainsi que la morphologie du robot comme nous l'avons expliqué précédemment dans le chapitre 3.

4.3 DETECTION D'EVENEMENTS

4.3.1 DETECTION DE LA CHUTE DU ROBOT

Afin de tester notre moyen de détection de chute, nous avons repris les mêmes conditions de tests que lors des acquisitions de données dans le chapitre précèdent. Un exemple de chute est montré à la Figure 54.

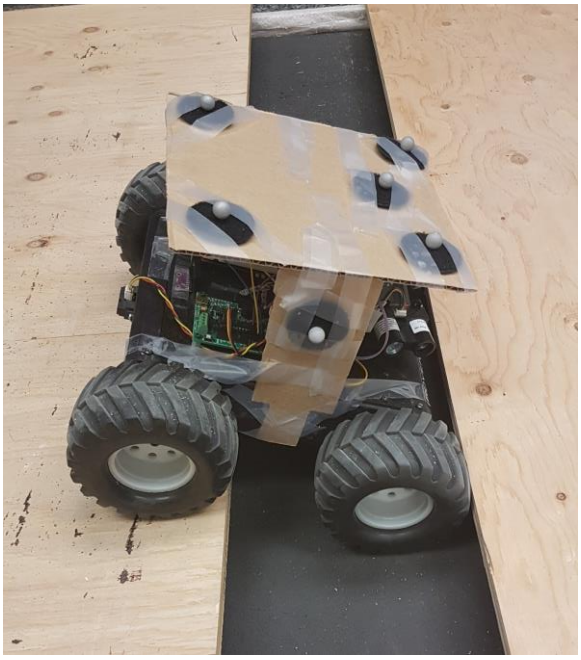


Figure 53. Chute frontale réalisée lors des tests

Sur une dizaine de tests, tous on réussit à détecter la chute. Une fois que le robot a chuté, le serveur le détecte avec la méthode de calcul sur les données de l'IMU et il ordonne au robot de stopper ses moteurs. Un exemple de données issues du robot est présenté dans les Figures ci-dessous.

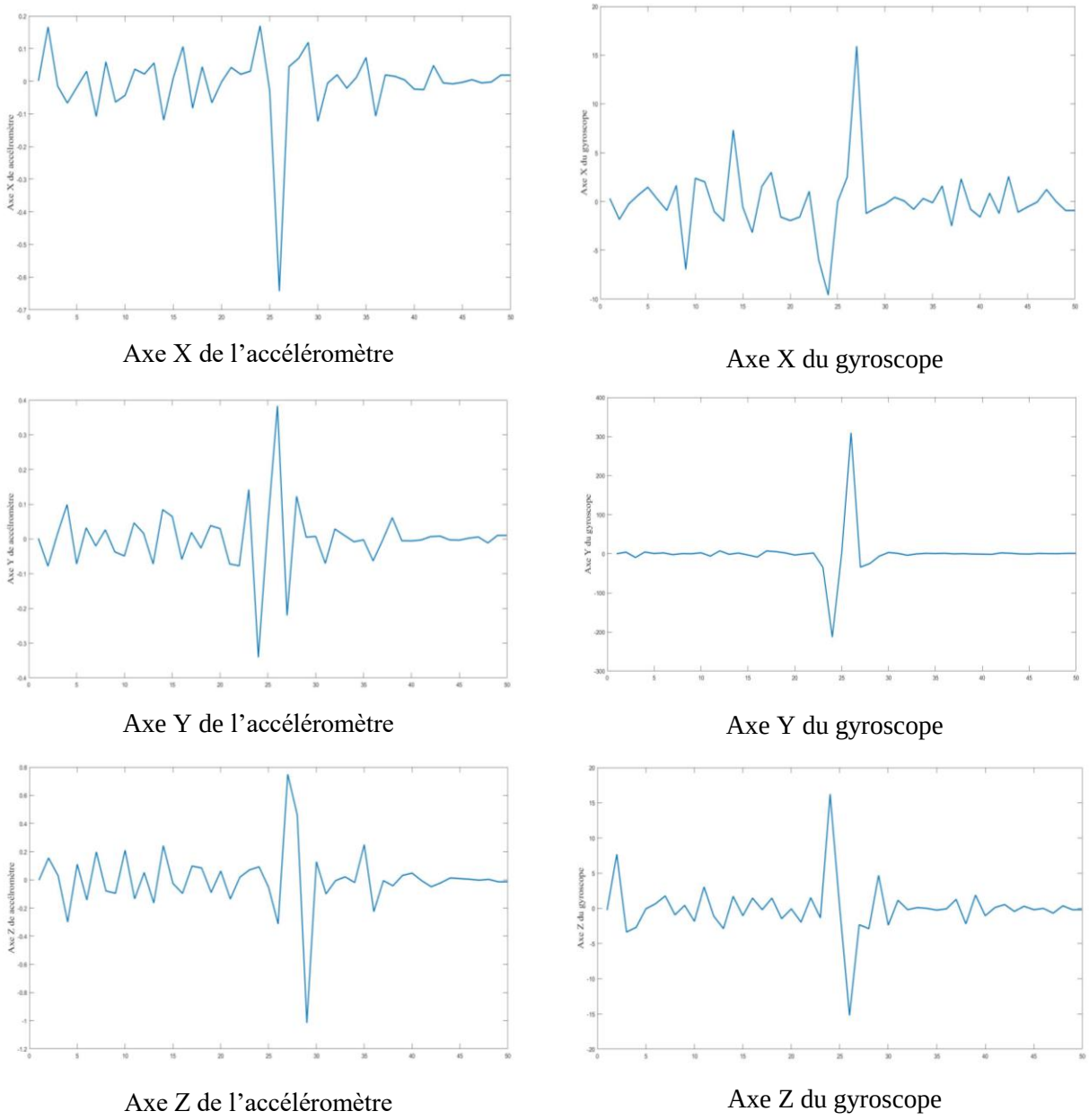


Figure 54. Données de l'IMU lors de la chute du robot

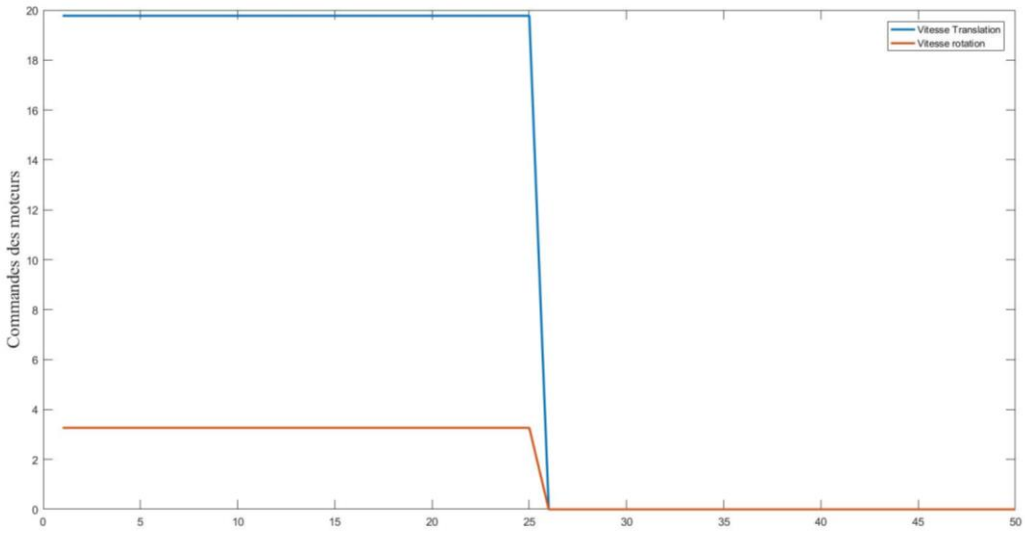


Figure 55. Commandes des moteurs lors de la chute

Au moment 26, le serveur a repéré la chute et a stoppé les moteurs comme le montre la Figure 56. Ce type de détection ne marche que sur des chutes frontales et assez fortes. Pour d'autres types de chutes, il faudrait faire une étude de chacun des cas et vérifié que cette solution fonctionne également. Pour certains types de chutes, le robot peut encore être fonctionnel. Nous ne prenons pas en compte ce type de cas puisque nous travaillons dans un environnement 2D dans ce projet. L'opérateur pourra apercevoir dans la base de données que le robot a chuté puisque son état est passé de 1 à 3, comme le montre la Figure 57.

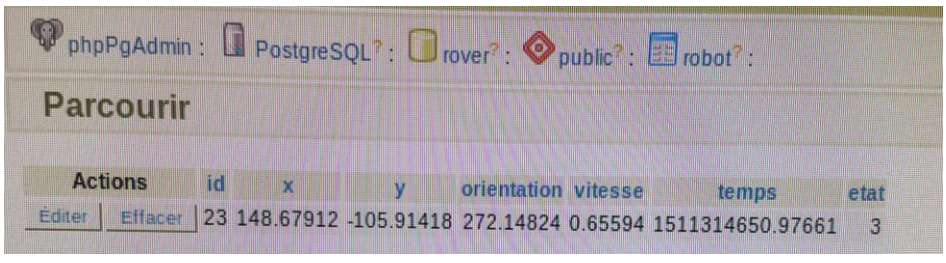


Figure 56. État du robot dans la base de données après la chute

4.3.2 DETECTION DU PATINAGE DES ROUES DU ROBOT

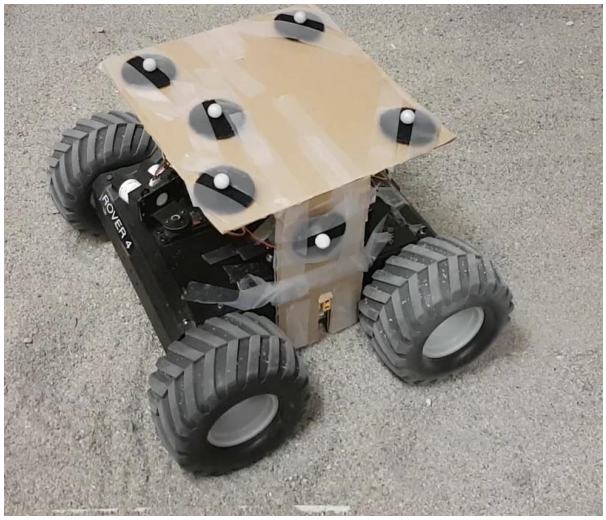
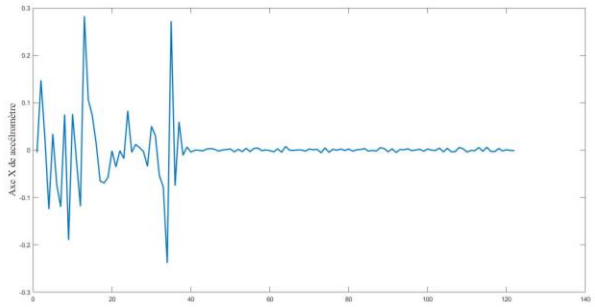
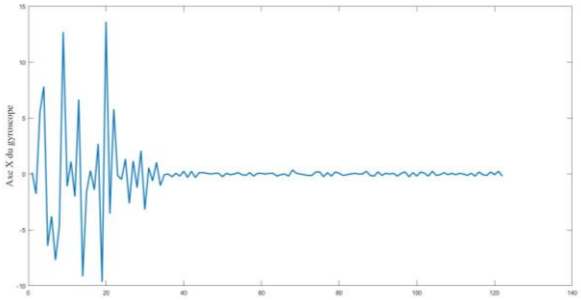


Figure 57. Enlisage du robot dans du sable

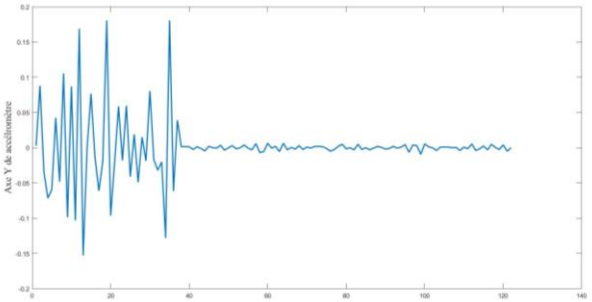
Dans le but de tester la détection du patinage des roues du robot, nous avons placé celui-ci sur du sable à vitesse réduite. Dès qu'il n'arrivait plus à bouger alors que ces moteurs étaient en marche, le serveur prend la décision de couper ses moteurs au bout d'un certain temps. Les données venant du robot sont présentées dans les Figures ci-dessous.



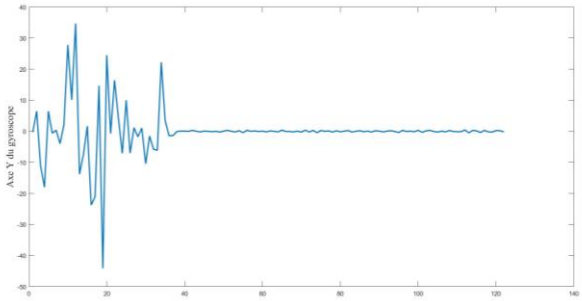
Axe X de l'accéléromètre



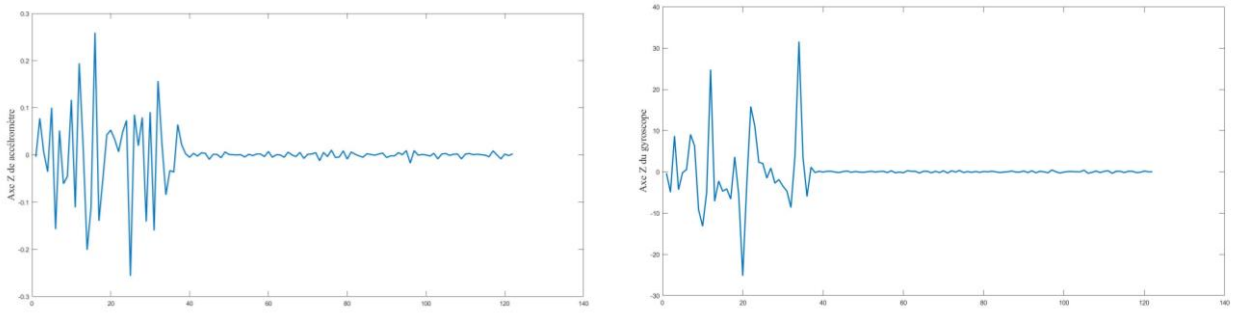
Axe X du gyroscope



Axe Y de l'accéléromètre



Axe Y du gyroscope



Axe Z de l'accéléromètre

Axe Z du gyroscope

Figure 58. Données provenant de l'IMU lors d'un enlisage du robot

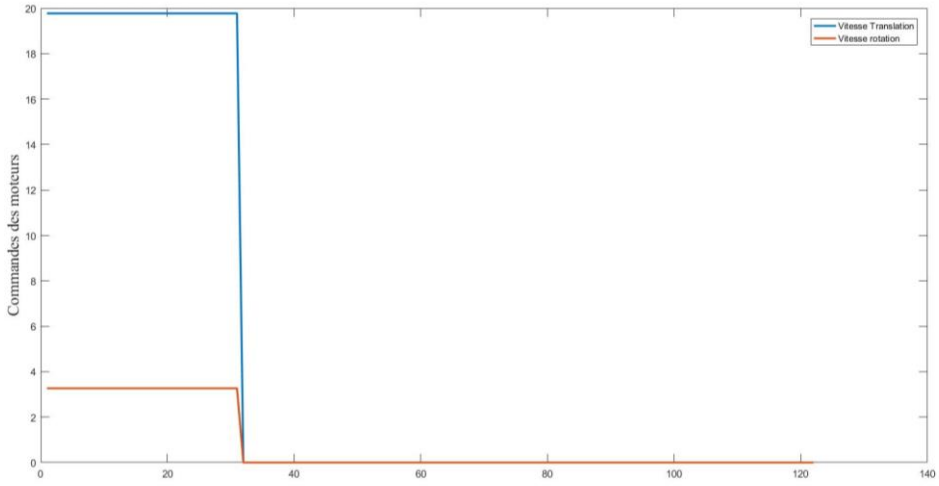


Figure 59. Commandes de moteurs lors de l'enlisage

Les données de l'IMU correspondent à une avance faible. L'enlisage se produit vers le moment 11. Dès lors, le serveur a détecté que le robot était immobile en comparant sa position précédente avec sa position actuelle. Comme les moteurs étaient en marche à ce moment-là, le serveur a attendu quelques secondes afin de voir si la situation allait se résoudre. Au moment 31, il a pris la décision de couper les moteurs puisque la situation n'évoluait pas. L'état du robot est alors passé de 1 à 4 dans la base de données.

Actions	id	x	y	orientation	vitesse	temps	etat
Éditer Effacer	23	34.40516	-69.98923	93.34177	0.58508	1511316308.73785	4

Figure 60. Changement de l'état du robot après l'enlisage

4.3.3 DETECTION DE LA COLLISION AVEC UN OBSTACLE

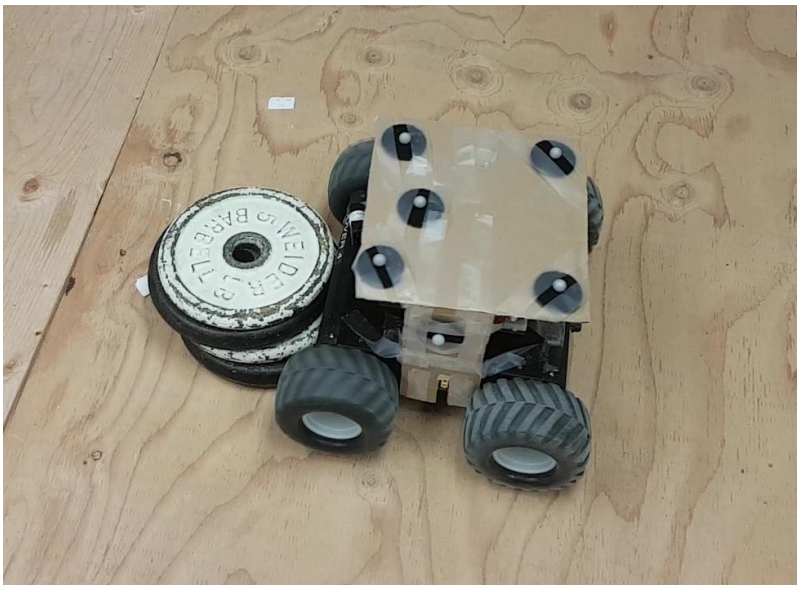
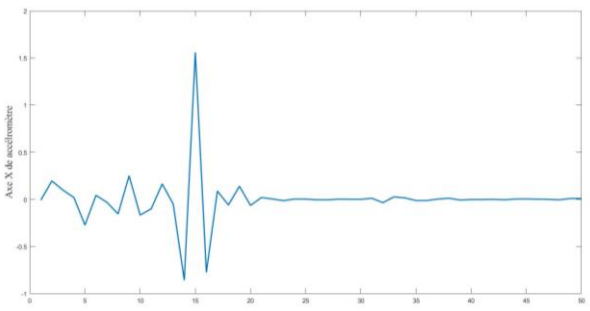
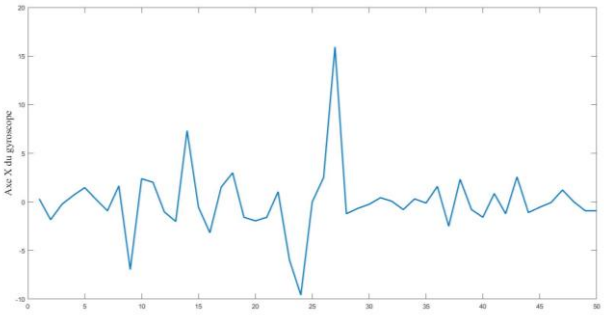


Figure 61. Exemple de collision du robot avec un obstacle

Pour détecter ce dernier type d'événement, nous avons placé un obstacle au milieu du trajet du robot. Cet obstacle n'est pas présent dans la base de données. Le serveur ne peut donc pas le détecter. Comme nous l'avons expliqué dans le chapitre 3, il est assez difficile de détecter ce type de collision puisque la fréquence de réception des données de l'IMU sur le serveur est trop faible. Sur une dizaine d'essais, nous avons pu détecter qu'une fois la collision. Les neufs autres essais se sont terminés par la détection du patinage des roues du robot puisque celui-ci ne bougeait plus. Les données de l'essai réussi sont présentées dans les figures ci-dessous.



Axe X de l'accéléromètre



Axe X du gyroscope

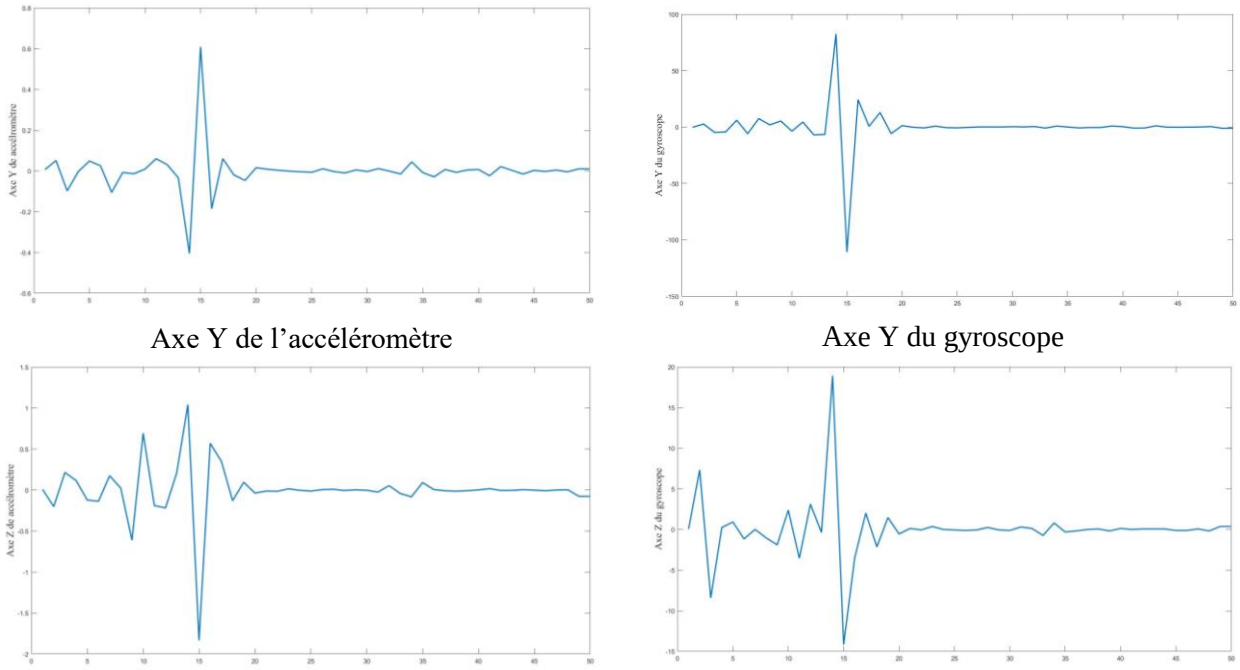


Figure 62. Données de l'IMU lors d'une collision

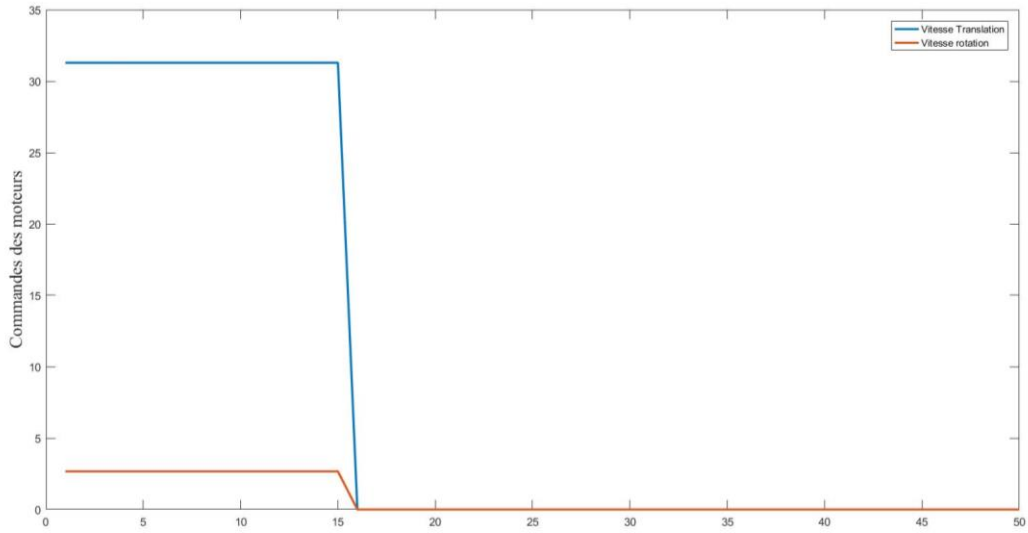
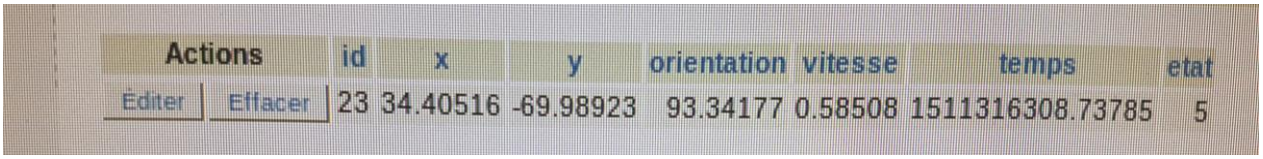


Figure 63. Commandes des moteurs lors de la collision

Les données de l'IMU montrent qu'une collision a eu lieu à l'instant 15. Le serveur l'a détecté et a donc immédiatement stopper les moteurs pour éviter que le robot s'abîme en heurtant une nouvelle fois l'obstacle. Afin d'obtenir cette détection, il a fallu faire énormément de tests pour l'enregistrer. La plupart du temps, le serveur réceptionnait des données de l'IMU montrant une

chute du fait de la durée trop longue des communications. Dans ce cas, il stoppait quand même le robot. Sur 20 essais, 2 ont été correctement menés, détectant ainsi la collision.



Actions	id	x	y	orientation	vitesse	temps	etat
Editer Effacer	23	34.40516	-69.98923	93.34177	0.58508	1511316308.73785	5

Figure 64. Changement d'état dans la base de données lors de la collision

4.4 CONCLUSION

Ces expérimentations ont été menées dans le but de montrer le fonctionnement de notre système. Dans un premier temps, nous avons étudié les différents déplacements du robot en fonction de différentes configurations afin de bien vérifié le bon fonctionnement de l'ensemble des algorithmes. Dans un deuxième temps nous avons observé la réactivité de notre système face à différentes situations vécues par notre robot.

Ce chapitre a montré que notre système pouvait être opérationnel bien qu'il y ait encore des améliorations à y apporter, notamment au niveau de la dynamique des robot près des obstacles et du temps de réactivité du système lorsqu'une personne se déplace. De même pour la détection des différents événements, certains ont été plus facilement détecté que d'autres. Afin d'améliorer cela, plusieurs autres études de cas pourraient être entreprises.

CHAPITRE 5

CONCLUSION

Le suivi d'un groupe de personnes par des robots mobiles est encore un sujet de recherches très actif dans le domaine de la robotique. Nous avons pu voir dans le chapitre 2 qu'il existait beaucoup de méthode de planification de trajectoire pour robot, ainsi que des moyens de communication entre l'opérateur et l'essaim de robots.

Certaines de ces méthodes sont utilisées dans des domaines très spécifiques mais peuvent être prises en compte afin d'en développer de nouvelles. Au cours de ce projet, nous nous sommes inspirés de ces méthodes pour mettre en place notre essaim de robots contrôlé par plusieurs algorithmes. C'est cette utilisation de plusieurs algorithmes bien agencés qui permet au système de fonctionner. Nous avons pu valider le fonctionnement de celui-ci au cours du chapitre 4 avec différents tests effectués, tant au niveau de la logique du système que de son temps de réaction. Comme nous l'avons vu, le système développé possède certaines limites. Il serait intéressant de chercher à les repousser pour améliorer ses performances et le rendre plus efficace sur le terrain.

TRAVAUX FUTURS

Tout d'abord nous nous sommes uniquement intéressés au positionnement des robots par rapport à la position des personnes. Nous pouvons imaginer par la suite d'identifier certains patrons de déplacement d'un groupe de personne afin de prévoir leur déplacement à l'avance mais aussi leurs réactions en cas de situations dangereuses. Une telle étude permettrait de mieux connaître le comportement global d'un groupe de personnes afin de coder le système de manière à optimiser la position de l'essaim de robots par rapport aux personnes.

Ensuite, le système que nous avons créé fonctionne en intérieur. Il faudrait continuer de travailler dessus afin de le développer pour une utilisation extérieure. Ainsi les caméras de tracking seraient remplacées par des drones qui pourraient fournir les informations sur l'environnement des personnes et des robots. De plus il faudrait développer le moyen de

géolocalisation pour les semelles et robots afin de pouvoir parfaitement suivre leur déplacement à distance.

Il serait également intéressant de poursuivre le travail sur la détection des différents événements pouvant affecter la tâche de l'essaim de robots. De nos jours, certains systèmes robotiques sont capables de coopérer en groupe pour résoudre certains problèmes comme nous l'avons vu dans le chapitre 2. On pourrait imaginer que si un robot tombe dans un trou, une partie de l'essaim de robots serait prêt à l'aider pour s'en sortir afin d'éviter l'intervention d'un opérateur. De même pour les autres événements détectés. On pourrait également continuer à améliorer la commande des moteurs ou ajouter différents capteurs afin que le robot puisse facilement faire face à ces situations.

Enfin, plusieurs améliorations pourraient être effectuées au niveau de l'algorithme de positionnement des robots. En effet, celui-ci ne prend pas en compte la présence des obstacles dans l'environnement des personnes. Si une position cible devait être assigné à un robot sur la position d'un obstacle, le robot tournerait autour sans jamais l'atteindre. De plus, de meilleurs modèles de positionnement des robots autour des personnes pourraient être à l'étude. Au lieu de les disposer de façon uniforme, on pourrait par exemple les positionner de manière à couvrir une zone plus dangereuse que les autres. Des améliorations sur l'algorithme de planification pourront aussi avoir lieu. On pourrait chercher à valider un chemin jusqu'à sa cible finale en vérifiant que nous ne rencontrons pas de problèmes d'oscillations de trajectoire, un peu à la manière de l'algorithme des états de collisions inévitables présenté dans le chapitre 2.

Ces développements continueront à parfaire le système de protection d'un groupe de personnes par un essaim de robots dans le but d'éviter le plus possible l'intervention d'opérateurs dans des situations dangereuses.

BIBLIOGRAPHIE

[1] L. Gisin, H. D. Doran, and J. M. Gruber, "RoboBAN: A Wireless Body Area Network for autonomous robots," in ICINCO 2016 - Proceedings of the 13th International Conference on Informatics in Control, Automation and Robotics, 2016, vol. 2, pp. 49-60

[2] M. Salayma, A. Al-Dubai, I. Romdhani, and Y. Nasser, "Wireless Body Area Network (WBAN): A survey on reliability, fault tolerance, and technologies coexistence," ACM Computing Surveys, Article vol. 50, no. 1, 2017, Art. no. 3.

[3] W. J. Yi and J. Saniie, "Smart mobile system for body sensor network," in IEEE International Conference on Electro Information Technology, 2013.

[4] I. C. Paschalidis, W. Dai, and D. Guo, "Formation detection with wireless sensor networks," ACM Transactions on Sensor Networks, Article vol. 10, no. 4, 2014, Art. no. 55.

[5] M. S. Kim, S. H. Kim, and S. J. Kang, "Middleware design for swarm-driving robots accompanying humans," Sensors (Switzerland), Article vol. 17, no. 2, 2017, Art. no. 392.

[6] I. C. Paschalidis, W. Dai, and D. Guo, "Formation detection with wireless sensor networks," ACM Transactions on Sensor Networks, Article vol. 10, no. 4, 2014, Art. no. 55.

[7] Robot Operating System, www.ros.org

[8] R. A. Jarvis, "On the identification of the convex hull of a finite set of points in the plane," *Information Processing Letters*, vol. 2, no. 1, pp. 18-21, 1973/03/01/ 1973.

[9] R. L. Graham, "An efficient algorith for determining the convex hull of a finite planar set," *Information Processing Letters*, vol. 1, no. 1, pp. 132-133, 1972.

[10] T. M. Chan, "Optimal output-sensitive convex hull algorithms in two and three dimensions," *Discrete & computational geometry*, no. 16, pp. 361-368, 1996.

[11] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The Quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software*, 1995.

[12] Nilsson, N. J., “A mobile automaton : An application of artificial intelligence techniques”, In 1st International Conference on Artificial Intelligence, pages 509-520, 1969

[13] Takahashi, O. and Schilling, R., “Motion planning in a plane using generalized voronoi diagrams”, IEEE Transactions on Robotics and Automation, 5(2) :143-150,1989

[14] Lavalle, S. M.,"Rapidly-exploring random trees : A new tool for path planning", Technical report,1998

[15] Cakir, M. ,"2d path planning of uavs with genetic algorithm in a constrained environment", In Modeling, Simulation, and Applied Optimization (ICMSAO), 6th International Conference , pages 1-5,2015

[16] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M., "Optimization by simulated annealing", Science, New Series 220 (4598) :671-680,1983

[17] Khatib, O., "Real-time obstacle avoidance for manipulators and mobile robots", In Robotics and Automation, Proceedings, IEEE International Conference on, volume 2, pages 500-505, 1985

[18] Fox, D., Burgard, W., and Thrun, S., "The dynamic window approach to collision avoidance", IEEE Journal on Robotics and Automation, 4, 1997

[19] Minguez, J. and Montano, L., " Nearness diagram navigation (nd) : a new real time collision avoidance approach", In Intelligent Robots and Systems (IROS 2000), Proceedings, 2000 IEEE/RSJ International Conference on, volume 3, pages 2094-2100 vol.3, 2000

[20] Erdmann, M. and Lozano-Perez, T., "On multiple moving objects", Algorithmica, 2 :477-521, 1987

[21] Quinlan, S. and Khatib, O., "Elastic bands: connecting path planning and control", In Robotics and Automation, 1993, Proceedings, 1993 IEEE International Conference on, pages 802-807 vol.2, 1993

[22] Lamiraux, F., Bonnafous, D., and Lefebvre, O., "Reactive path deformation for nonholonomic mobile robots", Robotics, IEEE Transactions on, 20(6) :967-977, 2004

[23] Yang, Y. and Brock, O., "Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation in dynamic environments", In Robotics : Science and Systems II, August 16-19, 2006

[24] Kurniawati, H. and Fraichard, T., "From path to trajectory deformation", In Intelligent Robots and Systems, IROS 2007. IEEE/RSJ International Conference on, pages 159-164, 2007

[25] Delsart, V. and Fraichard, T., “Navigating dynamic environments using trajectory deformation”, in Intelligent Robots and Systems, IROS 2008. IEEE/RSJ International Conference on, pages 226-233, 2008

[26] Fraichard, T. and Hajime, A., “Inevitable collision states a step towards safer robots”, In IEEE-RSJ Int. Conf. on Intelligent Robots and Systems, pages 388-393, 2003

[27] Zhang, P.-Y., Lu, T.-S., and Song, L.-B., “Soccer robot path planning based on the artificial potential field approach with simulated annealing”, Robotica, 22 :563-566, 2004

[28] Hu, C., Wu, X., Liang, Q., and Wang, Y., “Autonomous robot path planning based on swarm intelligence and stream functions”, In 7th International Conference, ICES, pages 277-284, 2007

[29] Montiel, O., Orozco-Rosas, U., and Sepúlveda, R., “Path planning for mobile robots using bacterial potential field for avoiding static and dynamic obstacles”, Expert Systems with Applications, 42(12):5177-5191, 2015

[30] Cacace, J., A. Finzi, and V. Lippiello. “Implicit robot selection for human multi-robot interaction in Search and Rescue missions”, in 25th IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN 2016

[31] J. Abich and D. J. Barber, "The impact of human robot multimodal communication on mental workload, usability preference, and expectations of robot behavior," Journal on Multimodal User Interfaces, vol. 11, no. 2, pp. 211-225, 2017.

[32] Bevacqua, G., et al, “Mixed-initiative planning and execution for multiple drones in search and rescue missions”, in Proceedings International Conference on Automated Planning and Scheduling, ICAPS. 2015.

[33] Zhang, L. and R. Vaughan. Optimal robot selection by gaze direction in multi-human multi-robot interaction. in IEEE International Conference on Intelligent Robots and Systems. 2016.

[34] Nagi, J., et al. Human-swarm interaction using spatial gestures. in IEEE International Conference on Intelligent Robots and Systems. 2014.

[35] Walker, P., et al. Human control of robot swarms with dynamic leaders. in IEEE International Conference on Intelligent Robots and Systems. 2014.

[36] Alonso-Mora, J., et al. Gesture based human - Multi-robot swarm interaction and its application to an interactive display. in Proceedings - IEEE International Conference on Robotics and Automation. 2015.

[37] E. Cardarelli, V. Digani, L. Sabattini, C. Secchi, and C. Fantuzzi, "Cooperative cloud robotics architecture for the coordination of multi-AGV systems in industrial warehouses," Mechatronics, vol. 45, pp. 1-13, 2017.

[38] Kim, J.Y., et al. mROBerTO: A modular millirobot for swarm-behavior studies. in IEEE International Conference on Intelligent Robots and Systems. 2016.

[39] Otsuka, A., et al. Algorithm for swarming and following behaviors of multiple mobile robots. in IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society. 2015.

[40] Kolling, A., et al., Human Interaction with Robot Swarms: A Survey. IEEE Transactions on Human-Machine Systems, 2016. 46(1): p. 9-26.

[41] Yie, Y., M.I. Solihin, and A.C. Kit, Development of swarm robots for disaster mitigation using robotic simulator software, in Lecture Notes in Electrical Engineering. 2017. p. 377-383.

[42] Kapellmann-Zafra, G., et al., Human-robot swarm interaction with limited situational awareness, in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2016. p. 125-136.

[43] Bamberger, R.J., et al. Wireless network communications architecture for swarms of small UAVs. in Collection of Technical Papers - AIAA 3rd "Unmanned-Unlimited" Technical Conference, Workshop, and Exhibit. 2004.

[44] S. Siddaiyan and R. W. Arokiasamy, "DVFH - VFH*: Reliable Obstacle Avoidance for Mobile Robot Navigation Coupled with A*Algorithm Through Fuzzy Logic and Knowledge Based Systems," presented at the International Conference on Computer Technology and Science (ICCTS), Singapore, 2012.

[45] KarimBenbouabdallah and Z. Qi-dan, "A Fuzzy Logic Behavior Architecture Controller for a Mobile Robot Path Planning in Multi-obstacles Environment," Research Journal of Applied Sciences, Engineering and Technology, 2013.

[46] V. Bayar, B. Akar, U. Yayan, H. S. Yavuz, and A. Yazici, "Fuzzy logic based design of classical behaviors for mobile robots in ROS middleware," presented at the 2014 IEEE International Symposium on Innovations in Intelligent Systems and Applications (INISTA) Proceedings, 2014.

[47] Wikipedia, https://fr.wikipedia.org/wiki/Marche_de_Jarvis

[48] GeeksforGeeks, http://www.geeksforgeeks.org/convex-hull-set-2-graham-scan/

[49] UCDenver, http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Convex_Hull_Finding_Algorithms

[50] Stackoverflow, https://stackoverflow.com/questions/13329345/quick-hull-worst-case-explanation

[51] http://www.mage.fst.uha.fr/riviere/rech/these/chap14.html

[52] http://www.utopisland.com/stf/article36.html

[53] https://mappingignorance.org/2013/02/14/rapidly-exploring-manifolds-when-going-from-a-to-b-aint-easy/

[54] ResearchGate, https://www.researchgate.net/figure/278645627_fig9_Figure-3-4-Organigramme-d%27un-algorithme-genetique-Vosniakov-et-al-2010

[55] https://www.calerga.com/products/Sysquake/robotnav.fr.html

[56] Wikipedia, https://nl.wikipedia.org/wiki/Vector_Field_Histogram

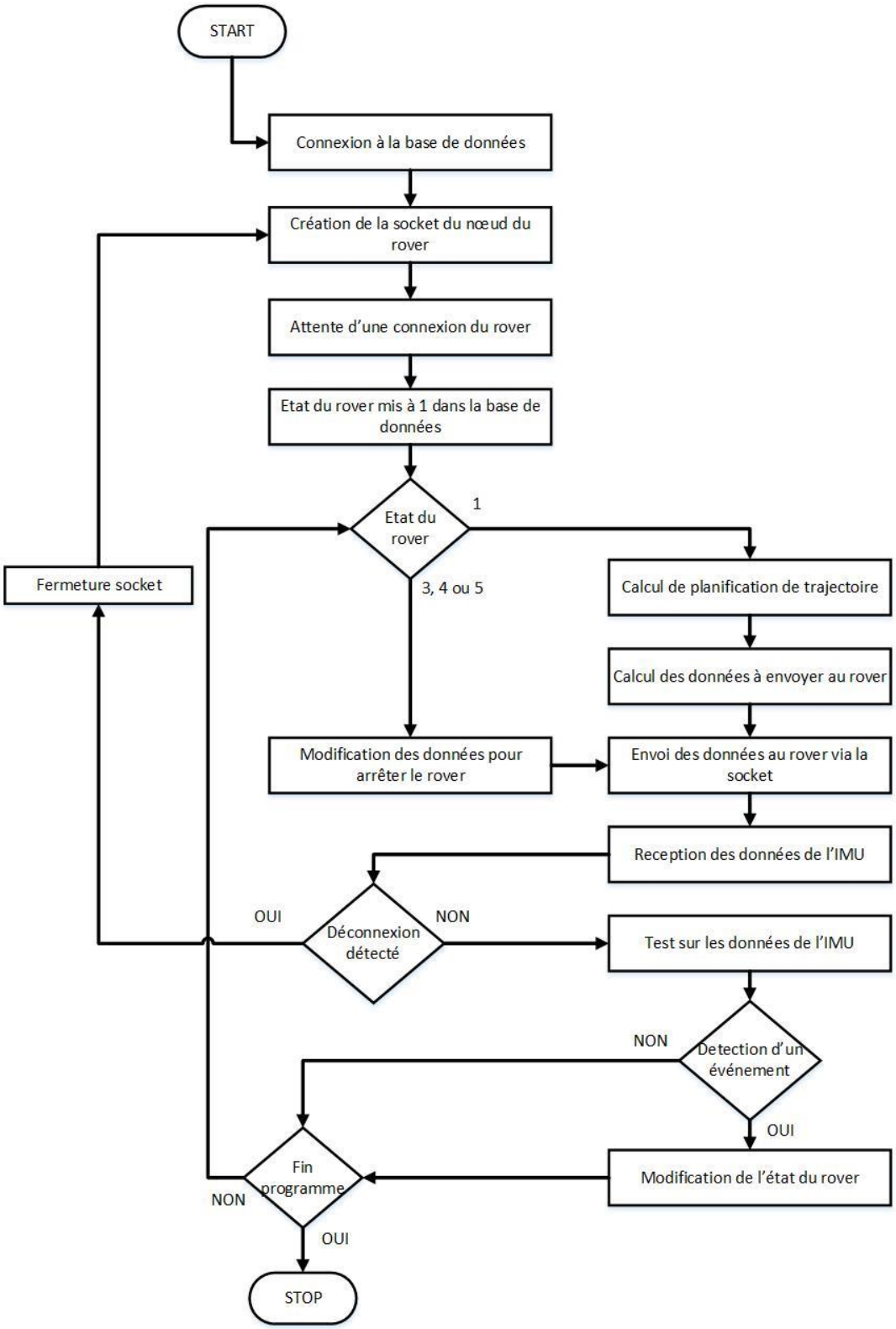
[57] OpenClassrooms, https://openclassrooms.com/courses/introduction-a-la-logique-floue

[58] http://www.pobot.org/Robot-Flou.html

[59] http://slideplayer.fr/slide/1173227/

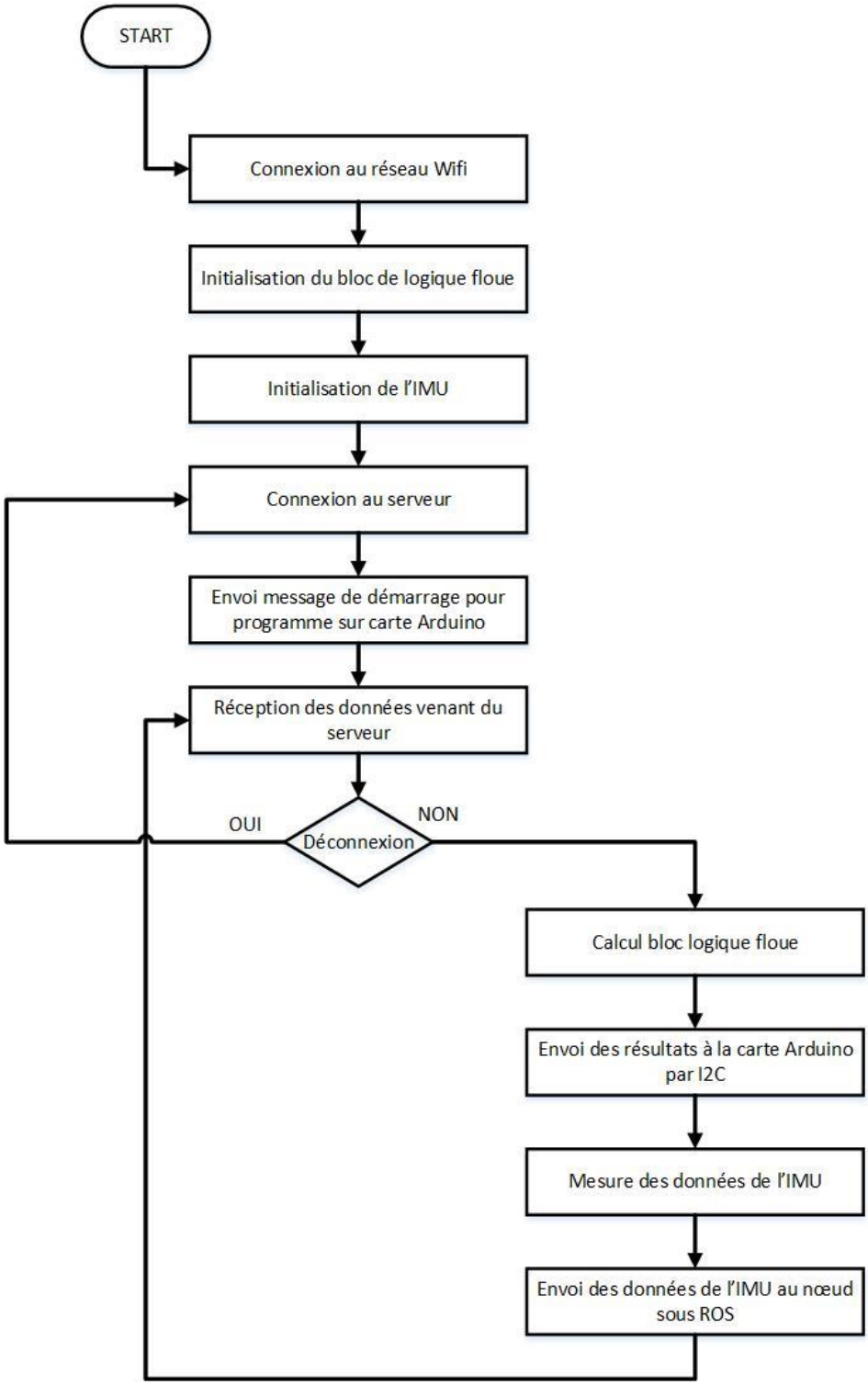
ANNEXE A

ORGANIGRAMME DU CODE DU NOEUD DE ROS POUR LE ROBOT



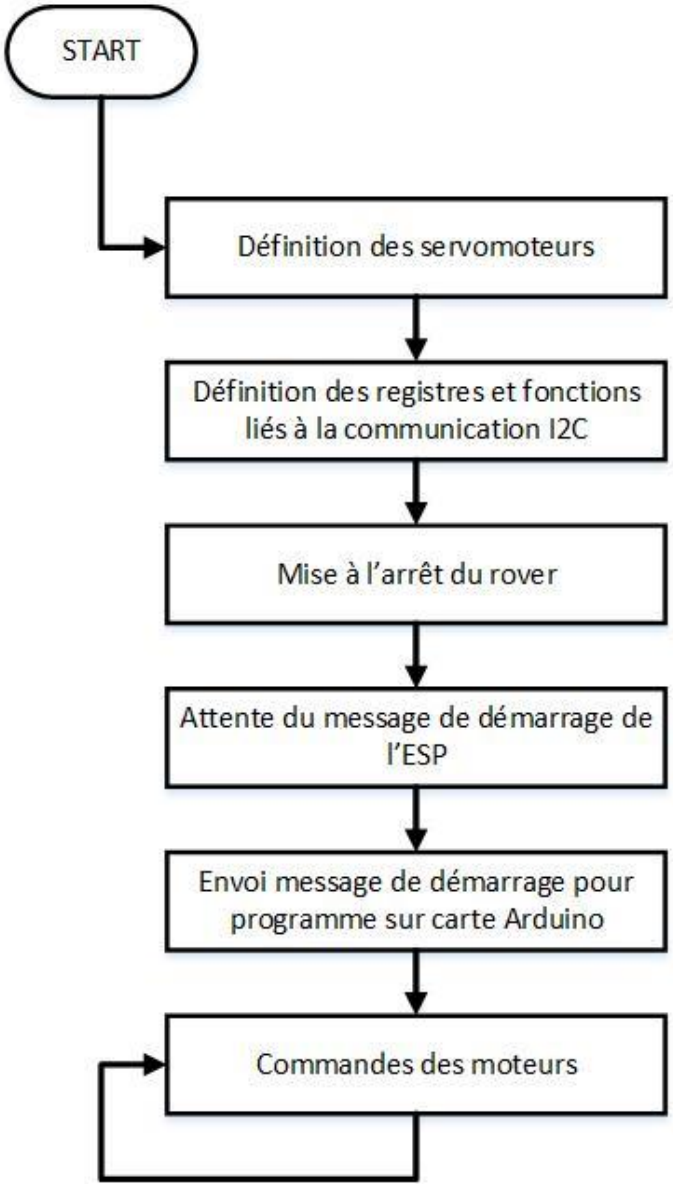
ANNEXE B

ORGANIGRAMME DU CODE DE L'ESP8266



ANNEXE C

ORGANIGRAMME DU CODE DU ROBOT



ANNEXE D

SCHEMA ELECTRIQUE DES COMPOSANTS SUR LE ROBOT