

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE



UNIVERSITÉ DE BLIDA
INSTITUT D'ÉLECTRONIQUE

MÉMOIRE DE MAGISTER

SPÉCIALITÉ : ÉLECTRONIQUE

OPTION : MICRO-ÉLECTRONIQUE

THEME

CONTRIBUTION À LA NAVIGATION DE
ROBOTS MOBILES AUTONOMES
Implémentation de la stratégie sur un circuit FPGA

Présenté par : M^{elle} Ouarda HACHOUR

Proposé par : M^r Karim ACHOUR

Soutenance devant le jury composé de :

M^r K. FERDJANI

Maître de conférence

USTB

Président

RESUME

L'objectif essentiel de ce travail est l'implémentation d'une stratégie de navigation d'un robot mobile autonome dans des environnements inconnus sur un circuit intégré FPGA (Field Programmable Gate Array). L'utilisation des techniques d'Intelligence Artificielle, de la logique floue et des algorithmes génétiques est nécessaire pour offrir un comportement intelligent aux véhicules autonomes similaire au système naturel. Cette approche doit rendre le robot capable d'accomplir ces tâches suivantes : de trouver un chemin vers sa cible et d'éviter les obstacles. Le travail est divisé en deux parties : une partie software et une partie hardware. La partie software a été développée en utilisant le langage de programmation Borland C++. Dans la partie hardware nous allons implémenter la même architecture développée en software. Afin de valider cette architecture nous avons opté pour une approche descendante basée sur la synthèse de Galileo, et comme langage de description nous avons utilisé VHDL (Very High Description Language). Nous proposons une description VHDL flexible et paramétrée qui peut être facilement adaptée à d'autres applications. La description VHDL de l'architecture est implémentée sur un circuit FPGA de la famille XC4000 de Xilinx.

ABSTRACT

The main objective of this work is the design and implementation of a FPGA (Field Programmable Gate Array) integrated circuit for a navigation approach in unknown environments. The use of techniques of Artificial Intelligent, fuzzy logic and genetic algorithms is necessary to bring the behavior of intelligent autonomous vehicle like the human. This approach must make the robot able to achieve these tasks : to make ones way towards its target, and to avoid obstacles. The work is divided in two parts : a software part and a hardware part. The software part has been developed using the Borland C++ language. In the hardware part, we implement the same architecture developed in software. In order to validate this architecture, we have opted for a top down approach based on logic synthesis and using the VHDL (Very High Description Language). We propose a flexible and parametric VHDL description, which can be easily adapted to other applications. The VHDL description of architecture is mapped into the FPGA Xilinx XC4000 family circuit.

INTRODUCTION GENERALE

De tous les temps, l'homme n'a cessé de comprendre et de décrire la nature qui l'entoure telle qu'il l'observe et la ressent. Cette description a été traduite par les fresques, la littérature, la musique, la peinture, la sculpture et d'autres formes d'art. Pour parfaire cette description et d'en saisir le sens, l'homme a mis à contribution toutes ses facultés qu'elles soient morales, physiques ou intellectuelles.

Son instinct dominateur l'a incité à vouloir maîtriser la nature afin de faciliter et d'améliorer son existence dans ce monde. En utilisant son intelligence à la recherche de moyens et de méthodes, il a voulu concevoir une machine à son image. Le premier automate connu date de l'époque pharaonique, c'est une réalisation d'un bras articulé représentant un boulanger pétrissant sa patte. Malgré toutes les difficultés et tous les obstacles rencontrés, l'être humain essaye toujours de mieux faire. Il ne veut pas que la machine imite tous ces faits et gestes seulement, mais aussi qu'elle pense et réfléchisse comme lui.

L'avènement des robots de la troisième génération capables de percevoir leur environnement et de réagir à celui-ci a fait faire un bond en avant aux théories et techniques de la perception. Parmi celles-ci, la robotique mobile présente un grand intérêt public et pratique dans l'univers technologique. Vue l'importance cruciale donnée à ce domaine, cette description a fait actuellement l'objet d'efforts inhérents dans la plupart des pays industrialisés. Les enjeux sont grands, ils sont à la fois :

- **Economiques** : on assiste actuellement à la naissance d'une véritable industrie des connaissances et à l'intégration des systèmes d'automatisation dans les systèmes d'informations et de production existants.
- **Stratégiques** : il s'agit d'exploiter au mieux les connaissances et le savoir faire, richesses souvent cruciales d'une entreprise ou d'une organisation. Les techniques de la robotique et l'Intelligence Artificielle sont aussi un moyen efficace pour maîtriser

l'information et de ce fait , augmenter la productivité et améliorer la qualité de la production.

- *Politiques* : comme l'attestent les grands projets internationaux sur des thèmes communs, etc.

Ceci justifie l'intérêt porté vers la conception et le développement des robots mobiles dans de nombreux laboratoires de recherches dans le monde .

Le pouvoir de faire déplacer un robot mobile d'un point source à un point but en sécurité est le point focal de toutes les recherches en robotique mobile , souvent le robot doit atteindre son objectif. Le robot autonome doit , avant tout, être capable de décider de ses actions de façon raisonnée , puis les exécuter dans un contexte réel [1]. Réagir aux événements imprévus peut conduire à modifier profondément le plan d'action établi par le planificateur , il faut donc contrôler à tout moment l'activité du robot pour assurer une bonne réactivité. En toute globalité , la navigation peut être résumée dans les réponses à ces trois questions : " ou` suis - je ? " , " ou` est-ce- que je vais , ? " , et " comment je vais atteindre mon but ? " . En outre , le mot clé de la robotique de troisième générations réside sur ce qu'on appelle l'autonomie , celle ci implique des capacités de perception de modélisation de l'environnement et de prises de décisions. Pour cela un robot est dit autonome s'il satisfait les conditions suivantes :

- Déplacement dans un environnement connu ou partiellement connu.
- L'autonomie qui s'explique par L'Intelligence et sources de puissances contenues dans un corps fermé.
- Compréhension de l'environnement qui est la reconnaissance du monde.

Afin d'assurer aux véhicules plus d'autonomie et plus d'intelligence avec des capacités de traitement en temps réel, l'utilisation des circuits ASICs comme les circuits FPGAs – **qui est l'objectif essentiel de notre présent travail** - est nécessaire pour compléter les lacunes et offrir un système robuste et rigoureux et ceci avec une rapidité accrue par rapport aux progrès de Soft- Computing basés sur les circuits à usage générale. Une nouvelle architecture s'impose alors en prise en charge des systèmes hybrides intelligents basés sur les algorithmes génétiques, la logique floue et les techniques d'Intelligence Artificielle pour la planification et le contrôle des robots mobiles. Cette méthodologie s'est alors mise en œuvre en utilisant le monde passionnant de la micro- électronique et qui a permit de dégager les meilleurs systèmes, en termes de facilité de mise en œuvre et de performances pour chaque comportement intelligent.

La technologie des FPGAs dédiée à l'implémentation d'une stratégie de navigation d'un robot mobile, nous a ramené à opter pour une approche basée sur une synthèse de haut niveau comme Galileo et une description hardware comme VHDL (*Very High Description Language*). Cet outil est le plus important pour un micro-électronicien pour qu'il puisse concevoir et réaliser un circuit intégré. Ce langage permet par son pouvoir de haut niveau de transformer un circuit modélisé en une réalisation physique du circuit. Selon les performances requises de l'application, le comportement de notre robot peut être mieux fait en hardware qu'en software avec un temps adéquat et convenable pour répondre encore plus aux exigences d'autonomie

dans des environnements parfaitement inconnus de manière similaire au système naturel et qui peut atteindre les meilleurs performances d'ordre pratique et théorique.

Notre travail s'inscrit dans le cadre d'un projet de recherche à la fois du laboratoire de Robotique et d'intelligence artificielles et du laboratoire de la micro-électronique du centre de développement des technologies avancées *CDTA* à *EL Madania*.

Ce projet qui s'intitule : "*Contribution à la navigation de robots mobiles autonomes – implémentation de la stratégie sur un circuit FPGA*» a pour objectif de simuler un planificateur local des trajectoires pour mener le robot de la source à la destination en évitant les obstacles dans le sens "*optimal*". L'objectif essentiel de ce travail se focalise en l'implémentation d'une stratégie de navigation d'un robot mobile autonome sur une carte électronique dotée du circuit FPGA. La partie Software de notre conception (progiciel) a pour but l'étude de la faisabilité ce qui va être implémenté en hardware.

Pour décrire nos travaux , nous avons organisé ce mémoire en quatre (04) chapitres.

Le premier chapitre expose la navigation des robots mobiles.

Le deuxième chapitre est destiné à exploiter les performances de la logique floue et les algorithmes génétiques. Ces deux outils ont été utilisés pour optimiser et contrôler une trajectoire donnée.

Le troisième chapitre décrit les circuits ASICs et les circuits FPGAs. Ces circuits nécessitent des connaissances de base sur cette technologie, une synthèse de haut niveau comme Galileo et un langage de description hardware comme le VHDL. Cette implémentation répond parfaitement aux exigences d'autonomies en un temps adéquat emballé par une rapidité accrue .

Le quatrième chapitre décrit l'analyse, le développement et la simulation proposée à la navigation des robots mobiles autonomes. Nous y détaillons toutes les étapes utilisées lors de la conception software et hardware. A cet effet, nous avons consacré une section pour la présentation de notre progiciel appelé AG-LF et une section pour la présentation de circuit FPGA avec une description et un traitement de la fonctionnalité. Pour chaque partie, nous présentons les résultats obtenus.

Nous terminons par une conclusion générale et par des perspectives suites à notre travail.

Chapitre 1

LA NAVIGATION DES ROBOTS MOBILES

1.1. INTRODUCTION

La robotique mobile expose un domaine qui s'étend rapidement et indéfiniment. Durant ces dernières années, supportées par l'état d'avancement de la technologie, plusieurs recherches ont été faites dans le cadre des robots mobiles autonomes articulées autour de deux axes principaux, à savoir la navigation et le guidage. On assiste également à une exploitation raisonnée de grandes bases de connaissances, ce qui constitue une des caractéristiques fondamentales de la robotique mobile qu'elle est fondée sur cette exploitation.

1.2. BREF HISTORIQUE

Le terme *Robot* est usuellement associé à la réalisation des tâches de façon automatique. Ce mot tire sa racine du bulgare *robu* qui signifie serviteur et qui a aussi donnée de le tchèque *robota* qui se traduit par travail forcé. C'est d'ailleurs par l'écrivain *Karel Capek* que c'est popularisé le terme du robot, nom qu'il avait donné à des petits artificiels (androïdes) évoluant dans sa pièce de théâtre *Rossum's Universal Robots* et qui répondaient parfaitement aux instructions de leur maître [2]. Cette pièce, restée célèbre par cette invention décrivant une lutte dans la société avec les automates travailleurs « *Ouvriers* » [3].

En 1961, *George Devol* et *Engelberger* ont pu mettre en évidence les procédés d'inventions des robots industriels, dans le même concept, le premier robot " *contrôlé par l'ordinateur* " a été développé par Ernest à Mit. Et c'est vers la fin des années soixante 1960 que le concept de la robotique mobile est apparu. Pour essayer en vraie grandeur les principes développés par les chercheurs, on construit *Shakey*. C'est une réalisation d'une machine à roues, connectée à un gros

ordinateur, et qui évolue dans un univers de cubes et de pyramides de tailles et de couleurs différentes, ses missions : prendre un objet et le porter ailleurs, quelques soit sa position absolue dans la salle d'évolution, relative par rapport à d'autres objets. Ses moyens de perception : essentiellement une caméra qui lui permet d'acquérir des images de son environnement. Ses performances : une cinquantaine de minutes pour effectuer une " mission " [4].

1.3. DEFINITION

Il existe plusieurs définitions pour ce terme : Larousse par exemple donne une définition (*robota*: corvée en tchèque) c'est à dire appareil capable d'agir de façon automatique par une fonction donnée, le petit Robert l'explique par une machine à l'aspect humain capable de se mouvoir et d'agir. D'autres définitions ont lui été contribuées, parmi celles-ci on trouve « *Manipulateur commandé en position, reprogrammable, capable de manipuler des matériaux, des pièces, des outils et des dispositifs spécialisés, au cours de mouvement variables et programmés pour l'exécution d'une variété de tâches* ».

Mais dans la conversation courante, ce terme couvre d'autres représentations, moins officielles : nous définissons aussi un robot mobile comme étant un système automoteur, disposant de moyens de traitement de l'information permettant une capacité décisionnelle suffisante de façon à pouvoir exécuter, sous contrôle humain un certain nombre de tâches précises, dans un environnement variable, non complètement connu à l'avance.

1.4. MOYENS DE PERCEPTION

Vu la complexité de l'espace dans lequel les robots sont amenés à évoluer et vu que l'environnement de ces automates est vaste, variable et peu ou pas structuré, plusieurs dispositifs ont été mis à la contribution de la perception qui résident sur les différents senseurs [4,5]. Ces derniers permettent aux robots de comprendre la structure de l'univers de la navigation, il serait donc nécessaire de concevoir des systèmes - perception - qui permettent de mesurer et de contrôler en permanence la position par rapport aux tâches qu'elles sont demandées. Ces dernières années, l'évolution de la technologie a enrichie le matériel de perception qui peut être présenté en deux classes:

1.4.1. Systèmes télémétriques passifs

Ces types concernent les capteurs tactiles et les caméras qui fournissent une vue panoramique de l'environnement. La télémétrie passive fournit une description de l'environnement très détaillée mais le nombre d'informations à traiter par l'ordinateur est très important et conduit à un temps de calcul très élevé.

1.4.2. Systèmes télémétriques actifs

Dans ce cas les capteurs ne fournissent pas une vue panoramique de

l'environnement, nous pouvons citer à titre d'exemple les capteurs ultrasonores qui émettent des ultrasons et déterminent la distance à un obstacle par des mesures effectuées sur l'écho renvoyé par l'obstacle. Les ultrasons retournent moins d'informations à traiter mais ce moyen de perception est plus sensible aux distorsions et aux perturbations au monde extérieur.

1.5. DECOMPOSITION CANONIQUE D'UN ROBOT MOBILE AUTONOME

A partir de la définition proposée dans le paragraphe (1.3), on peut, en premier temps, représenter un robot mobile autonome suivant le schéma de la figure 1.1. Trois sous systèmes principaux y apparaissent : le sous - système mobilité, le sous-système charge utile, le sous -système contrôle / commande / communication / décision ou C³D [4].

- Le premier associe l'ensemble des éléments nécessaires (électroniques, etc.) permettant d'assurer l'autonomie de mouvement nécessaire à l'accomplissement de la mission.
- Le second sous- système est le plus spécifique de la mission du robot. Ce peut être un outil de nettoyage (cas d'un robot *nettoyeur*), une caméra (avec ou sans traitement d'image) dans le cas d'un robot de *surveillance*, etc.
- le troisième sous- système regroupe l'ensemble des moyens d'acquisition ,de traitement et de calcul qui permettent au robot d'avoir une certaine d'autonomie décisionnelle.

1.5.1. Contrôle .Commande .Communication

Les fonctions du sous-système C³D sont celles qui vont qualifier une machine comme étant un robot mobile *autonome*. Il est nécessaire , dès maintenant, de bien mettre en évidence un certain nombre de notions clés relatives à certains concepts. Pour simplifier la compréhension du sujet, nous serons amenés à scinder en deux grandes parties la description du C³D : d'un côté les problèmes liés à la planification des actions, de l'autre côté, les techniques utilisées pour doter le robot de capacités de perception de l'environnement .

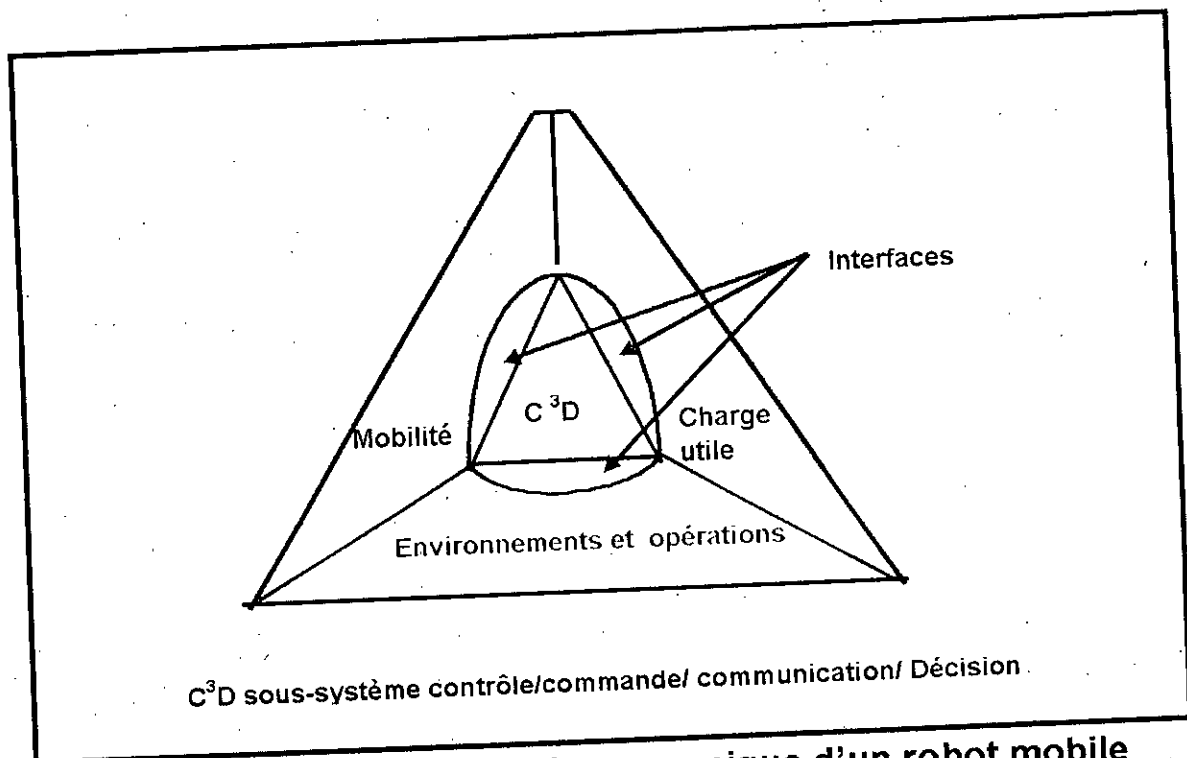


Figure 1.1 : décomposition canonique d'un robot mobile autonome

1.5.2. Problèmes de prises de décision

La commande et le contrôle des déplacements, ainsi que la perception de l'environnement du robot mobile autonome sont des éléments essentiels du système C^3D . Nous allons décrire ces éléments rapidement dans ce paragraphe.

Exemple

Supposons que nous donnions à un robot mobile qui opère des lieux (constituant le plan d'atelier) la mission suivante : effectuer l'opération O_1 sur les machines M_1 de l'atelier A_3 , la démarche à suivre peut se décomposer en une suite de tâches d'analyse et de prise de décisions. Celles-ci repassant dès la réception de l'ordre, et une partie d'elles vont continuer pendant toute la durée du travail s'agissant des opérations initiales, on aura l'analyse suivante

- Qu'est ce que l'opération O_1 ? en particulier, ya t- il besoin pour l'accomplir d'un objet spécial ? ou se trouve la machine M_1 ?
- Constatations de l'état du robot : position, état de charge des batteries, matériel disponible à bord, etc.
- Génération d'un plan d'action d'abord grossier (aller chercher l'objet manquant, puis rejoindre M_1 et effectuer le travail requis).

Cette **génération de plans** est relativement classique, et pendant toute la durée de la mission, le robot devra **contrôler l'exécution** de ce plan, étape par étape. Si le robot va rencontrer un obstacle inattendu il est nécessaire donc de faire une modification locale de trajectoire pour le contourner, autrement dit, dans ce cas il

va falloir *remonter* dans le raisonnement jusqu'au niveau de l'ordonnancement des tâches pour rejoindre un problème.

Il est donc nécessaire de doter le robot d'un mécanisme décisionnel dit contrôleur d'exécution dont la fonction va être, à partir des informations recueillies par les différents capteurs (d'environnement mais aussi proprioceptifs), de choisir à quel niveau doit être réévaluée la mission. L'écueil principal de réalisation réside dans le caractère en temps réel imposé par les conditions de travail de la machine. Il pose avant tout un problème d'architecture .

1.6. ROBOTIQUE ET INTELLIGENCE ARTIFICIELLE

Le vaste champ de la robotique mobile englobe l'ensemble des activités d'automatisation de tâches, notamment dans l'industrie (fabrication , manutention, réparation, etc.). Une partie des robots existants (par exemple les robots soudeurs ou peintres sur une chaîne de montage) se contentent de répéter indéfiniment une séquence d'opérations et ne possèdent donc aucune intelligence. En revanche, les nouvelles générations de robots sont de plus en plus dotées de capacités « *intelligentes* » de perception de leur environnement (notamment par la vision) et de planification de leur activité (génération d'une stratégie ou d'un plan d'action pour mener à bien une tâche). Ainsi, L'intelligence Artificielle IA joue un rôle croissant dans la robotique qui constitue une composante majeure de l'industrie de demain .

L'IA est une approche plus récente et très prometteuse qui tente de dépasser certaines de leurs insuffisances. Actuellement, plusieurs équipes de recherches développant des projets inhérents dans le vaste champ de la robotique constituent l'ossature d'un bon potentiel de niveau international. Les interactions entre les experts d'Intelligences Artificielle et de la robotique mobile dont le temps est grand illustre que la plupart des problèmes au niveau de l'automatisation et d'autres tâches de robotique mobile sont résolus en faisant appel à l'AI.

Cette illustration peut être vue à priori, comme la structure d'un plan pour résoudre le problème donné. Les systèmes développés aujourd'hui montrent que les recherches scientifiques en robotique mobile font appel de plus en plus à l'IA [6].

Les architectures qui comportent une fonction de planification en robotique mobile sont issues des techniques de l'Intelligence Artificielle du type tableau noir / tableau blanc, où le modèle de tableau noir propose un schéma pour organiser les domaines de connaissances et les étapes du raisonnement conduisant à la construction incrémentale de la solution au problème posé, ce modèle représenté à la figure 1.2 est composé de trois éléments majeurs [6,7,8] :

- Les sources de connaissances qui sont indépendantes utilisent exclusivement la base de données centrale.
- La base de données centrale ou tableau noir, mémorise, à chaque instant l'état courant de la résolution du problème . Les éléments de solutions sont mis à jour par les sources de connaissances.
- Le système de contrôle qui gère l'activité des sources de connaissances.

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ DE BLIDA
INSTITUT D'ÉLECTRONIQUE

MÉMOIRE DE MAGISTER

SPÉCIALITÉ : ÉLECTRONIQUE

OPTION : MICRO-ÉLECTRONIQUE

THEME

CONTRIBUTION À LA NAVIGATION DE
ROBOTS MOBILES AUTONOMES

Implémentation de la stratégie sur un circuit FPGA

Présenté par : M^{elle} Ouarda HACHOUR

Proposé par : M^r Karim ACHOUR

Soutenance devant le jury composé de :

| | | | |
|------------------------------|----------------------|------|----------------|
| M ^r K. FERDJANI | Maître de conférence | USTB | Président |
| M ^r B. BOUZOUIA | Maître de recherche | CDTA | Examineur |
| M ^r N. AMMOUR | Chargé de cours | USTB | Examineur |
| M ^r M. HADJ RABAH | Chargé de cours | USTB | Co- rapporteur |

Université de Blida, 2001

En 1983, *ELFES* propose une architecture décisionnelle sur le robot de type tableau noir pour le contrôle du robot *CMU ROVER*. Un ensemble de modules experts communiquant par échange de message via un tableau noir pour exécuter une mission sous la direction d'un plan de contrôle .

Dans la figure 1.3 : on démontre une autre recherche dévouée par l'équipe de *C.THORPE*, cette équipe développa en 1986, un prototype *NAVLAB* capable d'effectuer le suivi de route. L'architecture est de type de tableau blanc, pour se démarquer du modèle tableau noir, cette architecture permet un accès asynchrone à la base de donnée pour y déposer ou en retirer des informations.

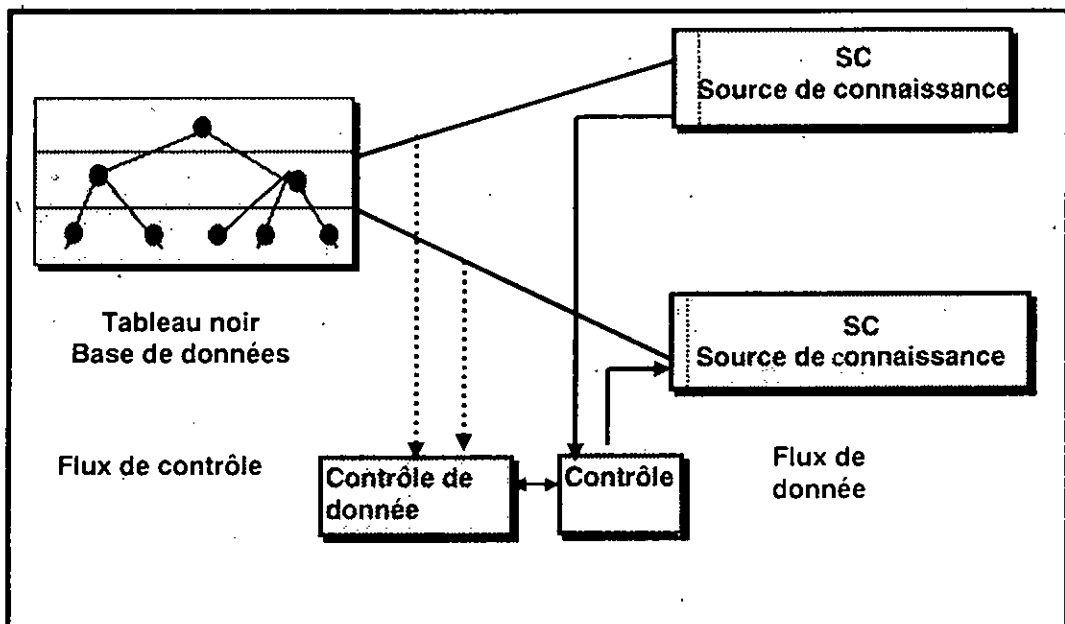


Figure 1.2 : Modèle de tableau noir

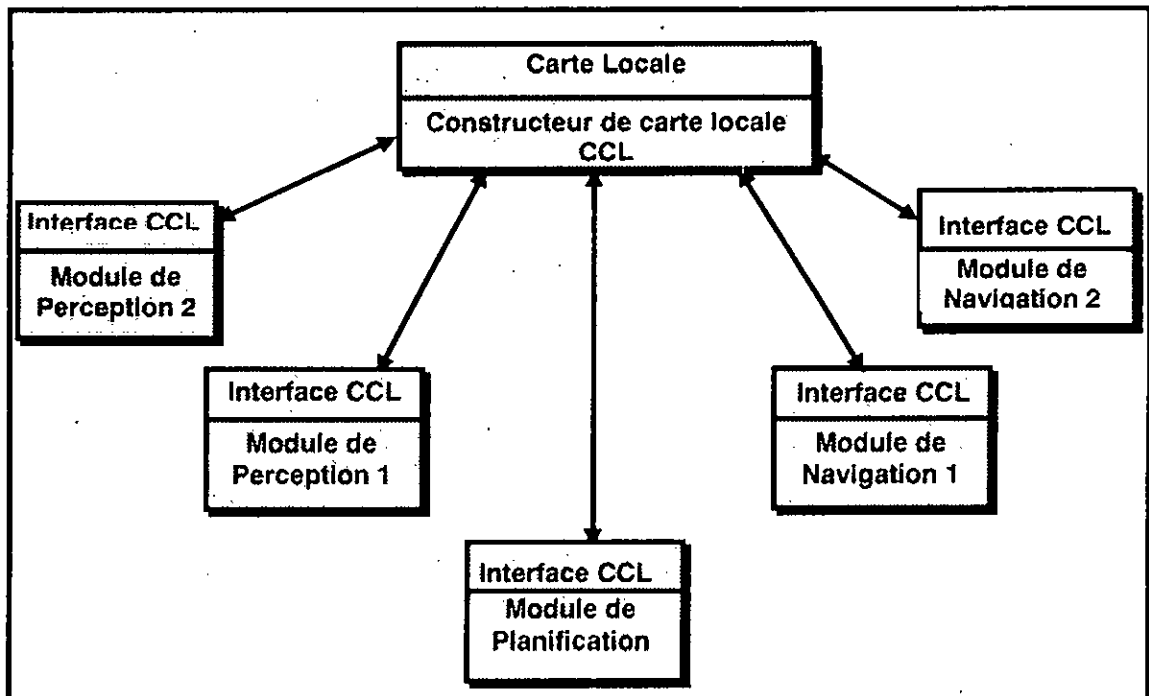


Figure 1.3 : Structure du robot NAVLAB



1.7. NAVIGATION DES ROBOTS MOBILES

Le robot mobile doit atteindre son objectif en sécurité malgré le manque d'information sur son environnement. Cependant la navigation autonome représente l'un des grands objectifs de la robotique mobile. La navigation est l'art de contrôler l'évolution d'un robot mobile au moment où il traverse un environnement (terre, mer ou air). Sans risques de collisions, la destination doit être atteinte à n'importe quel système de navigation, ceci implique la prise en charge de trois missions importantes à savoir: *le planificateur- la cartographie- le pilotage*. L'objectif de la navigation est de générer un chemin qui relie une configuration initiale à une configuration finale du robot en lui évitant toutes les collisions avec les imprévus de l'environnement. Dans ce domaine, plusieurs recherches ont été amenées pour modéliser l'environnement.

La modélisation de l'environnement est une façon de structurer l'espace d'évolution du robot. Celle-ci s'effectuera à partir des informations fournies par les moyens de perception (ultrasons, caméra, etc.). Un environnement peut être modélisé selon deux concepts:

- Le modèle de grille.
- Le modèle polygonale.

1.7.1. Modèle de grille

Une grille représente l'espace libre et l'espace occupé de l'environnement. Les cases de la grille résultent de l'intersection des lignes horizontales et des lignes verticales (cases carrées) et parfois diagonales (cases losanges).

Cependant, la case de cette grille est représentée par une certaine étiquette indiquant sa fonction « libre ou occupée ». L'avantage majeur c'est qu'elle est facile à mettre en œuvre mais l'inconvénient elle occupe beaucoup d'espace mémoire. Voici un exemple de modèle de grille à cellules carrées :

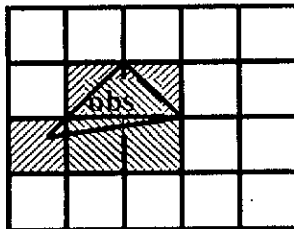


Figure 1.4 : modèle de grilles à cellules carrées

Le chemin de C_i à C_f ne peut être trouvé par le système dans la figure 1.5, cette dernière démontre le problème d'utilisation de ce modèle ; en effet, le chemin et le marquage successif sont difficiles à se retrouver lorsque une grille est à moitié pleine (voir la figure 1.5).

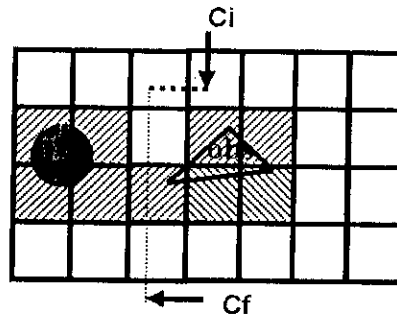


Figure 1.5: Exemple de décomposition occultant un chemin faisable

1.7.2. Modèle polygonale

L'espace libre du robot est divisé sous formes de polygones obtenus en reliant les sommets des obstacles aux autres sommets qui leur sont visibles, ensuite il y aura un graphe de connexité associé au modèle où les nœuds sont les cellules, et les arrêtes le lien entre les cellules adjacentes. Dans ce cas, un obstacle est alors représenté par les coordonnées de ces sommets dans le repère orthonomé. Voici un exemple :

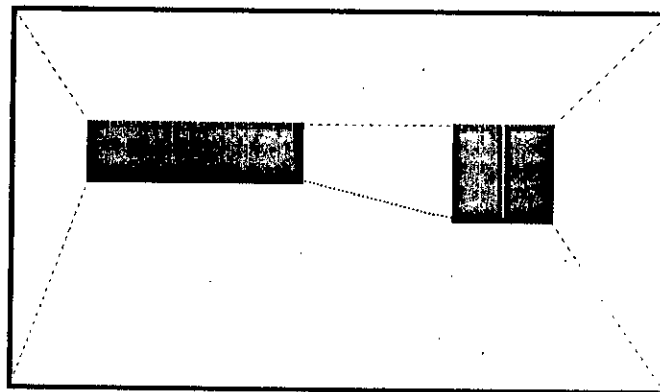


Figure 1.6 : Exemple de décomposition en polygones

1.8. PLANIFICATION DE LA TRAJECTOIRE

On définit la planification de trajectoire comme étant la possibilité de faire évoluer un robot d'une configuration initiale à une configuration finale, cela suivant un chemin bien approprié en évitant tous les risques de collision avec d'autres corps (obstacles susceptibles de se mouvoir sur son chemin) [9]. Cependant, on assiste à trois catégories de méthodes de générations de trajectoire à savoir :

- Planificateur global.
- Planificateur local.
- Planificateur mixte.

1.8.1. Planificateur global (approche statique)

Le planificateur global est destiné à un environnement parfaitement connu. Dans ce cadre de planification, plusieurs approches ont été développées pour

planifier une trajectoire d'un robot mobile autonome, nous pouvons citer à titre d'exemple la méthode des grilles homogènes proposée par monsieur *LEE*, le modèle polygonal proposé par monsieur *LOZANO-PEREZ*, la modélisation polygonale proposée par monsieur *BROOKS* et d'autres méthodes qui ont été développées dans le sens de trouver une trajectoire dans un environnement connu .

1.8.2. Planificateur local (*Approche dynamique*)

A l'opposé des méthodes globales que nous avons introduit précédemment, les techniques locales n'utilisent pas de modèle complet de l'espace libre. Elles sont *sans mémoire* , et ne prennent en compte à un instant donné que l'environnement proche du mobile pour modifier une trajectoire de consignes . Ce type de navigation explique qu'un dispositif de perception est indispensable car on considère que le robot évolue dans un environnement qui lui totalement inconnu. Elle est appelée approche locale du faite qu'une seule information partielle (locale) sur l'environnement peut engendrer des incréments de déplacement du robot mobile. Ainsi , les temps de réponses sont nettement améliorés , ce qui permet l'utilisation de telles approches dans des environnements dynamiques (inconnus par le robot à priori).

Ces méthodes de planification sont attrayantes par leur simplicité. Bien sûr, elles présentent le défaut de ne pas offrir la garantie d'arriver à l'objectif, le robot peut se trouver bloqué. Parmi les méthodes locales existantes, on choisit d'étudier l'approche la plus connue: *méthode des potentiels* développée par *OUSSAMA KHATIB*. Le principe de cette méthode consiste à introduire des forces s'exerçant à distance sur le robot, la cible exerce alors une force attractive et les obstacles exerçant une force répulsive (voir la figure 1.7).

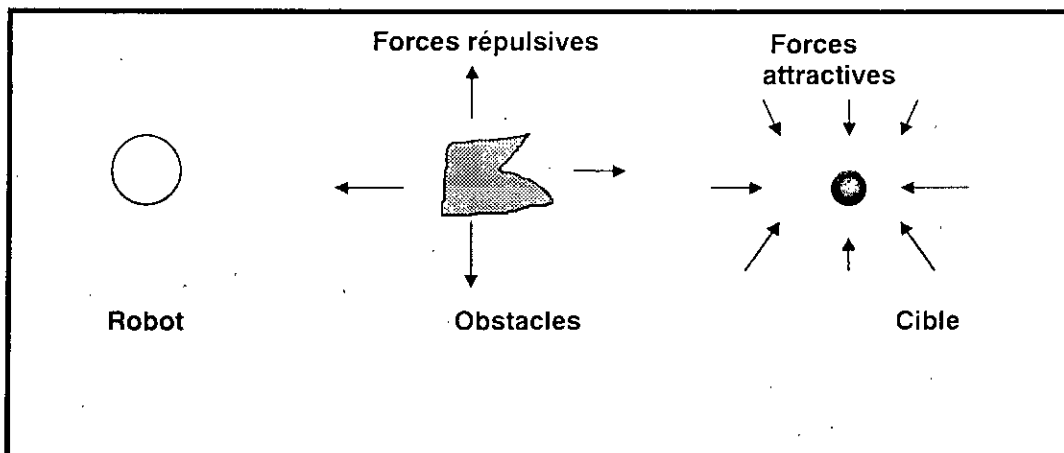


Figure 1.7 : Approche du champs de potentiel Artificiel

1.8.3 Planificateur mixte

Un couplage des méthodes locales avec les méthodes globales simplifiées, permet d'obtenir des solutions avec de bonnes garanties de terminaison. Ce type de planification revient à l'utilisation conjointe des méthodes globales et locales . Au niveau globale, on n'utilise pas de description géométrique des obstacles , mais seulement des poids dépendants de la probabilité de succès de la méthode locale à

déplacer le robot entre deux régions connexes.

Remarque

Les approches globales (statique : grille, polygonale) prennent un espace mémoire considérable, ainsi le temps d'exécution ne permet pas de concevoir un système intelligent pour la navigation. Par contre les approches locales (dynamiques) présentent certains avantages :

- Traitement en temps réel.
- Perspective de l'autonomie.
- Présentation des systèmes cognitifs et intelligents.

Les approches locales (classiques) ont été exploitées pour comprendre le fonctionnement du cerveau de l'être humain et de cela réputé intelligent un raisonnement logique pour que le robot peut se mouvoir, on assiste actuellement à l'introduction de la logique floue et les réseaux de neurones pour réaliser des systèmes anthropomorphes .

Dans ce qui suit, nous présentons notre conception de base pour développer notre approche, notre travail est consacré à la conception d'un planificateur local, grâce auquel on génère des trajectoires de références. Nous présentons d'abord la méthode de modélisation choisie pour notre environnement, ainsi que les étapes suivies pour parvenir à la construction du graphe de recherche. Ensuite, nous terminerons par l'analyse de l'environnement .

1.9. PROBLEMATIQUE DE NAVIGUER INTELLIGEMMENT DANS UN ENVIRONNEMENT INCONNU

Les problèmes que nous avons rencontré lors de la conception et qui ont été résolus pour répondre à une navigation donnée sont décrits dans la figure suivante :

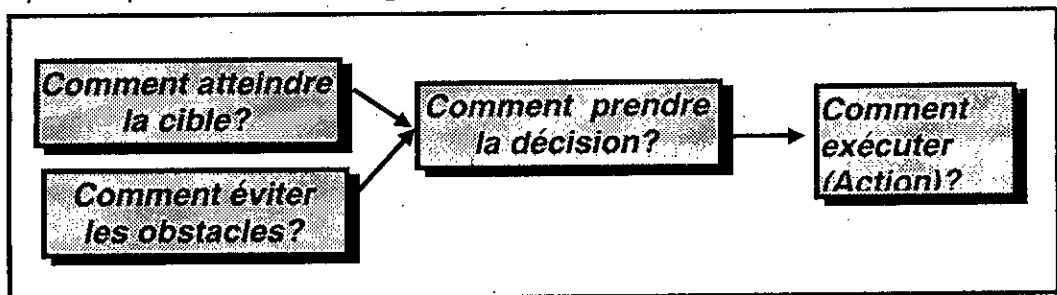


Figure 1.8 : Architecture globale de la problématique d'une navigation intelligente

Selon cette figure nous constatons que notre architecture peut répondre aux problèmes suivants:

- **Problème 1** : Ce problème consiste à donner un certain comportement intelligent, il faut donc trouver un moyen pour rejoindre la cible. Dans notre cas, nous allons élaborer tous les chemins possibles pour atteindre l'objectif en analysant la zone de sécurité du passage du robot (traiter pixel par pixel jusqu'à atteindre l'objectif).

- **Problème 2** : Les obstacles sont parsemés sans connaissance au préalable. Il faut donc trouver un moyen pour les détecter et les éviter par la suite. La richesse du langage de programmation (Borland C++) et ses clefs graphiques peuvent nous aider à résoudre ce problème. Une couleur différente par rapport à celle permise nous informe qu'il y a un problème (obstacle) donc il faut l'éviter par un contournement et une modification de la trajectoire. Il faut tester la valeur renvoyée (l'intensité lumineuse) de chaque pixel.
- **Problème 3** : Ce stade d'inquiétude pour le système mobile ne cesse de donner un certain raisonnement (calcul, etc.) pour prendre une décision logique et pertinente. Après les deux niveaux de tactique cités ci dessus, le problème de prise de décision exploite les résultats estimés avant de décider comment se diriger vers la cible. Ce comportement permet à notre robot de raisonner, à partir de ces résultats, et de décider de l'action appropriée lui permettant à la fois de se diriger vers sa cible et d'éviter les éventuels obstacles rencontrés.
- **Problème 4** : Le véhicule intelligent est capable de prendre cette décision par exécution de l'action décidée par l'étape précédent. Donc ça ne cesse de demander aux moteurs du véhicule d'exécuter ces actions.

1.10. CHOIX DE LA METHODE DE MODELISATION

Vue l'importance cruciale de la modélisation de l'environnement, il était nécessaire de choisir une bonne méthode de représentation qui tienne compte de toute les contraintes et critères imposés, et essentiellement de :

- La complexité de l'espace d'évolution du robot.
- La nature de la tâche que doit accomplir le robot.
- Les conditions liées du robot.

La modélisation de l'environnement est une façon de structurer l'espace d'évolution du robot, celle-ci s'effectuera grâce aux moyens de perception, dans notre cas, nous avons utilisé une caméra (en plus les capteurs ultrasoniques embarqués avec le robot). Ce moyen de perception peut nous fournir toutes les informations nécessaires (vue panoramique avec le moindre détail) sur l'environnement pour définir bien la cartographie de travail (qui est important pour nous du moment que l'environnement est parfaitement inconnu). En outre, nous tenons à préciser que les capteurs ultrasoniques limitent la cartographie par leur inconvénients. Une fois que la cartographie a été bien déterminée nous allons la décomposer en trois niveaux à savoir :

a. Niveau géométrique

Ce niveau correspond à la carte des lieux dans laquelle les objets sont représentés par leurs projections au sol. La surface du plan de la scène est assimilée à une surface plane idéale ou rien ne fait voir une hauteur ou un paramètre d'épaisseur d'un objet quelconque [10].

b. Niveau topologique

Ce stade constitue en la décomposition du modèle géométrique en zones libres et zones occupées, et le développement d'une structure qui définit la relation d'accessibilité entre les zones libres. Dans notre cas, du moment que l'environnement est parfaitement inconnu, il était nécessaire d'utiliser les techniques du graphisme du langage Borland C++ pour bien situer et étudier le problème, pour définir les circonstances lors de la navigation et les cas possibles qui peuvent avoir lieu.

c. Niveau sémantique

Il attribut à chaque élément (ou zone) du modèle topologique une étiquette ou un label représentant sa fonction. A ce stade de traitement, nous affectons à chaque case de la cartographie la probabilité suivante :

$$P: \begin{cases} 1: \text{Si la zone est libre.} \\ 0: \text{Sinon.} \end{cases}$$

1.10.1. Principe du modèle de grille

Pour réaliser notre projet, nous nous sommes inspirés du modèle de grilles pour représenter l'espace libre et l'espace occupé de l'environnement et qui était bien convenable dans notre cas. Les cases de la grille se résultent de l'intersection parallèle des lignes horizontales avec des lignes verticales qui peuvent nous offrir des cases carrés (grilles homogènes) avec la détermination des zones de sécurité et des zones dangereuses. Le niveau sémantique pourra bien apprécier son rôle ici après avoir affecté à chaque case un label, nous nous retrouvons avec des distributions de probabilités où la plus grande correspond bien à un état apprécié (libre ou occupé).

Ces grilles homogènes présentent un certain avantage majeur : c'est qu'elles peuvent trouver un marquage successif possible dans un environnement inconnu (chemin liant la source à la cible), mais le problème qui se pose lorsqu'une grille à moitié pleine (c'est à dire en réalité : elle est occupée par un obstacle) fait partie d'un chemin faisable exécutable comment peut-on remédier à ce problème?. Cette situation est la plus délicate dans l'étude et la conception, il a fallu bien étudier et tester le cas où une grille à moitié pleine fait partie d'un ensemble de sous-butts qui peuvent nous fournir un chemin faisable exécutable afin que le robot peut passer sans crainte ou risque de collisions. Pour cela nous avons déposé la condition suivante : si le robot peut passer en tenant compte de son encombrement, alors on peut prendre cette case comme étant une case accessible, sinon, elle sera inaccessible.

1.10.2. Planificateur local Appliqué à l'approche

Notre étude est basée sur la conception d'un planificateur local (notre environnement est parfaitement inconnu sauf les coordonnées de l'objectif et de la source), pour présenter une vraie tâche intelligente assistée par un ordinateur. Pour cela nous avons intégré un tel planificateur. Cette assistance intelligente est fondée sur les techniques d'Intelligence Artificielle IA qui est en osmose avec la robotique

mobile, d'après les relevés requises de notre travail, nous avons constaté que ces deux disciplines complémentaires et étroitement imbriquées concourent à doter des automates de facultés simulant celles de l'être humain et qui sont difficile à séparer nettement.

Le modèle choisi montre ainsi que l'IA joue un rôle croissant dans la robotique et qui constitue une composante majeure pour prendre une décision sur le chemin à choisir.

1.10.2.1. Approche développée

Notre architecture s'inspire du modèle possédant une fonction de planification *réactive*, ici le mécanisme de planification génère les trajectoires et contrôle leurs exécutions. Ces trajectoires peuvent être modifiées si le robot rencontre un obstacle imprévu ou un problème quelconque. Par les moyens de perception, le robot mobile autonome *mesure* des distances, des intervalles temporaires et effectue des calculs ainsi que des tests nécessaires. A cet effet, on effectue une génération de plan d'action un petit peu grossière (passer par le nord de l'obstacle, passer par ce coté gauche ,etc.).

1.10.2.2. Génération de plan

Pendant toute la durée de la mission, le robot devra "*contrôler l'exécution*" de ce plan étape par étape. Si le robot va rencontrer un obstacle inattendu, la modification locale de trajectoire est donc nécessaire pour le contourner, autrement dit, dans ce cas il va falloir "*remonter*" dans le raisonnement jusqu'au niveau de l'ordonnancement des tâches pour résoudre le problème. La difficulté clé est donc la détection du niveau auquel peuvent être réglées de telles situations: réflexe, conduite, tactique ou stratégique (voir la figure 1.9).

- **Réflexe** : cette technique correspond à diminuer la vitesse si un obstacle est détecté dans la zone de sécurité du robot (un virage ou lorsque le robot est près de la cible).
- **Conduite** : celle ci est résolue par une modification locale de la trajectoire, cela se traduit par un contournement d'obstacles lors d'un cas imprévu.
- **Tactique ou stratégique** : un nouveau calcul de chemin et de trajectoire, la trajectoire est changée, donc il est nécessaire de faire un autre calcul et un autre test possible pour une nouvelle mission .

Pour la planification et l'exécution des trajectoires nous avons élaboré :

- **Planificateur (planificateur global de trajectoires)** : nous avons élaboré tous les chemins possibles après un certain traitement pour atteindre l'objectif (nous avons élaboré quatre chemins possibles). Ce traitement se démarque par les sous- buts de passage du véhicule intelligent nécessaires afin de tracer et de planifier toutes les trajectoires possibles pour atteindre l'objectif , ensuite, il faut bien étudier le cas le plus favorable (celui-ci est le rôle du second planificateur).
- **Navigateur (planificateur local de trajectoires)** : celui ci utilise le plan d'action

du planificateur global en prévoyant une routine encore plus simple grâce à la combinaison hybride des algorithmes génétiques et la logique floue(que nous allons exposer dans le deuxième chapitre) en délivrant ainsi une trajectoire plus en accord et plus optimale.

- **Pilote** : Si on veut exécuter la trajectoire finale issue de les deux étages précédents, ça ne cesse de demander au pilote (moteurs d'actions) de naviguer sans crainte ou risque de collisions après avoir bien spécifier, traiter et planifier le chemin à poursuivre. Il se traduit donc par l'exécution de cette trajectoire finale obtenue après un certain temps de calculs mené par les deux planificateurs décrits ci dessus.

1.10.2.3. Architecture hiérarchisée

La planification de la trajectoire utilisée pour développer notre progiciel est décomposée comme suit :

Technique du tableau noir

La représentation du trajet du robot est illustrée sous forme d'un ruban, ce dernier est découpé en des segments qui sont décrits séquentiellement. Ces segments offrent un certain comportement important pour notre véhicule issu du principe de la technique de tableau noir. Le choix de cette technique est du son opportunité pertinente dans notre conception. En effet, le comportement adéquat offert par la technique du tableau noir " *Blackboard* " nous a été très intéressant d'offrir un comportement réactif intelligent à notre automate (voir la figure 1.9).

Lors du développement de notre architecture, nous avons remarqué un certain nombre de modules de travail constituant la base comme : percevoir (vision), tourner (modification locale), suivre (avancer tous droit suivant la nouvelle trajectoire), avancer (le nouveau calcul du trajectoire) et arrêter (l'arrivée au but) qui font appel à cette technique pour offrir un contrôle robuste. Ces modules constituent les sources de connaissances indépendantes de notre tableau noir, le rôle est donc comment définir chaque événement correspond à ces modules et comment relier entre eux pour construire la base de données qui donne à la fin l'allure du chemin final exécutable.

Ces modules peuvent échanger des informations en les transmettant vers la base de donnée centrale (celle ci mémorise tous les résultats possibles concernant un raisonnement bien approprié).

Pour cela, il était nécessaire donc de baptiser la navigation de notre robot sur un concept hiérarchique centralisé en utilisant la technique du tableau noir. Cette technique offre un comportement important dans une situation dramatique et peut rendre notre automate apte de naviguer sans crainte ou risque de collisions. Les résultats obtenus montrent que la technique du tableau noir est bien convenable pour donner une vraie autonomie .

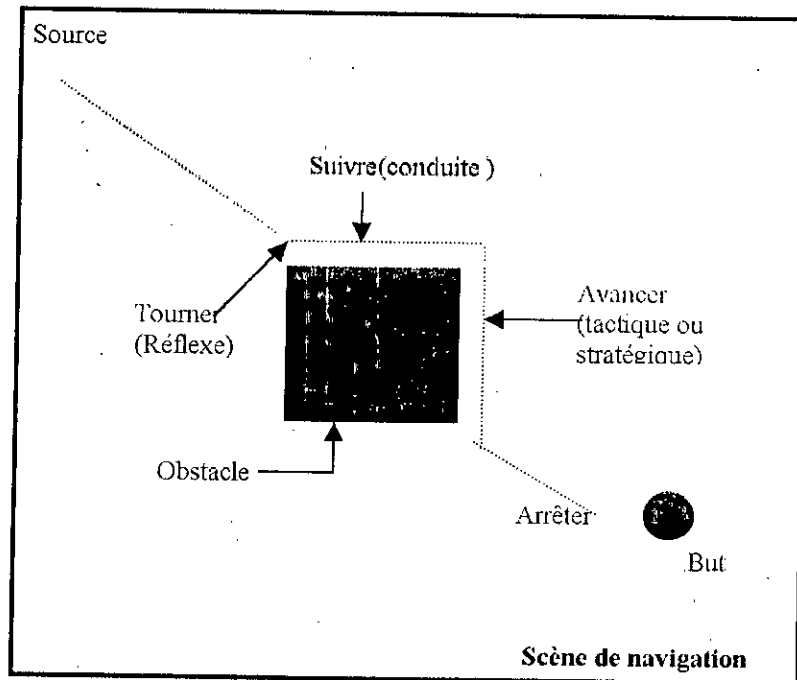


Figure 1.9 : un exemple de l'architecture développée

1.11. CONCLUSION

Le concept de la robotique mobile est en osmose avec l'IA et la perception (vision). Ces domaines complémentaires et étroitement imbriqués sont difficiles à séparer nettement. Ainsi que la composante de planification est donc indispensable dans la structure d'un robot mobile, à associer à la réactivité.

La modélisation de l'environnement est une façon de structurer l'espace d'évolution, une étape inhérente avant de tout traitement. Deux composantes majeures qui contribuent encore plus à la navigation : le taux de connaissance de l'environnement avant la mission, et le type de capteurs utilisés pour la détection des situations topologiques et imprévues.

Dédicaces

A mes très chère parents, l'univers d'affection et d'amour, qui ne cessent de me guetter par ses divers sacrifices aux pèlerins qui dévorent tout un carreau, qui m'ont toujours aidée durant toute ma vie et qui m'ont encouragée pendant mes années d'études, surtout en ces derniers moments. Pour leur bonté, leur compréhension et leur tendresse, qui m'ont éclairée dans l'ombre à la lumière de leur yeux qui cherchent sans cesse de me voir comme je suis aujourd'hui, je leur dédie ce mémoire.

À mes grands-parents qui m'ont toujours aimée et soutenue depuis mon enfance.

À tous les membres de ma famille, avec qui je suis heureuse, qui m'ont toujours ouverte les horizons de cette vie et qui m'ont aidée à tracer et parcourir le chemin. Je les remercie de tout mon cœur pour leur tendresse, leur aide, ainsi que leur haute bienveillance envers moi. Ils n'ont cessé de me chercher un avenir vivifiant, et avec qui j'ai passé les plus belles années de ma vie.

À mon frère Faouzi et ma sœur Souhila, avec qui j'ai partagé le destin, qui nous a préservé dans la vie de ce monde où nous avons reçu le désir de la perspective purifié par la volonté de Dieu.

Je dédie ce mémoire à mes cousines Wassila et Rosa, à qui je leur offre les perles de la pluie et je leur dit : il faut haïr le feu des anciens volcans, avoir la peau d'une bonne foie et creuser la terre jusqu'à près de la tombe et de ne l'arroser jamais avec les larmes pacifiques car ça coûte cher de parsemer les graines avec les souvenirs fleurrissants pour cueillir les plus beaux sourires derrière la gloire de la paie.

À mes meilleures amies Fouzia et sa fille Nazeel, Safia, Nassima Chahna et Dabbia, avec qui j'ai goûté la bonne et la mauvaise imitation de ce monde. Nous avons reçu ensemble les choix et les joies de l'aube et de la nuit, que nous avons semés dans le jardin du silence, et on les a offerts au maudit océans qui garde les otages de la vie morte.

À tous ceux qui ont versé dans le palais de l'indigent une portion des biens que le ciel les a départis aux flammes de leur splendeur, qui ont le vrai de culte pratique agitée en harmonie pour guetter la prédication pour devenir difficile, voire dangereux pour remédier aux clefs de diamants.

À tous ceux qui ont examiné la bonne, l'agréable et la parfaite de la cérémonie qui ne prosterneront point devant l'image taillée et qui ne serviront loin de sa figure inspirée.

À ceux qui portent les preuves de sagesse visibles promulguées pour non crainte salutaire rebellée contre la volonté propulsée.

Chapitre 2

PERFORMANCES DES ALGORITHMES GENETIQUES ET LA LOGIQUE FLOUE

A. ALGORITHMES GENETIQUES

A.1 INTRODUCTION

Les algorithmes génétiques AGs ont été développés par *JHON HOLLAND*, avec ses collègues et étudiants en 1975, à l'université de *MICHIGAN*. Leurs recherches avaient deux objectifs principaux :

1. Mettre en évidence et expliquer rigoureusement les processus d'adaptations de systèmes naturels.
2. Concevoir des systèmes artificiels (en l'occurrence des logiciels) qui possèdent les propriétés importantes des systèmes naturels.

Ce statut de recherche constitue le point focale de toute les conceptions dans ces dernières années et qui a été introduit comme une méthode robuste de recherche et d'optimisation pour divers problèmes.

Les AGs génèrent une séquence de population par l'usage d'un mécanisme de sélection, et utilise le *crossover* et la *mutation* comme des mécanismes de recherches. En revanche , les systèmes naturels sont robustes, efficaces et performants car ils s'adaptent à une large variété d'environnement . En reproduisant sous forme artificielle le principe naturel de l'algorithme de sélection de la meilleure adaptation, nous espérons atteindre la même polyvalence. Dans les faits, les AGs ont fait preuves de leur capacités dans de nombreuses études théoriques et expérimentales . Les différences séparant les AGs des techniques d'optimisation conventionnelles sont :

- La manipulation directe d'une population, et non d'un point unique.

- L'exploitation au moyen d'une population , et non d'un point unique.
- L'exploitation aveugle à partir uniquement des valeurs de la fonction.

La recherche sur les AGs a pour souci principal l'amélioration de la robustesse [11], l'équilibre entre la performance et le coût nécessaire à la survie dans des environnements nombreux et différents.

A.2 DEFINITION

On définit les AGs comme des algorithmes d'exploration fondés sur les mécanismes de la sélection naturelle et de la génétique de *GOLDBERG*. Ils utilisent à la fois les principes de la survie des structures les mieux adaptés, et les échanges d'information pseudo- aléatoires. A chaque génération, un nouvel ensemble de création artificielles (des chaînes de caractères) est créé en utilisant des parties des meilleurs éléments de la génération précédente , ainsi que les parties innovatrices à l'occasion. Pour un problème quelconque (en général un problème d'optimisation), les solutions vont être codées sous forme de chaînes de caractères appelés chromosomes , les caractères étant gènes. L'algorithme consiste en trois opérations : sélection , opération génétique et remplacement.

Le cycle typique d'un algorithme génétique est montré dans la figure 2.1. Une population comprend un ensemble de chromosomes qui se présentent comme des solutions candidats au problème [12]. Initialement, un ensemble d'individus appelé population initiale , est généré de façon aléatoire. Les valeurs d'adéquation de tout les chromosomes sont calculées par une fonction objective dans une forme décodée (phénotype). Chaque individu représente une solution potentielle. Une première phase de " *sélection* " sert à établir une liste de paires d'individus . Une phase dite de reproduction de la population est ensuite effectuée : elle consiste à appliquer des opérateurs dits génétiques sur les paires d'individus pour engendrer de nouveaux individus. La reproduction a pour but de faire évoluer la population en y propageant les caractéristiques des individus, les meilleurs étant, suivant la métaphore évolutionniste, les plus adaptés à leur environnement.

Une taille constante de la population étant requise, une phase de " *remplacement* " est effectuée. Elle consiste à remplacer les plus mauvais individus de la population courante par les meilleurs produits. La taille constante de la population induit un phénomène de compétition entre les individus. Le cycle génétique est réitéré sur la nouvelle population jusqu'à un certain critère d'arrêt.

A.3. LE CROISEMENT " CROSSOVER "

C'est un opérateur de recombinaison de deux chromosomes parents pour produire des chromosomes fils ayant les meilleures particularités des parents. Les utilisateurs des AGs considèrent que l'opérateur crossover est le facteur qui distingue des AGs à d'autres algorithmes d'optimisations [13]. Il existe plusieurs opérations de croisement à savoir : le croisement multi- point et le croisement uni point . Voici un exemple d'un crossover en un seul point proposé dans la figure 2.2. L'opération s'effectue en coupant une paire de chromosomes de la population en un point même choisi aléatoirement et produit deux nouveaux chromosomes en échangeant les parties coupées aux mêmes endroits dans les deux chromosomes. Le croisement est inspiré du procédé biologique par la

combinaison de gènes de deux parents, il travaille par couple d'individus en coupant en un ou plusieurs points.

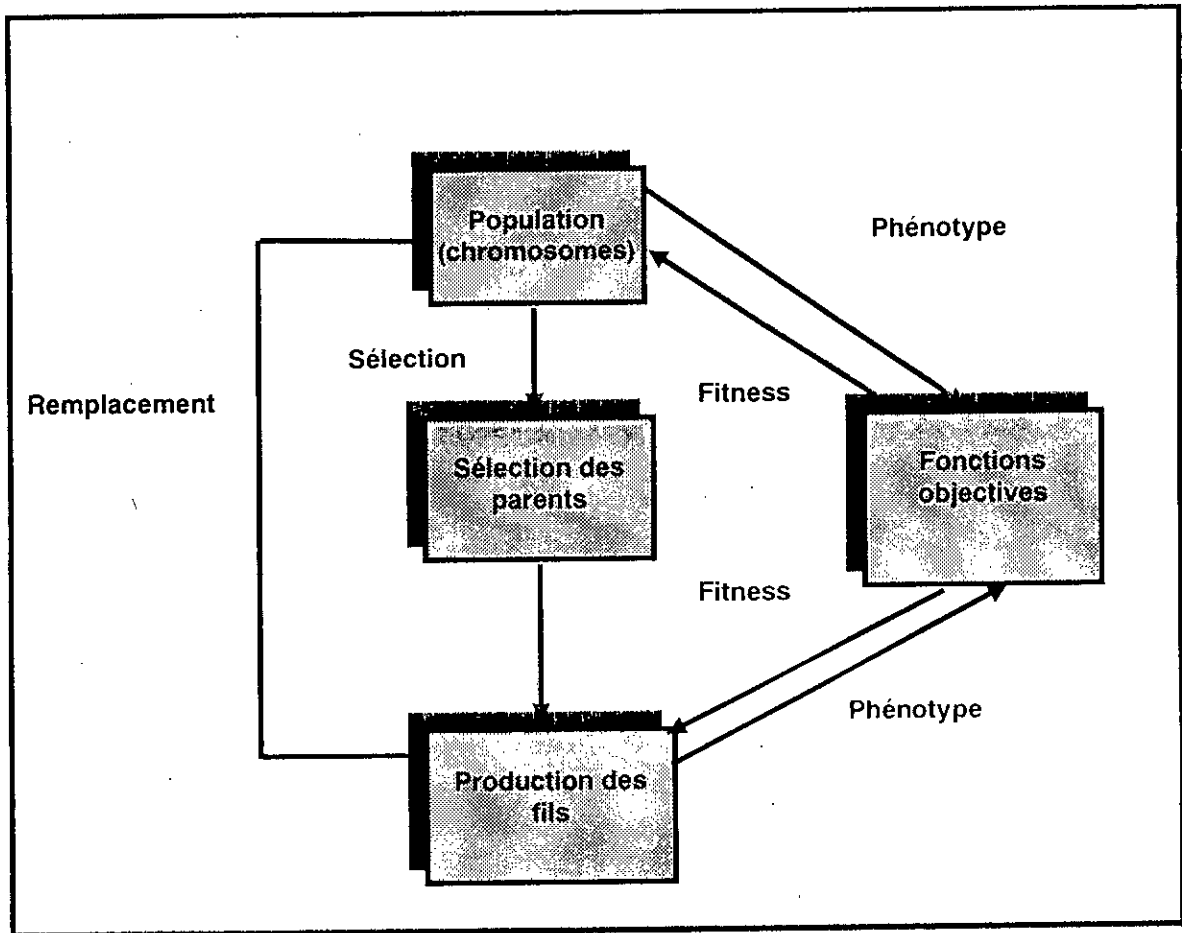


Figure 2.1 : Cycle d'un algorithme génétique

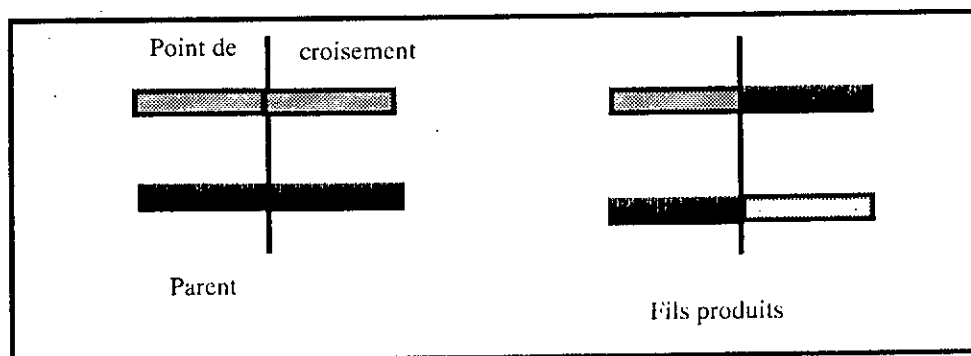


Figure 2.2 : exemple de crossover en un point

A.4. SELECTION

Le principe consiste de sélectionner au hasard des individus de la population sur la base de leur fonction d'adéquation [14].

Les principales difficultés d'emploi des AGs sont liées aux problèmes suivants

1. Trouver un moyen efficace pour modéliser le problème (au sens des AGs) d'une façon efficace et générique
2. Définir une fonction d'aptitude mesurant le meilleur coût à partir d'un individu.

A.5. MUTATION

La méthode de mutation des bits utilise une chaîne binaire aléatoire de taille égale au chromosome parent. A chaque position , les bits seront changés. Voici un exemple d'une mutation au troisième bit :

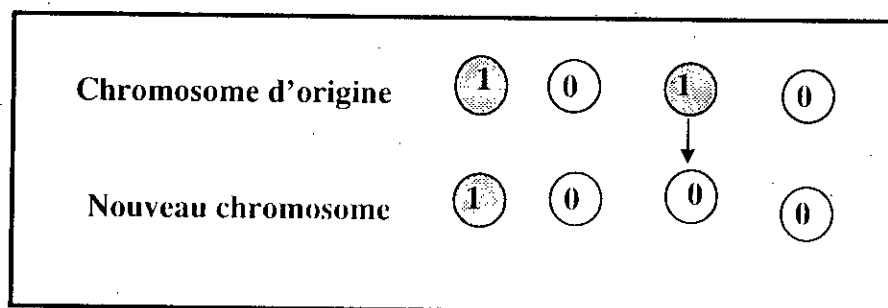


Figure 2 .3 : Exemple de mutation au troisième bit

A.6. FONCTION D'ADEQUATION " *FITNESS* "

Le rôle des fonctions objectives est celui d'optimisation, elles fournissent le mécanisme d'évaluation pour chaque caractère [14]. Sa rangée de valeurs varie selon les domaines des variables d'un problème posé. Cette fonction permet de donner une évaluation de la chaîne de population.

De cela on peut résumer que :

- La structure des AGs prévoit un outil pour optimiser la topologie ou la structure en parallèle avec les paramètres des solutions pour les problèmes particuliers.
- Le problème multi- objectif peut être adressé par des moyens d'approches génétiques.

A.7. ALGORITHMES GENETIQUES APPLIQUES AU PROBLEME DE LA NAVIGATION

Ce paragraphe expose comment les AGs ont été appliqués : ces algorithmes permettent de trouver la trajectoire la plus optimale (le paramètre est la distance) sous forme de points de passage sans collisions entre le point Source et le point But .

Modélisation du problème

On se propose à présent d'appliquer comme une alternative les AGs pour la recherche de la trajectoire la plus optimale. On montre que l'établissement d'une stratégie de renouvellement et de tri des coûts des chemins (la distance) permet de trouver le chemin le plus optimal pour effectuer la mission du robot. L'étape la plus délicate dans l'application des AGs est la modélisation du problème. Comme nous avons le graphe (les sous- passages) de l'environnement (ce graphe obtenu après la génération de plan et modélisation), la recherche du plus court chemin par les AGs utilise les informations déduites de celui-ci.

La population sera formée d'un ensemble de chemin, où chaque chemin est une suite de grille (sous-buts) de passage de robot. Ainsi que la population est prise comme un ensemble d'individus de taille connue (le nombre de sous points de passage sur le chemin). Ce dernier est représenté par une chaîne binaire de taille fixe, modélisant une suite de sous buts ou points du passage de robot dans l'environnement. En terme de biologie, les individus sont les chromosomes et les positions sont les gènes.

Le principe consiste à construire une population initiale de chemin de façon aléatoire, puis d'appliquer les opérateurs génétiques. Il s'agit de combiner des paires de chromosomes choisis aléatoirement, d'attribuer un coût à les trajectoires obtenues. Ce traitement se répétera jusqu'à obtenir un certain nombre " Pop Size " ou bien la taille de population d'individus. On relancera le processus de production de population autant de fois nécessaire pour obtenir les meilleurs individus (bien étendu le chemin optimal). L'arrêt est contrôlé par une fonction statistique vérifiant la variation des coûts des chemins, notre critère d'arrêt est appliqué lorsque la nouvelle population obtenue lors de la progéniture atteint le nombre $2^n - 4$ (n étant le nombre de chemin), on retranche le nombre 4 car on peut pas comparer et combiner un chemin avec lui même.

Fonction d'adaptation

La fonction d'adaptation sert à calculer le coût du chemin. Pour tout chromosome, on calcule la distance séparant le point courant et le point qui lui est visible, en évitant le cas de collision, ce point devient le point courant et on recommence le calcul, après sommation du trajet effectué jusqu'à atteindre la cible (c'est à dire jusqu'à ce que le point actuel soit égal à la cible).

Sélection et Opérateurs génétiques

On sélectionne au hasard des individus de la population caractérisés par leurs fonctions d'adaptation (la distance), on obtiendra par la suite une population P' sur laquelle on appliquera la reproduction avec croisement et mutation. Pour le crossover : on choisit au hasard deux individus (appelés parent 1 et parent 2), ensuite, on applique l'opérateur de combinaison et de comparaison, à chaque exploration, on compare le coût et on sauvegarde la valeur du coût minimal. Le processus s'effectuera 12 fois " critère d'arrêts " et on aura donc après plusieurs générations les meilleurs points de passage du robot. Cette opération sert à modifier plusieurs gènes de chromosomes après avoir choisir le meilleur coût. Le

chemin déduit de cette opération est le plus optimal pour la navigation et ceci bien sûr pour le gain en énergie et la capacité de puissance contenus dans le corps fermé (robot).

B. LOGIQUE FLOUE

B.1 INTRODUCTION

La logique floue dite "*fuzzy logic*" a été introduite en 1965 par le professeur *LOTFI ZADEH* à l'université de Californie à Berkeley, jusqu'à ces dernières années, la logique floue était restée le domaine réservée de quelques chercheurs curieux et peu d'applications pratiques n'avaient vu le jour. Elle permet de manipuler des symboles et d'inférer des actions en utilisant les règles logiques à partir de prémisses imprécises ou incertaines. Le terme logique est lié directement à la logique classique, logique binaire dont la caractéristique première est de connaître deux états bien définis vrais ou faux. Au contraire la logique floue est lancée pour la formalisation de modes de raisonnement précis est vu comme un cas limité du raisonnement approximatif. Dans nos jours la logique floue a déjà trouvé une application pratique dans des produits très divers.

Il existe dans l'industrie de très nombreux processus de fabrication "*optimisables*" en faisant appel à la logique floue. C'est dans ce domaine en particulier que peuvent se concrétiser les avantages très spécifiques de cette nouvelle logique, dans notre monde de l'électronique ce nouveau terme "*FLOUE*" ne tardera pas à remplacer les "*numériques*" dans différentes utilisations.

Le chemin le plus utilisé pour combiner la logique floue et d'autres disciplines est la fuzzification qui consiste à associer à chaque entrée du système plusieurs notes qui sont les degrés d'appartenances aux différents sous-ensembles flous pour chaque entrée. Dans les années quatre - vingt dix : quelques recherches spécifiées sont pour l'accord sur la fonction de degré d'appartenance et la reconnaissance des échantillons.

B.2. DEFINITION

La logique floue est considérée comme une extension de la théorie des ensembles. Etant donné un ensemble de référence X , on peut indiquer les éléments de X qui appartiennent à une certaine classe de X et ceux qui n'y appartiennent pas [15]. Si l'appartenance de certains éléments de X n'est pas absolue, on peut indiquer avec quel degrés chaque élément appartient à cette classe. Celle-ci est alors un sous-ensemble flou de X .

B.3. LES BASES DE LA LOGIQUE FLOUE

L'un des fondements les plus importants de la logique floue est la théorie des ensembles.

B.3.1. Théorie des ensembles flous

La logique floue travaille avec les ensembles flous, ces ensembles peuvent être définis mathématiquement par une fonction de degrés d'appartenance, qui définit pour tout élément le degré d'appartenance à cet ensemble. Ce degré varie de 0 à 1 [16].

Les notions d'inclusions, unions, intersections, complément et implication sont étendues aux ensembles flous [17].

B.3.2. Définitions

B.3.2.1. Définition d'un ensemble flou

Soit X un ensemble de points, avec un élément de X noté par x . L'ensemble flou A dans X est caractérisé par la fonction d'appartenance $FA(x)$, qui associe à chaque point de X le nombre réel limité par l'intervalle $[0,1]$, avec la valeur de la fonction $FA(x)$, représentant le degré d'appartenance de x dans A . Ainsi plus la valeur de $FA(x)$ s'approche de un plus le degré d'appartenance de x dans A est grand [18].

B.3.2.2. Définition d'un ensemble flou vide

Un ensemble flou est vide, si et seulement si sa fonction d'appartenance est identique à zéro sur x .

B.3.2.3. Définition de deux ensembles flous égaux

Deux ensembles flous A et B sont égaux, écrits comme $A = B$, si et seulement si $FA(x) = FB(x)$ pour tout élément x dans X .

B.3.2.4. Définition de l'intersection de deux ensembles flous

L'intersection de deux ensembles flous A et B avec des fonctions d'appartenance respectives $FA(x)$ et $FB(x)$ et l'ensemble flou C , écrits comme $C = A \cap B$, donc la fonction d'appartenance est allée à celle de A et B par :

$$FC(x) = \text{MIN} [FA(x), FB(x)], x \in X.$$

B.3.2.5. Définition de l'union de deux ensembles flous

L'union de deux ensembles flous A et B avec des fonctions d'appartenance respectives $FA(x)$ et $FB(x)$ et l'ensemble flou C , écrits comme $C = A \cup B$, dont la fonction d'appartenance est allée à celle de A et B par [19]:

$$FC(x) = \text{MAX} [FA(x), FB(x)], x \in X.$$

B.3.2.6. Définition du complément d'un ensemble flou

Le complément de l'ensemble flou A est noté Par A' est défini par $FA' = 1 - FA$.

B.3.2.7. Définition de d'implication de deux ensembles flous

L'opération est donnée par : si A est vrai, B est il aussi vrai ? $A \rightarrow B$

En logique binaire l'opération d'implication est que si A n'est pas vrai, aucune information n'est connue pour B , sur ce n'importe qu'elle valeur de B est acceptable. Cette relation est donnée par le tableau suivant :

| A | B | $A \Rightarrow B$ |
|---|---|-------------------|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |

Figure 2.4 : Table de vérité de l'implication

B.3.2.8. Définition d'un sous-ensemble flou

Un sous-ensemble classique A est défini par une fonction caractéristique G_A qui prend la valeur "0" pour les éléments de x n'appartient pas à A et la valeur "1" pour ceux qui appartiennent à A : $G_A(x) : X \rightarrow [0,1]$. Un sous ensemble flou A est défini par une fonction d'appartenance qui associe à chaque élément x de X , le degrés $FA(x)$, compris entre 0 et 1 , avec lequel x appartient à A : $FA(x) : X \rightarrow [0,1]$. En général les fonctions d'appartenance sont des fonctions mathématiques simples de forme trapézoïdales, et ont pour but de faciliter le calcul et la représentation .

B.4. L'INFERENCE FLOUE

Ce qu'on appelle inférence floue, c'est l'inférence d'action utilisant des règles logiques à partir des prémisses imprécises (floues) . Le type de règles utilisé est simple du genre : si condition 1, condition 2, etcondition n alors action [20].

B.5. DEFINITION D'UN CONTROLEUR FLOU

Un contrôleur flou est un système définissant une fonction (par exemple à deux entées X, Y et une sortie U) basé sur des règles exprimées par des catégories linguistiques telle que [21,22]:

Si (X est A1) et (Y est A2) alors (U est B)

ou` A1, A2, B sont des prédicats mentionnés comme " grand positif ", " moyen " , " petit négatif " etc.

Pour concevoir un contrôleur flou , il faut :

- Définir les variables d'entrée et les variables de sorties.
- Définir les ensemble flous correspondants.
- Définir les règles d'inférences.

En termes plus formels, un contrôleur flou (dont la structure générale est donnée dans la figure 2.5 [23]) se compose de trois modules opérant sur un système d'entrée et délivrant un système de sortie. Ces modules sont les suivants [22, 24, 25]:

B.5.1. Fuzzification :

Comme les valeurs d'entrée du contrôleur ne sont pas floues , il est nécessaire de les rendre floues, afin de pouvoir faire l'inférence floue et de faciliter le calcul [26]. La fuzzification consiste à calculer les degrés d'appartenance de chaque valeur d'entrée aux ensembles flous correspondants à la variable qu'elle présente. Ces ensembles sont utilisés mathématiquement par une fonction d'appartenance . La fuzzification a pour but de faciliter les calculs et représentation, les fonctions utilisées sont des fonctions triangulaires et trapézoïdales . La figure 2.6 représente les différents cas d'ensembles flous :

B.5.2. L'inférence floue

Cette technique est déjà expliquée ci dessus et qui représente une étape importante pour un contrôleur flou. En effet, c'est une application des règles d'inférence pour déterminer la grandeur de sortie. La logique floue a besoin d'une large variété de concepts et de techniques pour la représentation et la déduction à partir des connaissances . Celles ci se focalisent sur les règles d'inférences de types *If / THEN* [27,28,29,30,31]. Un grand nombre de relations exprimées sous formes d'implications floues peuvent être utilisées pour la représentation de la signification d'une règle de production tel que comme : *if (proposition 1) THEN (proposition 2)*.

B.5.3. Défuzzification

Elle consiste à calculer la valeur de sortie finale du système, grâce aux sorties floues et aux fonctions d'appartenance de sortie. Parmi les techniques de défuzzification les plus utilisées nous pouvons citer : la méthode du centre de gravité et la méthode de la moyenne pondérée.

À ceux qui sont encastrés dès les premières lueurs de l'aube, ceux qui résonnaient au bruit parvenait par les bribes de fréquents, ricocher entre les toits d'un cinéma, croyant avoir la cage d'un acteur et vraiment qui peuvent renverser sans scrupules le manipulateur d'un favori des scénarios.

À tous ce qui sont entrecroisés par les trous du crayon, sur la timidité tacite à pinaterie, inclinée par la cloison de peur pour s'écarter à des degrés de mesure d'un fil ombre doté à le trafiquer.

À ceux qui creusent le reste du temps derrière le flot des piétons, lorsque les trottoirs sont encombrés, ceux qui sont vraiment coincé au balayage d'un fétu de paille leur poussés soudaient à reprit sentencieux .

À ceux qui distribuent le fletce en temps décline emplissant les yeux au contre – coinant basculé souvent par les nuages de la librairie bandée, qui ne s'écraseront jamais aux regards sombres logés au près de la vague de la majesté.

À ceux qui mesurent le marché des connaissances essayer par les rayants nombreux prometteurs qui circulent dans les parois de l'expérience et qui nous heurtent par les activités utiles livrées par une certitude très claire à discipliner désormais de tous freins qui prenait la loi , ou plus que jamais le temps a son prit.

Qu'on ne peut guère réclamer d'autrui, et c'est bien regrettable...

Mais il viendra le pouvoir de penser aux sobriété un texte de réminiscence, orné son esprit de ces traits légers ou moraux qui agrémentent les conversations d'un art de bon aloi habité à l'abnégation de la modestie.

À ceux que je leur offre les termes de la bougie ou` ils jasaient le soir à petits bruits, quand la fenêtre rougit pour ouvrir les âmes d'un petit récit, ou` ils ne refusaient jamais les sons des surprises car faire fuir tous ceux qui ont quoi rendérissent les fripons qui ne finissent point.

À tous ceux qui m'ont aidée, aimée, conduite à la vie éternelle de prix égaré dans le voile assez... , d'une façon ou d'une autre, et qui par oubli ne figurent pas ici.

À ceux qui vont sûrement reconnaître cette piste arrosée par le manque de termes.

En voilà assez ! il faut s'arrêter... pour toucher le voile pétrissant et dire :

À tous ceux qui ont incinéré leur passé comme font les indiens à leurs morts et qui s'il leur reste des cendres les jettent à la mer.

Ouarda



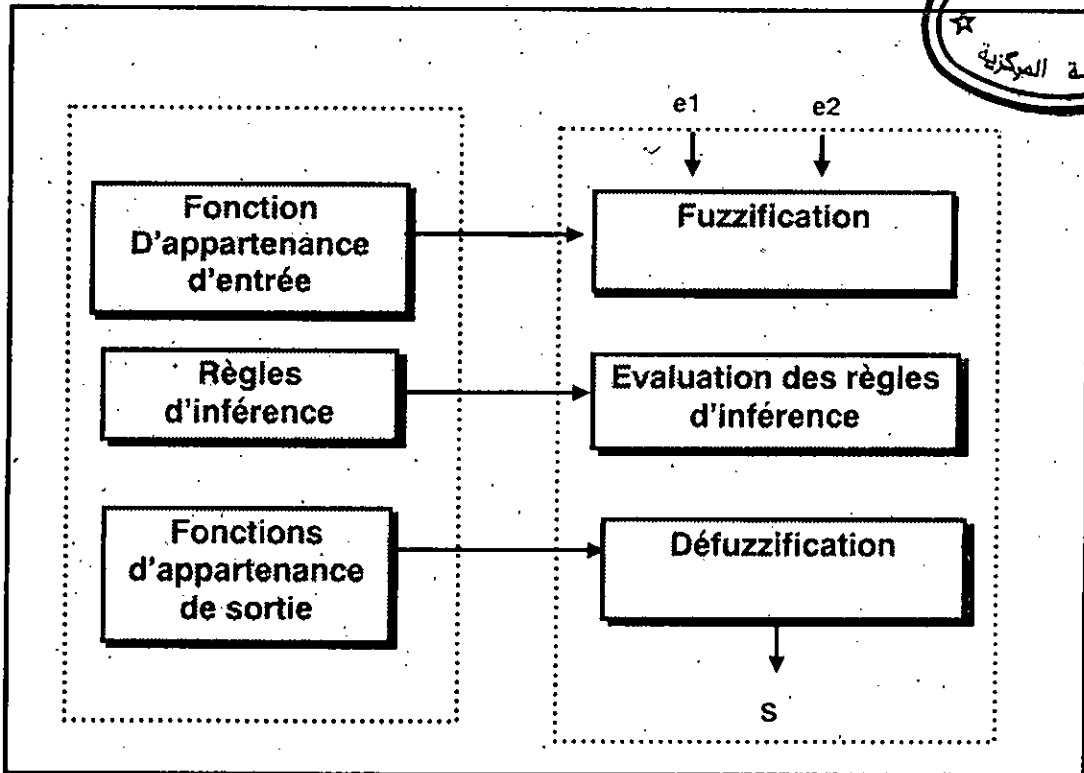


Figure 2.5 : Structure d'un contrôleur flou

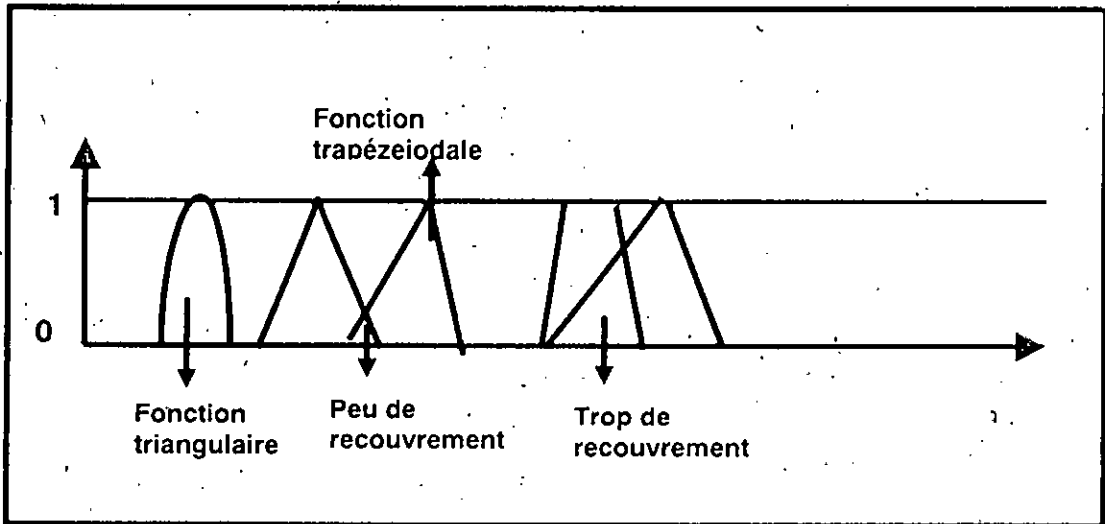


Figure 2.6 : Différents cas d'ensembles flous

B.6. CONTROLEUR FLOU APPLIQUE AU MOUVEMENT DU ROBOT

Pour une bonne exécution de la trajectoire de référence (issue de l'étage des algorithmes génétiques expliqué dans la première section A), nous pourrons encore plus optimiser cette référence en appliquant le contrôleur logique flou [58]. Ce dernier peut encore nous offrir une bonne assurance d'atteindre la cible en

évitant les obstacles et réaliser un bon lissage afin de gagner et réaliser les exigences d'autonomies.

Contrôleur flou appliqué pour avoir une bonne optimisation

En ayant la trajectoire de référence, le mouvement du robot peut être contrôlé par un contrôleur flou. Connaissant l'allure du chemin à suivre, le robot doit viser le sous but à atteindre à chaque pas d'avancement. Les directions à prendre dépendent alors des points intermédiaires de la trajectoire de référence. Le contrôleur utilisé est illustré dans la figure suivante :

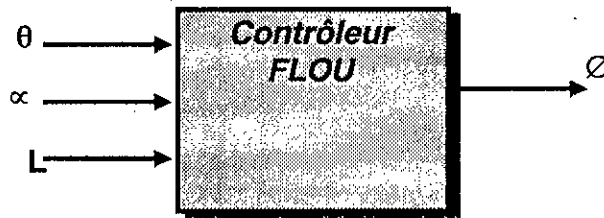


figure 2.7 :Le modèle de contrôleur flou utilisé

Selon cette figure le contrôleur a comme entrée

- θ : est l'orientation courante du robot (actuelle).
- α : l'orientation avec laquelle le robot s'oriente pour rejoindre le point visé.
- L : la distance intermédiaire entre le point actuel et le point visé, le point courant et le point intermédiaire et le point intermédiaire et le point intermédiaire et le point visé.

L'orientation déduite du contrôleur flou ϕ est l'orientation délivrée pour le robot, donc cette sortie du contrôleur représente l'orientation pour rejoindre le point visé en évitant les obstacles, le processus se répète jusqu'à ce que le point visé devienne la cible (voir la figure 2.8).

Le processus de traitement est décrit par les démarches suivantes :

- Calculer l'angle θ : celui-ci est déterminé par la pente du segment de la droite liant le point actuel (X_1, Y_1) et le point qui lui est intermédiaire (X_i, Y_i) donné par

$$\theta = \tan^{-1} \frac{(Y_i - Y_1)}{(X_i - X_1)}$$

- Calculer l'angle α : cet angle représente la différence de phase avec laquelle le robot rejoint le point visible et qui est illustré par :

$$\alpha = \tan^{-1} \frac{(Y_v - Y_1)}{(X_v - X_1)} - \theta$$

- Calcul de la distance selon les cas : entre le P_1 et P_i , P_i et P_v et P_1 et P_v (voir la figure 2.8).

Ce processus se répétera à chaque segment et à chaque pas d'avancement jusqu'à ce que le point visé devient la cible (l'objectif est atteint).

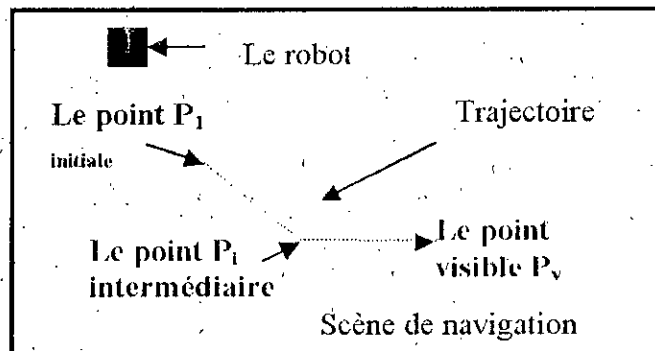


Figure 2.8 : exemple d'une scène de navigation

B. Variables linguistiques

B.1. Variable Linguistiques utilisées pour l'angle α et θ sont :

Grand gauche LB (*Left Big*).
 Petit gauche LS (*Left Small*).
 Petit droit RS (*Right Small*).
 Grand droit (*Right Big*).

B.2. Variable Linguistiques utilisées pour la distance L :

Proche N (*Near*).
 Moyen M (*Medium*).
 Loin F (*Far*).

B.3. Variable Linguistiques pour l'angle \varnothing

Grand danger gauche (LDB) : *Left Danger Big*.
 Petit danger gauche (LDS) : *Left Danger Small*.
 Petite sécurité gauche (LSS) : *Left Safety Small*.
 Grande sécurité gauche (LSB) : *Left Safety Big*.
 Grand danger droit (RDB) : *Right Danger Big*.
 Petit danger droit (RDS) : *Right Danger Small*.
 Petite sécurité droite (RSS) : *Right Safety Small*.
 Grande sécurité droite (RSB) : *Right Safety Big*.

C. Fonctions d'appartenances

Les fonction d'appartenances utilisées sont les suivantes :

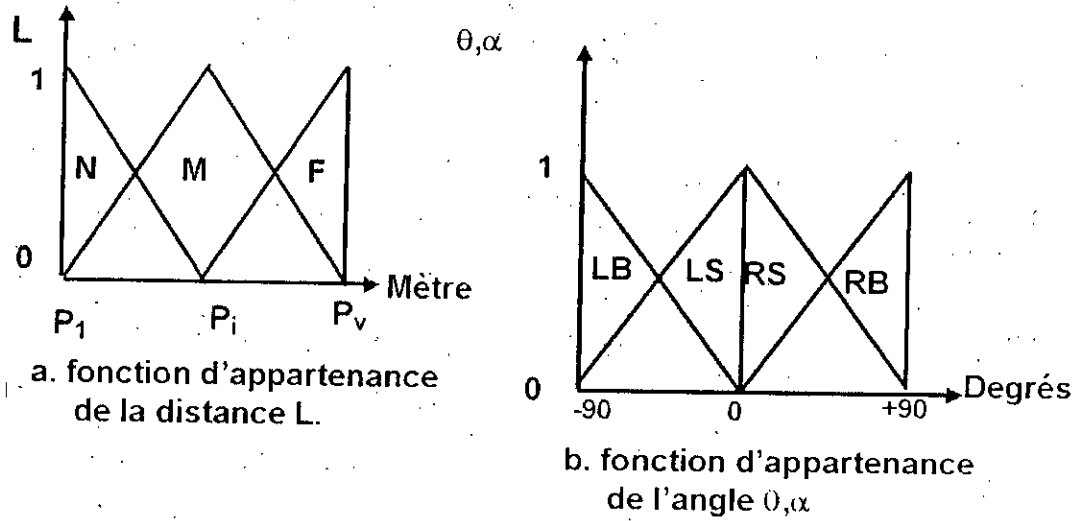


Figure 2.9 : Les fonction d'appartenances utilisées pour les variables d'entrées

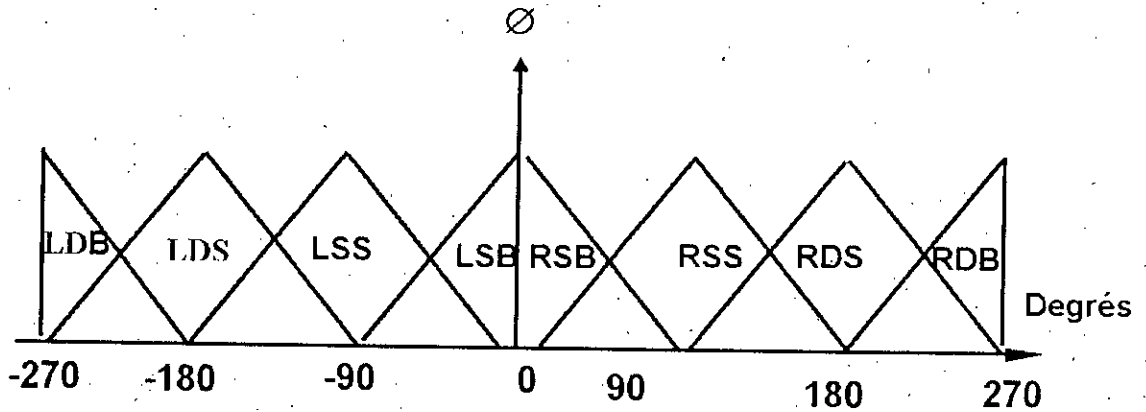


Figure 2.10 : Les fonctions d'appartenances de la direction ϕ

La défuzzification qui se traduit par la décision du système s'expose par la formule suivante

$$G = \frac{\sum (\mu_i * g_i)}{\sum \mu_i} \quad 1 \leq i \leq m.$$

Ou, m : le nombre de règles inférences .

g_i : le centre de gravité.

μ_i : degrés d'appartenance.

Les règles d'inférences sont définies comme suit



| | |
|---|---|
| Si (θ est LB et L est N) Alors (\emptyset est LDS) | Si (α est LB et L est N) Alors (\emptyset est LDS) |
| Si (θ est LB et L est M) Alors (\emptyset est LSS) | Si (α est LB et L est M) Alors (\emptyset est LSS) |
| Si (θ est LB et L est F) Alors (\emptyset est LSB) | Si (α est LB et L est F) Alors (\emptyset est LSB) |
| Si (θ est LS et L est N) Alors (\emptyset est IDB) | Si (α est LS et L est N) Alors (\emptyset est IDB) |
| Si (θ est LS et L est M) Alors (\emptyset est LDS) | Si (α est LS et L est M) Alors (\emptyset est LDS) |
| Si (θ est LS et L est F) Alors (\emptyset est LSS) | Si (α est LS et L est F) Alors (\emptyset est LSS) |
| Si (θ est RS et L est N) Alors (\emptyset est RDB) | Si (α est RS et L est N) Alors (\emptyset est RDB) |
| Si (θ est RS et L est M) Alors (\emptyset est RDS) | Si (α est RS et L est M) Alors (\emptyset est RDS) |
| Si (θ est RS et L est F) Alors (\emptyset est RSS) | Si (α est RS et L est F) Alors (\emptyset est RSS) |
| Si (θ est RB et L est N) Alors (\emptyset est RDS) | Si (α est RB et L est N) Alors (\emptyset est RDS) |
| Si (θ est RB et L est M) Alors (\emptyset est RSS) | Si (α est RB et L est M) Alors (\emptyset est RSS) |
| Si (θ est RB et L est F) Alors (\emptyset est RSS) | Si (α est RB et L est F) Alors (\emptyset est RSS) |

Figure 2.11 : Les règles d'inférences utilisées.

Remarque : Les vitesses du véhicule sont considérées constantes dédiées pour trois phases : accélération si le point visé est loin, vitesse est constante dans des distances intermédiaires et décélération dans la dernière portion de la trajectoire pour rejoindre le point visé (dans un autre cas ce point visé devient la cible) et ralentissement dans un virage (évitement d'obstacles), dans ce cas la vitesse du robot doit être suffisamment réduite pour qu'il ne dérape pas et ne sort pas de la trajectoire.

Notre objectif était en fait de jouer sur les directions du moment qu'on a une trajectoire de référence optimale et de contrôler et d'optimiser plus à partir de cette référence. Nous voulons réaliser une tâche qui répond aux critères de la navigation afin de gagner plus en temps, en coût, en consommation, etc.

2. CONCLUSION

L'idée ultime de ce chapitre consiste à exposer les performances de la logique floue et les algorithmes génétiques pour résoudre les points sensibles d'un problème d'optimisation et raisonnement non binaire, nous avons donc pris connaissance de la robustesse et comment manipuler les AGs et la LF.

Les systèmes hybrides intelligents basés sur les AGs et la LF offrent un comportement intelligent pour un robot mobile autonome.

Les AGs sont des alternatives inhérentes pour le problème d'optimisation de la trajectoire, en établissant une stratégie de renouvellement et de tri des coûts des chemins. Les AGs permettent en un temps acceptable d'obtenir la meilleure solution.

A côté de cette solution ultime, la logique floue est un outil très puissant qui peut faciliter la modélisation des problèmes basés sur le raisonnement humain, sans avoir recours à des constructions mathématiques complexes. Cet

outil nous a permis de trouver la meilleure orientation à donner pour notre robot pour accomplir la mission demandée. De plus, la logique floue rend le mouvement plus robuste et plus fiable. Ce mouvement lisse est adéquat pour exécuter une trajectoire faisable. Ceci n'exclut pas le fait de présenter quelques inconvénients, à savoir : la nécessité de bien définir le problème et surtout la nécessité d'évaluer et d'affiner les résultats.

Chapitre 3

LES ASICs et LES FPGA

A. CIRCUITS ASICs

A.1. INTRODUCTION

Le rôle des fabricants de semi-conducteurs pendant ces trois dernières décennies ne s'est pas limité à faire progresser la technologie et, ainsi, à augmenter le nombre de fonctions par circuit ; il s'est aussi attaché à définir des types de circuits à créer et à commercialiser pour répondre aux demandes des utilisateurs. La demande des utilisateurs a toujours été une constante : être capable de réaliser des fonctions de plus en plus complexes occupant le moins de place possible, c'est à dire coûtant le moins cher, utilisant le moins d'énergie et fonctionnant le plus rapidement possible[32,33,34]. Pour ce faire , les fabricants de semi-conducteurs ont proposé au fil des années des produits susceptibles de répondre à ces besoins.

Dans les années 1960 :

- Les circuits intégrés SSI (*Small Scale Integration*), les portes.
- Puis les MSI (*Medium Scale Integration*), les registres à décalages, les compteurs.

Dans les années 1970 :

- Les circuits LSI (*Large Scale Integration*), les microprocesseurs.

Dans les années 1980 :

- Les circuits intégrés VLSI (*Very Large Scale Integration*) : la révolution moderne de la micro-électronique.

Notons que cette évolution a amené, au milieu des années 1970, à résoudre la majeure partie des problèmes. Cette évolution s'est heurtée à plusieurs obstacles. D'une part la gamme des produits offerts aux utilisateurs n'est pas infinie, ceux-ci ne trouvent pas toujours les circuits adaptés à leurs applications, d'autre part, les fabricants de semi-conducteurs ont des difficultés à poursuivre l'innovation interne et, enfin vu le retard des outils de conception et de test rend plus difficile et plus coûteuse la conception de produit très complexes.

Ainsi les produits standards, les plus complexes soient-ils, ne peuvent pas tous répondre, dans les meilleures conditions, aux préoccupations de tous les utilisateurs.

La fabrication d'un circuit intégré nécessite un certain nombre d'étapes technologiques, afin de réaliser les différents composants du circuit. Cependant, elle est extrêmement coûteuse en temps de conception et en investissement de logiciel et, de plus, nécessite une bonne connaissance des règles technologiques du fondeur. Pour cela une voie s'est ouverte dans les années 1980 sous un nouveau concept spécifique : l'utilisation des *ASICs* (*Application Specific Integrated Circuits*) ou bien "circuits intégrés pour des applications spécifiques". Cette voie a pu voir le jour grâce surtout aux progrès des outils de CAO (*Conception Assistée par Ordinateur*). Ils sont, en effet, devenus suffisamment performants pour permettre aux concepteurs, de concevoir un circuit intégré rapide. Les *ASICs* pour les années 90 tiennent une place aussi importante que les microprocesseurs dans les années 70 et 80.

La conception de ces circuits nécessite une culture générale en électronique de base, des notions élémentaires de technologie et la maîtrise des outils de CAO, alors ; Qu'est-ce qu'un circuit ASIC ? Pour répondre à cette question, la section suivante peut éclairer la richesse du domaine.

A.2. DEFINITION

Par définition, les circuits *ASICs* regroupent tous les circuits dont la fonction peut être personnalisée d'une manière ou d'une autre en vue d'une application spécifique, par opposition aux produits standards courants de fonctionnement normaux [34].

Dans ce qui suit, nous allons passer sur les différents types de circuits *ASICs* qui peuvent exister.

A.3. STYLES DE CONCEPTION

Du concept ASIC découle quatre grands styles de conceptions (*Design Style*) à savoir :

- Les circuits sur mesure " *Full Custom* "
- Les circuits précaractérisés " *Standard Cells* "
- Les circuits prédiffusés " *Gate Array* "
- Les réseaux logiques programmables " *FPGA, PLD* "

Chaque famille diffère par rapport à l'autre dans : le niveau de complexité, un type d'outils de conception et une méthodologie de conception. Dans ce qui suit, nous allons exposer les quatre familles :

A.3.1. Circuits sur mesures “ *Full Custom* ”

Historiquement, ce sont les premiers ASICs qui ont été développés. Dans une conception full custom les dimensions et les performances d'un circuit sont optimisées. Toutes les étapes de fabrication sont spécifiques mais le temps de conception, ainsi que les coûts de développements sont très élevés par rapport aux circuits précaractérisés.

A.3.2. Circuits précaractérisés “ *Standard Cells* ”

Les circuits précaractérisés sont constitués de cellules élémentaires qui existent sous forme logiciel délivrées par le fondeur grâce au l'outil de CAO. Ces cellules prédéfinies qui sont indépendantes permettent d'offrir une approche plus flexible dans la mesure où elle permette une meilleure densité d'implantation sur le silicium. L'utilisateur est obligé de “ *faire avec* ” les cellules en bibliothèque, cela entraîne un cycle de fabrication long ainsi qu'un coût élevé.

A.3.3. Circuits prédiffusés “ *Gate Array* ”

Un circuit prédiffusé est constitué de cellules séparées par des réseaux d'interconnexions. Un certain ensemble d'éléments est implémenté par le fondeur dans l'attente d'une personnalisation pour réaliser les fonctions désirées par les utilisateurs. Cette personnalisation consiste à réaliser les interconnexions par des niveaux de métal. La conception d'un circuit prédiffusé nécessite un temps de conception élevé.

A.3.4. Réseaux logiques programmables “ *FPGAs, PLD* ”

Ce sont des circuits achetés sur catalogue et qui ne nécessitent pas un retour chez le fabricant. Cette approche offre une réalisation d'une matrice de cellules logiques séparées par des interconnexions déjà effectuées dans la structure interne. Ces composants en stock sont facilement programmables par l'utilisateur qui peut réaliser sa fonction désirée.

A.4. AVANTAGES ET INCONVENIENTS DES *ASICs*

A.4.1. Avantages des circuits *ASICs*

Presque tous les avantages des *ASICs* par rapport à l'utilisation des composants standards proviennent de la réduction de la taille des systèmes, Nous pouvons expliquer plus par : gain en place, extension des fonctionnalités,

augmentation de la vitesse de fonctionnement, réduction de la consommation et une amélioration de la productivité [34].

A.4.2. Inconvénients

Les inconvénients des ASICs résident dans l'effort à fournir pour les utiliser. Cet effort est de deux types : financier et technique. On peut expliquer par le coût important des outils de CAO et les frais de développement élevés du circuit.

A.5. CYCLE DE CONCEPTION D'UN ASIC

Dans ce paragraphe nous aborderons uniquement le cycle de développement économique d'un ASIC qui se décompose en deux étapes :

A.5.1. Faisabilité

L'étude de la faisabilité est fondamentale parce qu'elle conduit à déterminer la stratégie et les grands choix. Elle portera sur :

- **Etude marketing** : permet de préciser les volumes et les coûts de production.
- **Partition du système** : détermine ce qui peut être fait en ASIC de ce qui ne le peut pas.
- **Choix de la technologie** : détermine le type d'ASIC et sa technologie.
- **Etude des coûts** : appel d'offre aux différents fondeurs pour obtenir le prix de développement de l'ASIC et son prix unitaire en production.

A.5.2. Cahier des charges

Beaucoup de circuits ont dû être refaits parce que le cahier des charges avait été mal élaboré. Celui-ci devra définir le principe de fonctionnement par exemple. Il aboutira à la signature d'un contrat avec le fondeur, qui définit toutes les étapes du développement, les interfaces de travail entre ces sociétés, les procédures de validation de ces étapes intermédiaires et les conditions économiques et juridiques.

Remerciements

C'est avec infiniment de plaisir que je saisi l'occasion de la fin de ce travail pour présenter mes remerciements.

Je remercie tout d'abord mon promoteur et directeur de thèse, M. Karim ACIIOUR, pour m'avoir acceptée dans les laboratoires du CDTA (Robotique et Intelligence Artificielle, laboratoire de la micro-électronique et laboratoire architecture et réseaux des systèmes) et dirigée tout au long de cette étude.

Je le remercie aussi pour les moyens qu'il a mis à ma disposition et pour m'avoir permis d'accéder aux laboratoires même les week-ends et les jours de fêtes (et surtout les stations de travail disponibles pendant le long cursus de travail même au mois d'août et d'autres occasions) Je lui exprime aussi mes remerciements pour les contacts qui m'ont permis d'avoir des relations avec les laboratoires et aux centres de recherches scientifiques internationaux (micro-électronique, Robotique et Intelligence Artificielle, vision) et pour m'avoir mis à ma disposition la connexion sur Internet.

Je profite de l'occasion pour le remercier vivement pour la collaboration de ses collègues (Professeurs d'universités, Chargés de recherches et Chercheurs) qui se trouvent à l'étranger ou en Algérie par leurs opinions, les faits des recherches faites et réalisées sur les scènes nationales et internationales, et surtout les discours et les communications écrites et verbales pour la contribution à l'élaboration de ce présent travail, je le remercie infiniment pour les téléchargements des articles à travers l'Internet avec toutes les indications possibles présentes aujourd'hui, hier même pour le prochain cursus de recherche.

B. CIRCUITS FPGAs

B.1. INTRODUCTION

Les recherches sur les circuits ASICs s'étendent sur un domaine qui occupe une place aussi importante que les microprocesseurs dans les années 70 et 80. C'est ainsi que les circuits FPGAs (ASICs) dit circuits programmables constituent aujourd'hui une préoccupation dont la pertinence est incontestée par la communauté de chercheurs qui ont dévoués leurs efforts à réduire les contraintes et à élargir le royaume de la micro-électronique basée sur l'électronique moderne.

Le premier circuit programmable qu'on ait conçu est la PROM, on utilisait des lignes d'adresses comme entrées d'un circuit logique et la sortie de PROM comme sortie de la fonction logique. En effet, cette approche montre qu'une mémoire peut servir à codifier un ensemble de vecteurs logiques avec les entrées sur l'ensemble des bits d'adresse et en sortie les bits de donnée mais constituée d'une matrice de cellules configurées pour la lecture uniquement, en plus, programmable une seule fois.

A cet effet, Un lancement des projets de recherches – développement très ambitieux regroupant le plus souvent des industriels et des laboratoires de recherches s'est alors mis en œuvre. Ce plan qui vise à augmenter la productivité et améliorer la qualité de la production se termine par un nouveau hardware évoqué par le concept *MPGA* "Mask Programmable Gate Array", cette version essentielle montre que la majeure partie des problèmes est réglée, mais l'inconvénient réside de retourner chez le manufacturier, d'où un temps de fabrication long et un coût élevé lui en découlent malgré tous les avantages.

Dans l'industrie électronique, il est vital d'atteindre le marché avec de nouveaux produits en un temps le plus court que possible, cette contrainte du temps est devenue de plus en plus importante [35]. De plus, il est important que les risques financiers entraînés dans le développement d'un nouveau produit soient limités de telle sorte que plus d'idées nouvelles peuvent être prototypées [36].

Pour ce point de vu économique, cet *MPGA* a inspiré les inventeurs des FPGAs comme une solution ultime aux problèmes de risques financiers, car ils permettent une production instantanée et des prototypes à faible prix comparés aux *MPGAs*. Le profil de cette révolution technologique expose une compétence cruciale du fait que le circuit FPGA est un circuit qui peut remplacer plusieurs opérations coûteuses en temps et en coût. En outre, l'intérêt majeur est porté par le fait qu'il peut offrir une circuiterie aussi complexe qu'elle soit, celle ci peut être programmée en interconnectant des macrocellules logiques de façon modifiable et caractérisée par :

- Haute performance.
- Grande densité d'intégration.
- Flexibilité et reprogrammation possible.
- Régularité et reconfiguration remarquable.
- Grande rapidité et une vitesse pertinente.

Dans ce qui suit nous allons décrire cette nouvelle circuiterie « FPGA » marquée par le pouvoir économique sur le plan industriel de l'électronique [36].

B.2. DEFINITION

En passant par les chaînes commerciales citées ci-dessus, la question qui peut heurter le curieux micro-électronicien est la suivante: comment peut-on définir ce composant différent par rapport aux autres composants standards? Pour répondre à cette question, nous proposons la définition suivante.

Un FPGA est un composant électronique composé d'une matrice de blocs élémentaires dans lesquels une fonction logique peut être programmée. Ce composant dispose de ressource de routage (également programmables) permettant d'associer ces blocs élémentaires. De cela, un FPGA consiste alors en une matrice d'éléments singuliers (blocks logiques) qui peuvent être interconnectés de façon général, ces interconnexions entre les éléments sont programmées par l'utilisateur.

Un circuit logique peut être implémenté dans un FPGA en subdivisant la logique dans les blocs logiques individuels et en interconnectant ces blocs à travers les switches nécessaires. Les circuits FPGAs sont caractérisés par deux catégories de blocs logiques: ceux qui sont basés sur les tables de transfert « *Look-up-table* » ou bien « *LUT based* », et ceux qui sont basés sur les multiplexeurs « *multiplexor-based* », en d'autres caractérisations: « *MUX-based* ».

B.3. CIRCUITS FPGAs DE LA FAMILLE XILINX

En 1985, la compagnie Xilinx a fait rentrer les premiers FPGAs sur le marché, par la suite plusieurs autres compagnie telles comme Actel, Altera, Plus, Advanced Micros Devices (AMD), Quick logic et autres ont introduit de nouvelles architectures des FPGAs. Ces composants sont disponibles en quatre classes principales et commercialement disponibles à savoir: « *Symmetrical* », « *Row based array* », « *Sea of gates* » et les « *Hierarchical PLD* » (voir la figure 3.1).

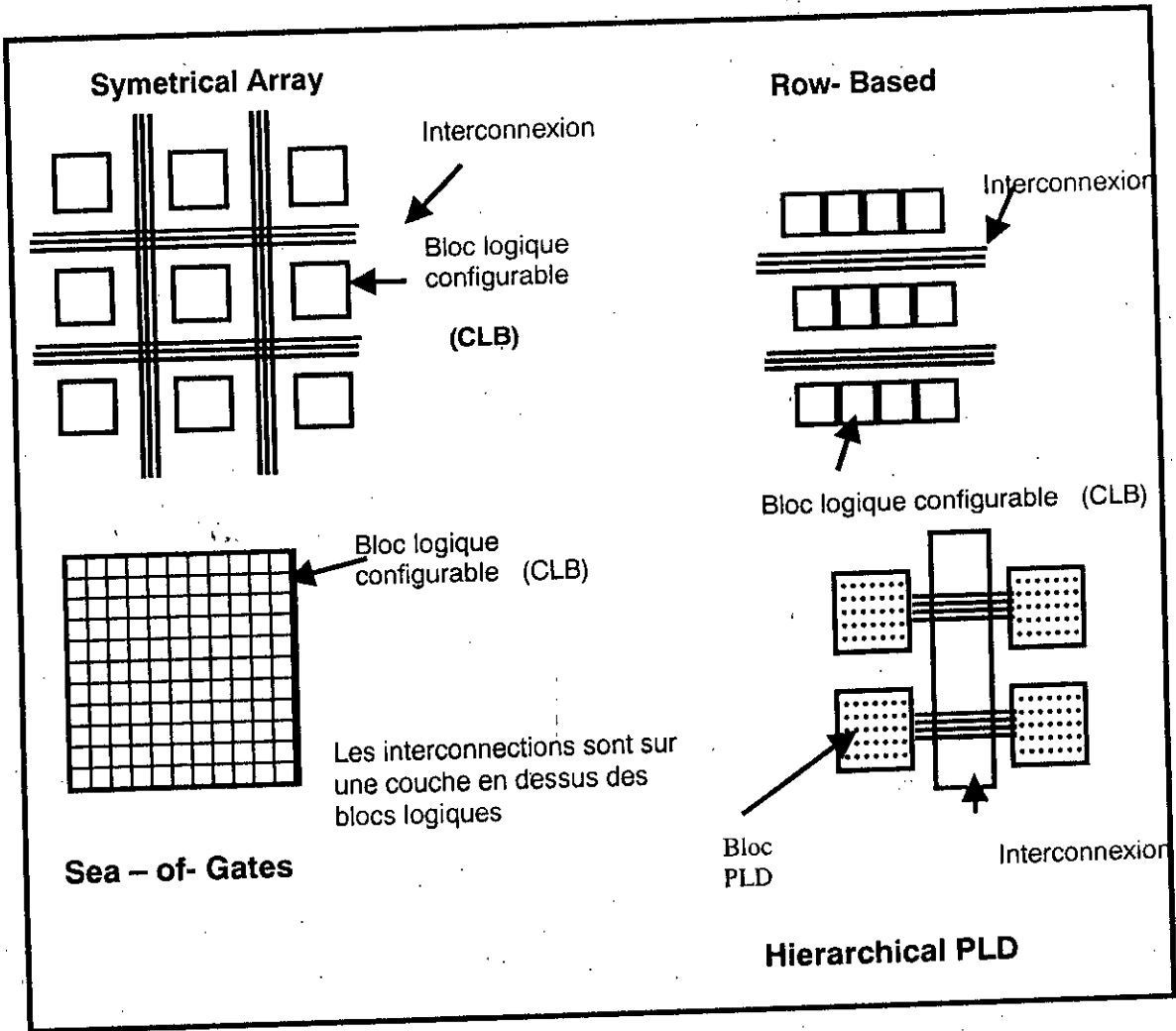


Figure 3.1: Les quatre classes des FPGAs commercialement disponibles

B.3.1. Famille de FPGA

Dans l'industrie électronique, il existe plusieurs structures et familles de circuits FPGA. Les chaînes commerciales de telles structures diffèrent d'une fondation à une autre par des techniques et des clés de la constitution. En particulier, si on se tourne vers la compagnie Xilinx, on se retrouve avec une large gamme de circuits **Reprogrammables** possédant des caractéristiques communes telles que la régularité, la flexibilité, la rapidité et une haute densité d'intégration. Il n'est pas évident de faire le choix entre les familles de FPGA avant de connaître et de spécifier le type de la logique utilisée dans l'application cible. Nous pouvons citer :

Famille XC3000 : Cet FPGA possède deux générateurs de fonctions, cinq entrées et deux sorties logiques par CLB, ce qui minimise ses options et restreint ses modes d'applications (voir la figure 3.2).

Famille XC4000 : Ce composant possède trois générateurs de fonctions (LUT F , LUT G, LUT H), des bascules FF et quatre sorties. Cette famille intègre un nombre important de portes. Il est très utilisé dans les applications DSP « Digital Signal

Processing » pour sa grande densité d'intégration et sa retenue logique rapide [37,38,39,40] (voir la figure 3.4).



Famille XC6000 : Ces circuits sont conçus pour opérer à proximité et en coopération avec les micros- contrôleurs et les processeurs. Ils sont utilisés comme interface ou comme fonction intégrée traditionnellement implémentée sur ASIC.

La compagnie Actel offre d'autres nouvelles architectures des circuits FPGAs, ces architectures sont caractérisées par une organisation de Mux- based, c'est à dire que le bloc de base est une configuration de multiplexeurs avec quelques portes logiques AND et OR. La figure 3.3 montre l'architecture de base d'un bloc logique de cette nouvelle catégorie de FPGA présentée par la compagnie Actel.

B.3.2. Ressources des interconnexions

Un composant électronique de type FPGA dispose des ressources de routages qui doivent être programmées pour réaliser la connexion désirée. Il existe deux types d'interconnexions : les interconnexions **Reprogrammables** et les interconnexions programmables une seule fois « **One Time Programmable OTP** ». Un élément programmable peut être implémenté de plusieurs façons par une certaine technologie de programmation. Au bout de la chaîne commerciale, on trouve les technologies de programmation suivantes qui sont les plus utilisées dans les produits : les cellules statiques RAM (SRAM) et les « **antifuses** » [41,42,43].

Particulièrement , la technologie SRAM est offerte par la compagnie Xilinx qui offre des circuits FPGA reprogrammables. Cette circuiterie montre que la connexion entre les différents blocs logiques est assurée par un programme enregistré dans une mémoire interne du FPGA, celui ci est chargé automatiquement lors de chaque mise en route. En effet, une petite modification du programme peut changer la fonctionnement du circuit même en cours de marche du système, cette flexibilité est d'une opportunité importante pour certaines applications, programmer c'est prévoir !

La compagnie Actel offre une technologie **antifuse**, les circuits produits par cette fondation sont des FPGAs **OTP** qui s'opposent par une différence immense à la technologie de la compagnie Xilinx.

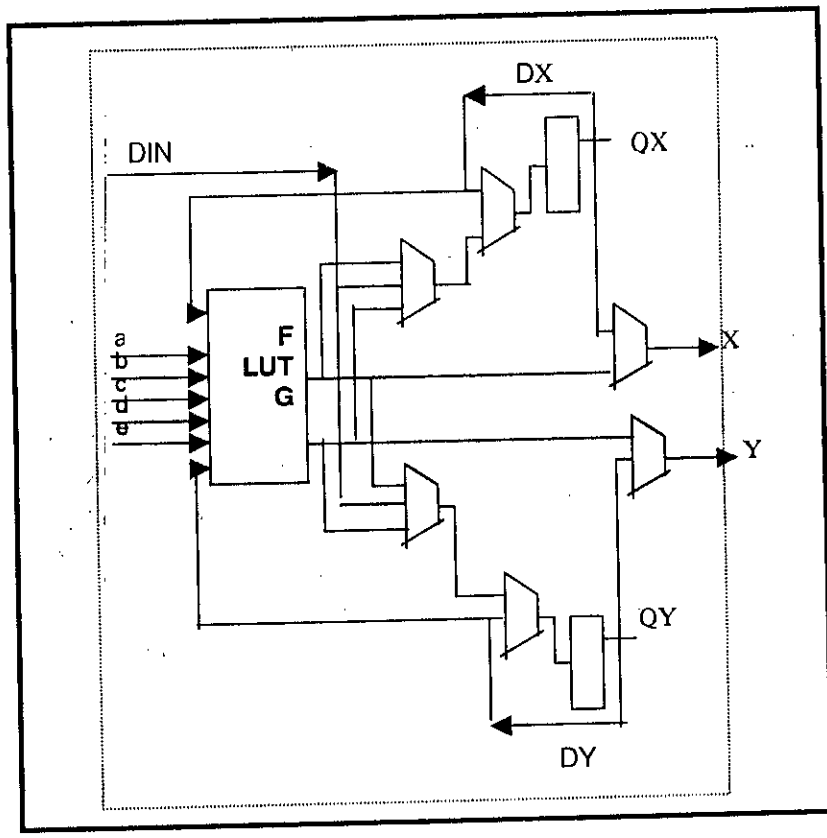


Figure 3.2 : Architecture de bloc logique configurable de la compagnie Xilinx – famille XC3000.

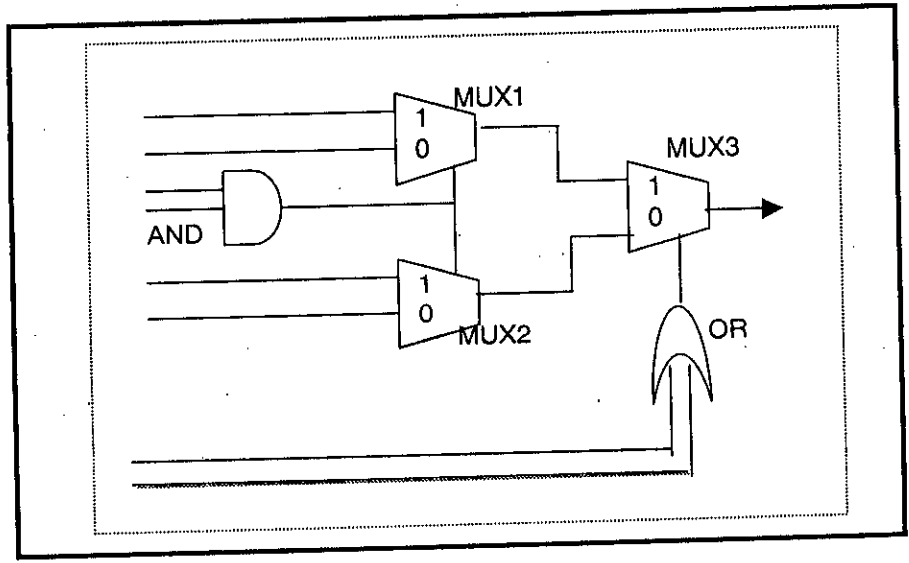


Figure 3.3 : Architecture de bloc logique configurable de la compagnie Actel

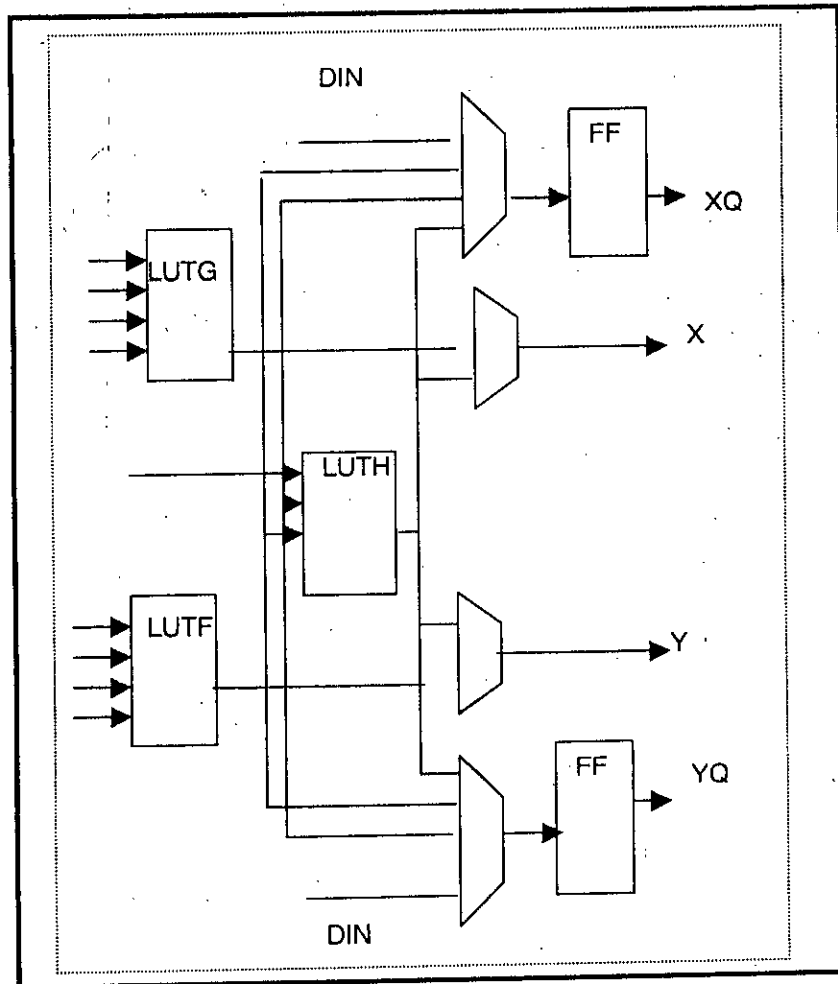


Figure 3.4 : Architecture de bloc logique configurable de la compagnie Xilinx Famille XC4000

B.4. LANGAGE DE DESCRIPTION HARDWARE " VHSIC "

Concevoir à un haut niveau d'abstraction répond au problème de la complexité croissante des systèmes électroniques et à la nécessité d'être les premiers sur le marché. Les langages de description hardware (HDLs) ont joué un rôle important dans la méthode de conception et de description avec la synthèse. Ils sont utilisés pour la spécification, la simulation et la synthèse d'un système électronique [44,45,46,47,48].

Le VHDL est l'un des plus réponsus de nos jours. La méthode de conception VHDL n'est devenue populaire et ne fut acceptée par les dessinateurs hardware qu'à partir de ces dernières années. Cette technique de conception s'expose par beaucoup d'avantages : elle améliore la productivité, elle permet au concepteur de créer un design en un temps réduit et elle peut être aussi utilisée par des personnes qui n'ont pas une grande expérience dans le dessin.

B.4.1. DEFINITION

Le VHDL est un langage de description de matériel électronique. Ce terme est l'acronyme du mot anglais (VHSIC HDL : *Very High Speed Integrated Circuits Hardware Description Language*). Ce langage de programmation est conçu en vue de la modélisation et de la simulation de circuit ASIC. Le VHDL traite des circuits intégrés à très grande spécification, la simulation et la synthèse d'un système électronique. VHDL est l'un des plus répandus de nos jours.

B.4.2. NECESSITE DU VHDL

La nécessité d'une description non ambiguë des systèmes matériels s'imposa au département de la défense des Etats – Unis (le DOD) au début des années 80. En effet, un phénomène nouveau apparaît alors : la durée opérationnelle des équipements de ce ministère oblige ses systèmes à traverser des évolutions technologiques de plus en plus rapprochées. La problématique s'est commencée dès le départ, la conception de circuits intégrés utilise depuis longtemps la simulation : aucun ne peut se permettre de fabriquer une puce, de tester son fonctionnement, et de détecter des erreurs et de les corriger avant la fabrication[44].

La micro- électronique est un des domaines où les erreurs coûtent le plus cher. Ces simulations aux niveaux logiques (les portes) sont hélas encore indispensables pour concevoir un circuit " *bon au premier coup* " et donc disponible rapidement sur le marché. Bien que ne permettant pas de détecter exhaustivement toutes les erreurs, la simulation reste néanmoins le seul moyen pratique d'en éliminer un bon nombre. C'est donc tout naturellement que l'on a commencé à utiliser des simulateurs fonctionnels (VHDL n'est pas le premier) en micro-électronique.

Q' est ce qui distingue VHDL des autres langages issus de divers laboratoires ? la normalisation de VHDL permet de travailler avec un langage connu de tous. Cette même norme autorise la création d'outils (de simulation, de vérification, etc.) de différents horizons et garantit ainsi une bonne qualité et l'indépendance vis à vis des constructions de ces outils.

Où s'arrête VHDL dans le domaine de la modélisation de systèmes ? Personne ne le sait bien car ce langage permet la manipulation de fils, de bus, de messages ou de protocoles reliant des boîtes. Certains se servent même de ce langage pour décrire et simuler d'autres systèmes.

B.4.3. Organisation pour une standardisation

L'effort de standardisation s'échelonna de 1983 à 1987 sous l'égide du DOD. De nombreuses sociétés rassemblées autour de *Intermetrics*, *IBM* et *TEXAS Instruments* fournissent un travail important qui aboutit à la norme IEEE, approuvée le 10 Décembre 1987. Depuis, VHDL est maintenu par le groupe américain VASG (*VHDL Analysis and Standardisation Group*). La norme définitive avait été révisé en 1992.

Généralement, une description d'un système en VHDL est constituée :

1. D'une ou plusieurs bibliothèques dans laquelle sont stockées des unités de conception.
2. Plusieurs catégories différentes peuvent configurer ces unités de conception qui sont les vues externe des entités (*entity* et *architecture*), à titre d'exemple la figure 3.B.5 montre les deux vues internes différentes du même modèle.

Parmi les avantages du VHDL nous insistons sur les aspects suivants :

- Le langage offre une organisation hiérarchique : un système peut être modélisé par un ensemble de composants . Ceux ci peuvent à leurs tours être modélisés par un ensemble de sous composant.
- Le VHDL permet des descriptions géométriques ou paramétriques.
- Par rapport à d'autres langages , VHDL se distingue par sa généralité (ce n'est pas un langage spécialisé)
- Le VHDL offre une simulation exhaustive de tous les éléments de base de la circuiterie complexe.
- L'équipement peut intégrer, sur une même - plate forme VHDL, les diverses descriptions de composants ou de fonctions.

Pour plus de détail sur le VHDL, nous conseillons le lecteur de consulter les références [44,45,46,47,48]

B.5. Synthèse

B.5.1. définition

On définit la synthèse comme étant une translation d'un dessin d'un niveau d'abstraction à un autre plus bas. Le processus de translation qui permet de translater une description comportementale en une description structurelle ou physique est similaire à la compilation d'un programme en langage évolué comme le C.

B.5.2. Les outils de synthèses

La synthèse se fait par des processus de différent types. Ces derniers permettent de concevoir un circuit intégré de la spécification comportementale à celle de la spécification structurelle ou plus réelle résultante de la conception.

Pour la représentation structurelle, le mapping d'une représentation comportementale se réalise sur un ensemble de composants et de connexions sur des composants tel que le coût, la surface et le délai.

Lorsqu'on veut réaliser un mapping d'une structure sur du silicium, il s'agit dans ce cas d'une représentation physique, ce qui fait définir l'opportunité de la synthèse dans toute conception comportementale théorique.

B.5.3. Synthèse avec le VHDL

La sémantique de simulation est l'une des richesses de VHDL. Pour le domaine de la synthèse, son utilisation est réduite à un sous-ensemble qui correspond à des représentations digitales identifiables. Les résultats d'une synthèse dépendent du style VHDL utilisé. A l'heure actuelle, les algorithmes de synthèse avec le VHDL les plus utilisés permettent d'optimiser les circuits décrits au niveau comportemental et de cibler la technologie correspondante.

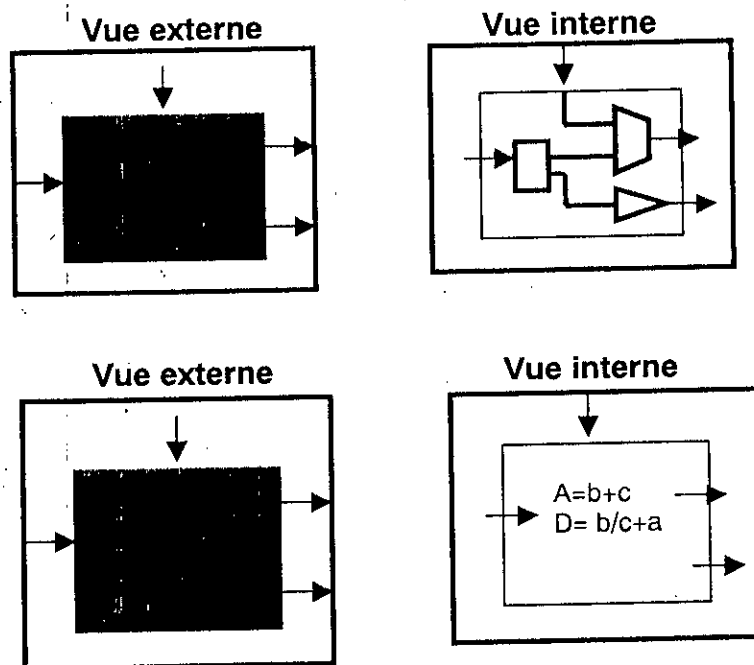


Figure 3.5: Deux vues internes différentes du même modèle.

Dans ce qui suit, nous présentons comment notre architecture a été conçue.

B.6. APPROCHE DE CONCEPTION

L'approche de conception est l'approche basée sur la synthèse avec le VHDL. La méthodologie de conception est illustrée dans la figure 3.6. Une architecture en VHDL est alors développée pour doter le comportement Hardware de l'architecture, cette phase est suivie par le mapping de cette architecture en une description VHDL. Comme outil de synthèse, nous avons utilisé le Galileo [49], ce dernier réalise automatiquement le mapping du VHDL en technologie FPGA.

L'automatisation se fait en tenant compte les contraintes de vitesse ou surface. Une netlist en format Xnf sera alors produite indiquant ainsi le résultat de la synthèse. Par la suite, le résultat est utilisé à l'entrée de l'outil XACT [50] de la

compagnie Xilinx qui va réaliser le placement et le routage de l'architecture. A chaque phase, une vérification de la fonctionnalité du circuit est nécessaire.

Dans ce qui suit, nous allons toucher les progrès exponentiels de la VLSI, le fonctionnement et la circuiterie au niveau plus haut peuvent être ciblés par le concepteur à fin de simuler la fonction désirée de son circuit. Nous allons décrire en détail chaque phase de l'architecture.

B.7. ARCHITECTURE DEVELOPPEE

Lors de la conception, il a fallu tenir compte de certaines contraintes qui sont

- Le parallélisme qui permet le traitement en temps réel.
- Les performances relativement à la complexité de l'architecture.
- La flexibilité ou l'adaptabilité du circuit.
- La surface du silicium.

Pour atteindre ces objectifs, une bonne implémentation VLSI doit avoir les propriétés architecturales suivantes [51] :

- simplicité de conception caractérisée par une architecture robuste.
- Régularité de la structure permettant de réduire les interconnexions.
- Possibilité d'extension et de réduction de l'architecture.

Dans notre cas, lors de la conception un grand nombre d'éléments ont été utilisés, notre architecture peut implémenter le même composant pour plusieurs opérations différentes et cela avec : une généricité désirée et une bonne vérification à chaque fois lors de la simulation.

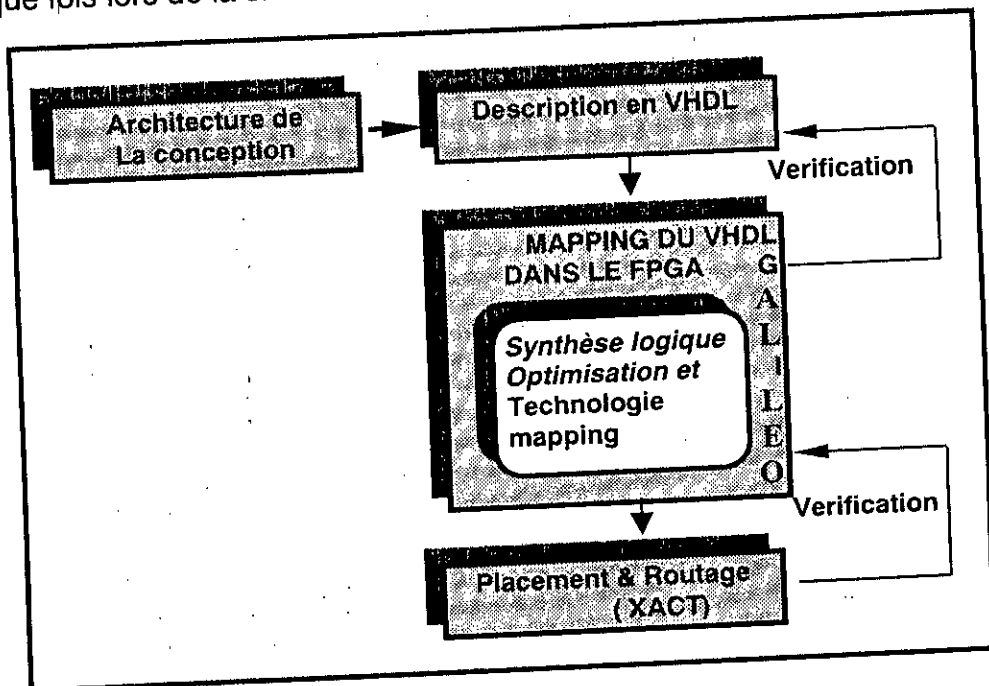


Figure 3.6 : Méthodologie de conception utilisant les outils Galileo et XACT

J'exprime aussi ma profonde reconnaissance pour la documentation qu'il m'a fournie, les conseils qu'il m'a donnés, ainsi que pour son assistance sans relâche et la patience et sa compréhension dont il a fait preuve à mon égard. Je lui dis « Merci de vous avoir constamment occupés de moi ».

Je remercie vivement les membres du Jury :

Monsieur K. Verdjani, Maître de conférences à l'Institut d'Électronique de l'USTB, pour l'honneur qu'il m'a fait en acceptant la charge de participer et de présider le Jury de cette thèse.

Je tiens à exprimer ma gratitude à Monsieur B. Bouzouia, Maître de recherche au CDTA, pour m'avoir continuellement encouragé dans mes travaux de recherche depuis mon intégration dans le laboratoire « Robotique et Intelligence Artificielle ». Je suis également reconnaissant pour l'intérêt qu'il a porté à ce travail et d'avoir accepté de l'analyser en faisant l'honneur de participer au Jury.

Je remercie Monsieur N. Ammour, Chargé de cours à l'USTB, pour avoir accepté de m'apporter ses remarques pour l'avancement du présent travail. Je tiens à témoigner ma profonde gratitude pour l'honneur qu'il m'a fait en acceptant de porter attention à ce travail et de faire parti du Jury.

Je tiens également à remercier Monsieur HADJ RABAH, Chargé de cours à l'USTB, mon co-promoteur à l'Université de Blida, pour sa contribution essentielle à l'avancement de ce travail. Je le remercie pour l'évaluation de ce projet et en acceptant de participer au Jury de thèse.

Je remercie tous les enseignants de l'Université de Blida, pour les enseignements précieux acquis pendant le cursus universitaire.

Je tiens à remercier vivement les membres du laboratoire Robotique et Intelligence Artificielle pour la sympathie et l'aide morale qu'ils m'ont apportée.

Je tiens à exprimer ma profonde reconnaissance aux membres du laboratoire Micro électronique et du laboratoire Architectures et Systèmes pour la sympathie et le soutien moral dont ils ont fait preuve à mon égard.

Enfin, que tous ceux qui, de près ou de loin, ont contribué à l'élaboration de ce travail trouvent ici l'expression de ma sincère gratitude.

Ouarda



B.8. DESCRIPTION DIGITALE DE L'ARCHITECTURE DEVELOPPEE

Une fois une image est acquise, elle est toujours mémorisée en mémoire. On peut accéder à cette image grâce à des interfaces. Les données de cette image sont envoyées à la mémoire RAM externe du circuit FPGA. Par la suite, ces données sont traitées par notre circuit FPGA (voir la figure 3.7).

Nous proposons l'architecture de FPGA pour le modèle du *AG-LF* (voir la figure 3.7). Comme l'illustre cette figure ce modèle est principalement basé sur quatre composants décrits comme suit :

- Une partie mémoire RAM : qui permet de stocker les données qui viennent de l'extérieur. Les données issues après la perception vont être adressées à cette mémoire externe de notre circuit pour être utilisées par le second étage.
- Une commande : cette commande permet de donner un ordre et de choisir un environnement parmi ceux qui ont été proposés pour la navigation.
- Un composant *AG-LF* qui permet d'optimiser et de trouver le chemin le plus optimal.
- Un composant *LF* : celui ci est destiné à contrôler la trajectoire de référence.
- La partie contrôle qui contrôle la partie opérative du circuit.

B.9. Architecture de composant AG

L'architecture de l'étage d'optimisation qui concerne les algorithmes génétiques est illustrée dans la figure 3.8 composée essentiellement de sous-composants suivants :

- Mux : le multiplexeur permet de sélectionner les facteurs désirés pour une opération donnée. On utilise ce composant par exemple pour sélectionner un nombre désiré pour effectuer la multiplication ou une opération mathématique, ces opérations qui s'effectuent à la fois pour des nombres différents nécessitent un multiplexeur pour sélectionner l'élément adéquat.
- Additionneurs : ces composants qui sont utilisés pour effectuer l'opération d'addition de deux nombres ou plus pour une opération donnée. Par exemple si nous avons la distance de chaque segment du mouvement de notre robot et nous voulons savoir la distance totale parcourue, on utilise un additionneur de N entrées (N représente le nombre de segment) pour calculer la distance parcourue lors de la navigation.
- Soustracteurs : ces composants sont utilisés, par exemple, si nous avons un nombre initial et un nombre final de points de passages d'un robot mobile et nous voulons savoir le nombre du points de passage parcourus entre ces deux positions, nous utilisons le soustracteur pour évaluer ce nombre.

- **Comparateurs** : ils sont utilisés pour effectuer la comparaison entre deux nombres ou deux valeurs , par exemple nous avons utilisé un comparateur qui fournit vers la fin la valeur minimale de deux nombres comparés. Ces composants sont utilisés à titre d'exemple pour le concept génétique, si nous avons le coût du chaque chemin parcouru (la distance) et nous voulons sauvegarder le coût minimal, on utilise un comparateur pour comparer entre ces deux coûts .
- **Compteurs** : ces composants sont utilisés pour plusieurs opérations. Ils sont utilisés par exemple pour compter le nombre des points du passage de notre robot .

B.10. Architecture de composant LF

Le composant LF est constitué de (voir la figure 3.9) :

- une mémoire (RAM) permettant de stocker les règles d'inférences.
- Une mémoire (RAM) permettant de stocker les fonctions d'appartenances.
- Un grand nombre de multiplieurs. Ces composants sont utilisés, par exemple, dans le cas de fuzzification, on a besoin de ces composants pour calculer les produits de degrés d'appartenances avec le centre de gravité.

Comme nous pouvons remarquer, l'architecture résultante a comme caractéristiques :

- Une grande densité d'intégration.
- Un haut degré de complexité.
- Une régularité de la structure.

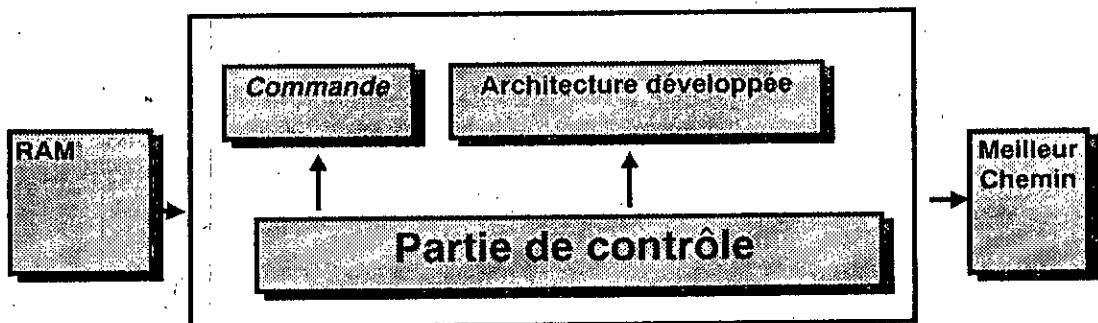


Figure 3.7: Architecture de la conception

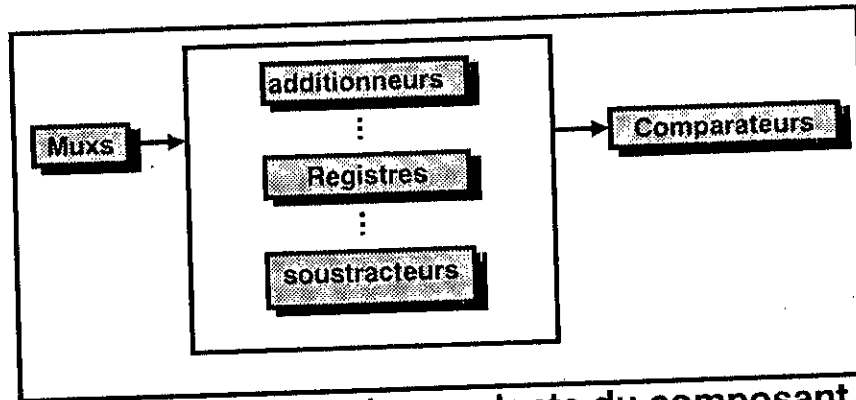


Figure 3.8 : Architecture descendante du composant AG

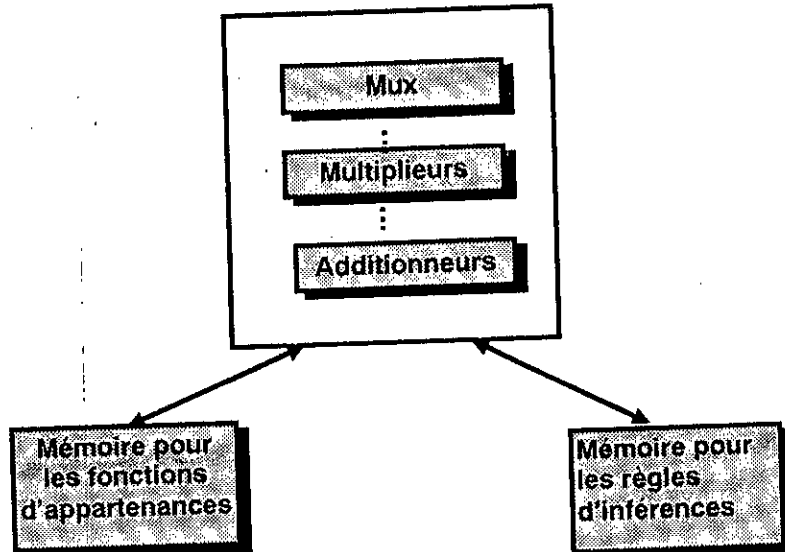


Figure 3.9 : Architecture descendante du composant LF

La section 3.11 et 3.12 expose comment ces composants ont été programmés en VHDL.

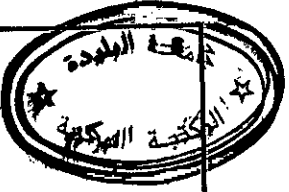
B.11. PARAMETRES DE FLEXIBILITE

Pour réaliser la flexibilité de notre architecture, nous avons déposé au préalable une description de paramètres flexibles. Les paramètres qui réalisent la flexibilité de notre architecture sont : la taille du vecteur d'entrée (nombre de bit), la profondeur des mémoires (la taille) et l'instanciation des composants.

Par conséquent, nous pouvons appliquer cette description à une autre architecture de taille différente. Nous pouvons, grâce à cette description, changer les performances de notre architecture. Avec cette description, nous pouvons facilement modifier le nombre de modèles ou de composants par une simple modification du code VHDL.

B.11.1 Généricité

Nous présentons ci-après un exemple de description d'un additionneur décrit en VHDL :



```
entity ADD is
    generic(WIDTH:integer:=N);

    port (clk,rst:bit;
          A,B,C,D,E,F,G,H,I,J:in unsigned((2*WIDTH) downto 1);
          X10:out unsigned((2*WIDTH) downto 1));

end ADD;
```

Figure 3.10: un exemple de description en VHDL d'un paramètre de flexibilité générique.

Cette figure (3.10) illustre comment la flexibilité a été réalisée où les vecteurs d'entrées sont de taille extensible, en d'autre termes le nombre de bit n'est pas fixe. Le mot clé utilisé est donc le terme *generic*. La générativité est le moyen de transmettre une information (statique) à un bloc. Un bloc générique est vu de l'extérieur comme un bloc paramétré. De l'intérieur du bloc, ces paramètres sont vus comme des constantes et que l'on peut manipuler comme tels.

B.11.2 Instanciation

L'instanciation d'un composant est le fait de prendre une copie d'un composant déclaré et de le personnaliser afin de répondre aux spécifications de la conception. En VHDL, cette instanciation est aussi un héritage. Par exemple, si on a déclaré un composant soustracteur, on va pouvoir instancier un composant **Sous1**, puis une autre fois, un autre composant **Sous2**, etc. (voir le figure 3.12)

La syntaxe de cette instruction est :

Label : *nom_du_composant_modèle*

Si l'on suppose que le composant (modèle soustracteur) est déclaré comme :

```
component sous
    generic (WIDTH:integer:=8);
    port(clk,rst: bit;
          X,Y :in unsigned(WIDTH downto 1);
          S :out unsigned(WIDTH downto 1));
end component;
```

Figure 3.11 : le composant soustracteur en VHDL

Et si on suppose les signaux S1,S2,A1,B1, E1 sont déclarés comme suit :

```
signal S1,S2, A1,B1,E1: unsigned(WIDTH downto 1);
```

On peut instancier ce composant deux fois par :

```
Sous1 : sous generic map (WIDTH)
        port map (clk,rst,B1,A1,S=>S1);

Sous2 : sous generic map (WIDTH)
        port map (clk,rst,E1,A1,S=>S2);
```

Figure 3.12 : exemple d'instanciation en VHDL.

On obtient alors deux composants **Sous1** et **Sous2**, assurant la connexion de nos signaux.

B.12 Description de l'architecture développée et quelques Composants utilisés en VHDL

Lors de la conception, nous avons trouvé un grand problème pour implémenter l'architecture sur le circuit FPGA. A cet effet, nous avons pu descendre encore plus bas pour voir comment on traite une opération arithmétique par exemple.

A titre d'exemple, si on veut effectuer une addition en VHDL, il faut définir le nombre de bits de la donnée, les ports d'entrée (vecteurs d'entrées) et les ports de sortie (vecteur de sortie). La description qui suivra peut éclairer encore plus le détail.

Il est nécessaire, avant de commencer toute phase de construction d'avoir une idée claire de ce que l'on veut construire. Il faut donc définir une boîte noire (spécification d'entité) avant de décrire le fonctionnement de son architecture interne (modèle de construction d'une architecture en VHDL).

Toute conception en VHDL est spécifiée par deux catégories d'unité de conception. La première catégorie consiste en la définition d'une entité (mot clé **entity**), cette dernière est la description d'un système vu comme une boîte noire. La deuxième catégorie définit l'architecture de l'entité (mot clé **architecture**) : c'est le corps de l'entité, elle décrit l'intérieur de la boîte noire.

Pour mieux expliquer, nous allons présenter les modèles ou composants qui ont été utilisés lors de la conception, et nous allons expliquer comment nous avons pu arriver à les concevoir.

B.12.1 EXEMPLE D'UN COMPAREUR

Si on veut concevoir un comparateur à titre d'exemple, nous allons définir tous d'abord notre boîte noire avant de décrire l'architecture de son fonctionnement. Cette boîte noire est donnée comme suit :

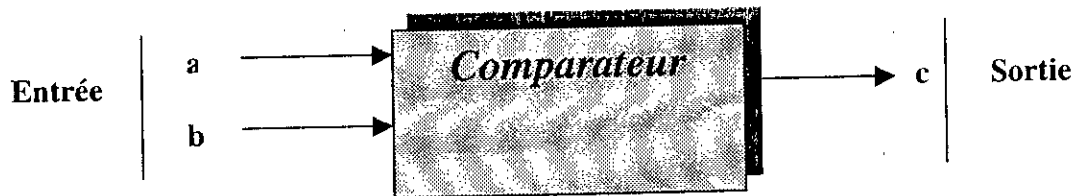


Figure 3.13 : Exemple d'un comparateur de deux entrées.

Ayant cette boîte noire, nous pouvons donc imaginer clairement ce qu'il faut avoir comme entrée et qu'est ce qu'on va avoir comme sortie. Mais cette entité (boîte noire) a besoin d'une architecture pour définir son fonctionnement. A ce stade là, on commence à décrire l'architecture en VHDL. La suite de notre exemple se termine par la description décrite dans la figure 3.14.

Comme le met en évidence cette figure (3.14), nous pouvons bien constater que cette description nécessite une bonne connaissance en VHDL. Comme tout un autre langage de programmation, il faut bien connaître les mots clés (*if*, *endif*, *entity*,...), le raisonnement approprié et la syntaxe d'écriture pour bien décrire une architecture en VHDL. Si on oublie par exemple un point virgule, la simulation ne peut être effectuée.

Nous pouvons aussi décrire ce type de composant par une autre architecture décrite en VHDL (voir la figure 3.15). Ce comparateur renvoie vers la fin la valeur minimale de deux entrées (vecteurs de bits) fournies à l'entrée (même si les deux vecteurs ont la même donnée à présenter). Nous avons utilisé ce type de l'architecture dans la sauvegarde du coût minimal du chemin (distance) lors de l'application du principe génétique au mouvement d'un robot.

B.12.2 Exemple d'un additionneur

Si on veut réaliser un additionneur, il existe plusieurs façons pour définir son architecture en VHDL. Nous pouvons utiliser la table de vérité d'un additionneur (équation déduite), ou bien utiliser deux demis additionneurs pour définir un additionneur complet. La figure 3.16 illustre un exemple d'additionneur en VHDL utilisé lors de la conception.

```

library ieee ;
use ieee.std_logic_1164.all ;

entity compareur is port (
    a,b : in  std_logic_vector ( 3 downto 0);
    C : out std_logic      );

end compareur;

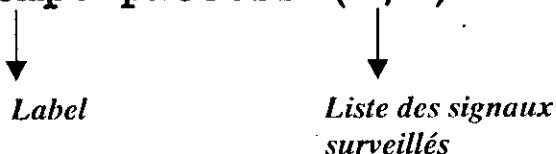
architecture archcomp of compareur is
begin
    comp: process (a,b)
    begin
        if a=b then
            c<='1';
        else
            c<='0';
        end if;
    en process comp;

end archcomp;

```

Figure 3.14 : Description d'un comparateur en VHDL

Lors de la conception, nous avons utilisé des processus qui font l'objet fondamental manipulé par le simulateur. Comme l'indique la figure 3.14 le processus est défini comme :

comp: process (a,b)

Label *Liste des signaux surveillés*

La figure 3.15 illustre un autre exemple de processus donné par :

min: process(rst,clk)

Si le programmeur en VHDL oublie la liste de **sensibilité** (liste des signaux surveillés) la compilation fonctionne normalement avec succès mais la simulation risque de ne pas fonctionner, car la liste de sensibilités joue un rôle important . Le processus fonctionne à boucle sans les signaux de surveillances. Ce piège est l'un des problèmes rencontrés lors de la conception. Nous pouvons conclure que le VHDL est très sensible comme langage de programmation hardware.

```

library ieee;
use ieee.numeric_bit.all;

entity min is
generic(WIDTH:integer:=8);
port(clk,rst: bit;
      X :in unsigned( WIDTH downto 1);
      Y :in unsigned( WIDTH downto 1);
      Z :out unsigned( WIDTH downto 1));
end min;

architecture behavioral of min is
begin
min: process(rst,clk)

begin

if (rst='1') then
    Z<=(others=>'0');

elsif (clk='1' and clk'event) then
    if X<Y then Z<=X;
    elsif X>Y then Z<=Y;
    elsif X=Y then Z<=X;
    end if;

end if;
end process min;
end behavioral;

```

Figure 3.15: exemple d'une autre description de comparateur en VHDL

```

library ieee;
use ieee.numeric_bit.all;

--2*1
entity ADD2 is
generic(WIDTH:integer:=8);
port (clk,rst:bit;
      A,B:in unsigned((2*WIDTH)
downto 1);
      X2:out
unsigned((2*WIDTH) downto 1));
end ADD2;

```

```

Architecture behavioral of ADD2 is
begin
add: Process(rst,clk)
begin
if (rst='1') then
    X2 <=(others=>'0');

    elsif (clk='1' and clk'event)
then
    X2<= ( A+B);

    end if;

end process add;

end behavioral;

```

Figure 3.16 : exemple d'un additionneur décrit en VHDL.

Pour d'autres illustrations, il faut se conférer à l'annexe où nous avons présenté d'autres architectures décrites en VHDL utilisées pour développer notre système.

Notre architecture est composée de ces modèles (additionneurs, soustracteurs, etc). Nous avons utilisé ces modèles comme des sous- composants. Cela signifie que nous commençons par la simulation des composants (additionneurs, soustracteurs, multiplieurs, etc.) . Ensuite, on leur fait appel comme des sous- composants. Ces sous –composants seront à leur tour vus comme des sous-composants et ceci jusqu' à ce que le fonctionnement de la simulation totale (l'architecture finale) . Notre architecture de navigation sera alors vue comme celle présentée dans la figure 3.17.

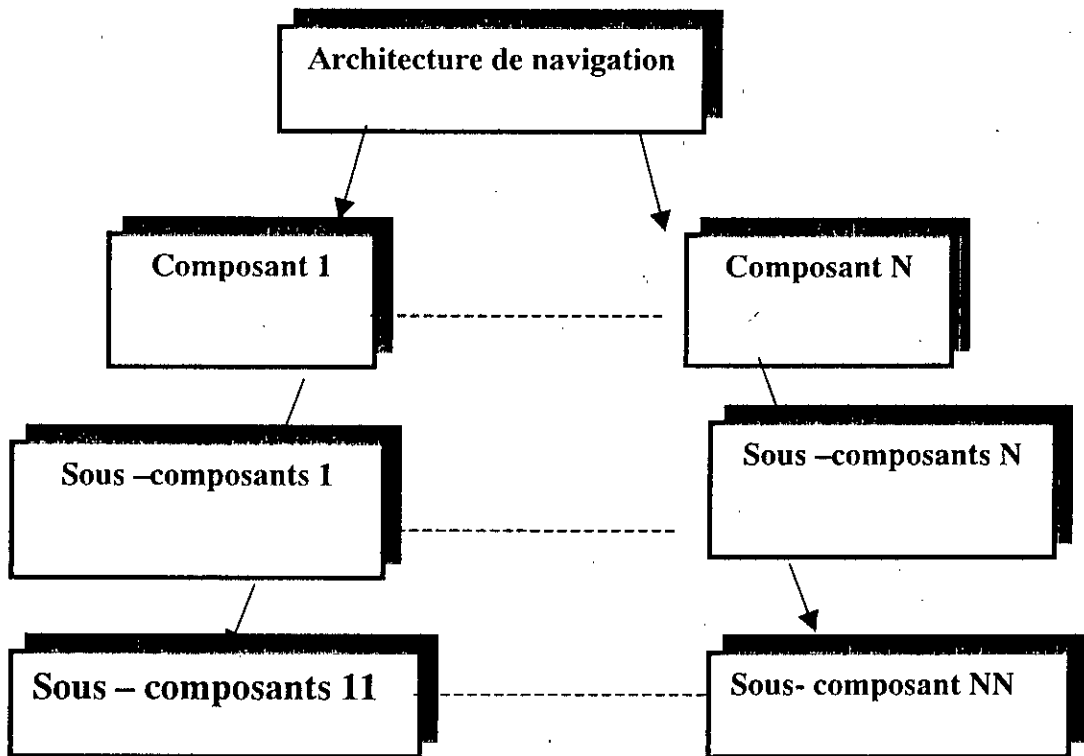


Figure 3.17 : architecture résultante

Notre architecture est alors décrite comme suit :

```

library ieee;
use ieee.numeric_bit.all;

entity navigal is
    generic( WIDTH:integer:= );
    port (clk,rst,load:bit;
        |
end navigal;

architecture behav_navigue of navigal is

component commande --sous-composant
    generic(WIDTH:integer:=);
    port (rst:bit;
        |
end component ;

component REGIS1 --sous- composant
    generic(WIDTH:integer:=8);
    port (clk,rst,load:bit;
        |
end component;

component ADD2 --sous-composant
    generic(WIDTH:integer:=8);

        |
end component;

component sous2 -sous-composant
    generic (WIDTH:integer:=8);

        |
end component;

component min --sous-composant
    generic(WIDTH:integer:=8);

        |
end component;

component navigue 1 --composant
    generic(WIDTH:integer:=8);

        |
end component;

begin

naviga_1 : naviguel generic map (WIDTH)
    port map (clk,rst,load, A,B,E,AL,F,K,sort=>sort1);

end behav_navigue;

```

Figure 3.18 : Pseudo-code en VHDL de l'architecture de navigation.

SOMMAIRE

| | |
|--------------|---|
| Dédicaces | |
| Remerciement | |
| Résumé | |
| Abstract | |
| Sommaire | |
| Introduction | 1 |

CHAPITRE 1 : LA NAVIGATION DES ROBOTS MOBILES 4

| | | |
|--------|---|----|
| 1.1 | Introduction | 4 |
| 1.2 | Bref historique. | 4 |
| 1.3 | Définition | 5 |
| 1.4 | Moyens de perception. | 5 |
| 1.4.1 | Systèmes télémétriques passifs . | 5 |
| 1.4.2 | Systèmes télémétriques actifs. | 5 |
| 1.5 | Décomposition canonique d'un robot mobile autonome. | 6 |
| 1.5.1 | Contrôle . Commende. Communication. | 6 |
| 1.5.2 | Problèmes de prises de décision. | 7 |
| 1.6 | Robotique et Intelligence Artificielle. | 8 |
| 1.7 | Navigation des robots mobiles | 10 |
| 1.7.1 | Modèle de grillé. | 10 |
| 1.7.1 | Modèle polygonal. | 11 |
| 1.8 | Planification de la trajectoire. | 11 |
| 1.8.1 | Planificateur global (approche statique) | 11 |
| 1.8.2 | Planificateur local (approche dynamique). | 12 |
| 1.8.3 | Planificateur mixte. | 12 |
| 1.9 | Problématique de naviguer intelligemment dans un environnement Inconnu. | 13 |
| 1.10 | Choix de la méthode de modélisation . | 14 |
| 1.10.1 | Principe du modèle de grille . | 15 |

3. Conclusion

Ce chapitre expose les circuits ASICs en général, et comme circuit ASIC notre travail se focalise sur le circuit FPGA, vue les pertinences et les performances offertes encore plus par ce type de circuiterie.

Dans ce chapitre, nous avons proposé une nouvelle méthodologie pour l'implémentation d'une stratégie de navigation d'un robot mobile, basée sur la synthèse et en utilisant le langage de description VHDL. L'architecture proposée pour la navigation est complexe et nécessite un temps énorme pour la conception. L'utilisation d'un outil de synthèse comme Galileo permet de réduire considérablement ce temps. Le circuit FPGA obtenu a été synthétisé.

Nous avons proposé une architecture FPGA régulière et intégrant un haut degré de parallélisme. Nous avons aussi proposé une approche de description VHDL flexible et paramétrique, par de simples modifications, le robot peut se retrouver. De plus, l'utilisateur peut modifier cette architecture dans un cas différent ou par exemple le nombre de bits des vecteurs utilisés ou la profondeur de la mémoire sont changés.

Chapitre 4

SIMULATION

A. CONCEPTION SOFTWARE

Nous présentons dans ce qui suit , le progiciel *AG_LF* que nous avons développé en laboratoire de Robotique et Intelligence Artificielle, en donnant toutes les étapes nécessaires pour son utilisation. Comme nous avons déjà expliqué dans le premier chapitre, ce travail est consacré à la conception d'un planificateur local, pour émerger les techniques d'Intelligences Artificielles avec la robotique mobile : deux domaines complémentaires difficile à séparer nettement. Nous présentons les étapes suivies pour parvenir à la construction de la trajectoire optimale.

Notre plan de recherche est décomposé en deux parties, la première consiste en le développement d'un progiciel capable d'offrir un comportement intelligent pour une navigation donnée. Ce progiciel a comme objectif de trouver un chemin qui peut mener le robot d'un point source vers un point but. Notre progiciel a été développé en utilisant un langage de programmation très évolué (Borland C++). Le choix de ce langage est dû à ses multiples caractéristiques et avantages, citons par exemple, la richesse de son environnement et ses possibilités graphiques multiples. Le deuxième plan de la recherche consiste en : l'implémentation de l'algorithme « *développé déjà en Software* » sur une carte électronique dotée de circuit FPGA en utilisant le langage de description matériel très évolué (VHDL). Cette tâche constitue le fond du sujet : nous voulons que la navigation s'effectue grâce au circuit FPGA pour un usage spécifique. La première partie de notre conception (progiciel) a pour but, en effet, d'étudier la faisabilité de ce qui va être implémenté en FPGA.

A cet effet nous allons exposer deux sections :

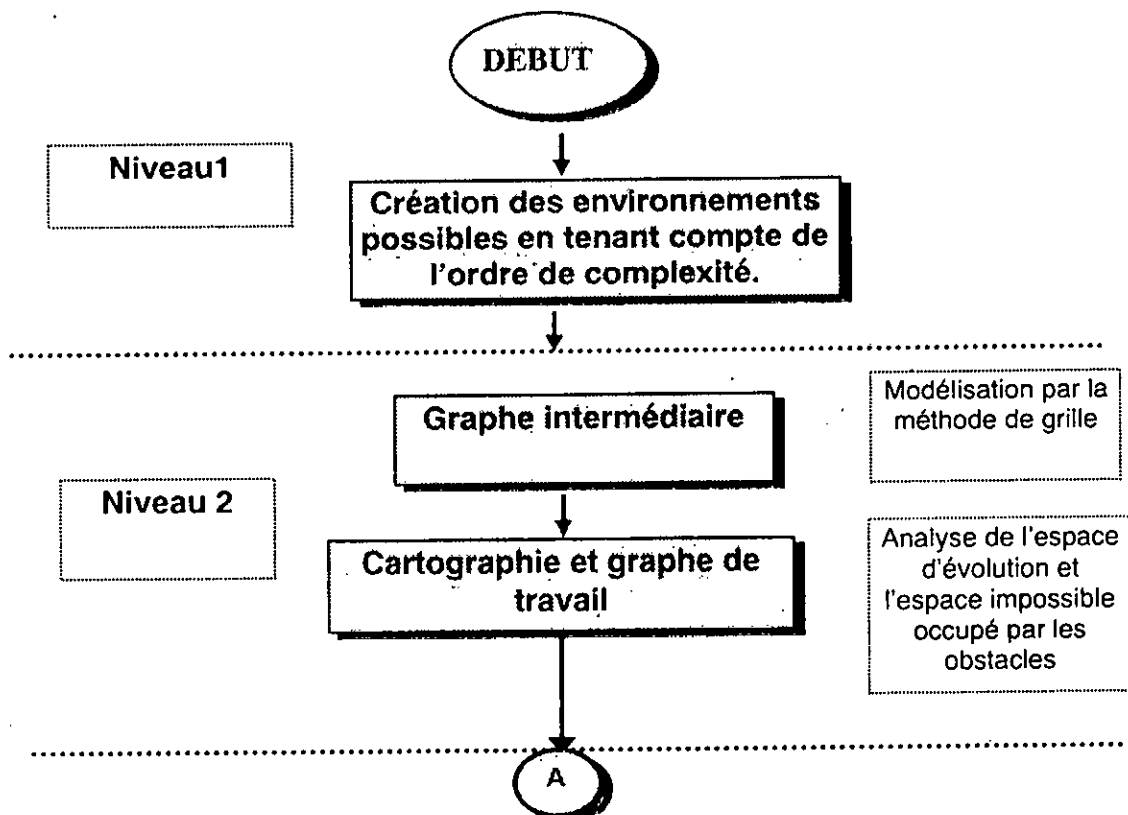
- La première est consacrée à présenter toutes les étapes nécessaires pour le développement du progiciel *AG_LF*.
- La deuxième consiste en la description hardware de l'architecture *AG_LF* par l'implémentation sur une carte électronique dotée de circuit FPGA.

A.1. Présentation de progiciel *AG_LF*:

Les approches actuelles de planification et de contrôle, nécessitent certains outils comme : *AG* et *LF*. Ces outils ont été utilisés soit en systèmes intelligents se basant sur l'une de ces approches actuelles, soit en systèmes hybrides intelligents *SHI* se basant sur les différentes combinaisons de ces approches exploitant leurs propriétés complémentaires. En effet, l'intégration des *AGs* et la *LF* s'est avérée une solution pour développer des applications utiles dans des environnements réels.

A.1.1. L'organigramme de travail

Le but de cette réalisation est de toucher l'avantage d'un planificateur local de trajectoire capable de générer des chemins optimums dans un environnement non défini. L'approche hybride combinée par les techniques citées précédemment pour donner un système *SHI* est résumée par l'organigramme présenté dans la figure 4.1 regroupant toutes les phases et les méthodes de traitement nécessaires pour sa réalisation.



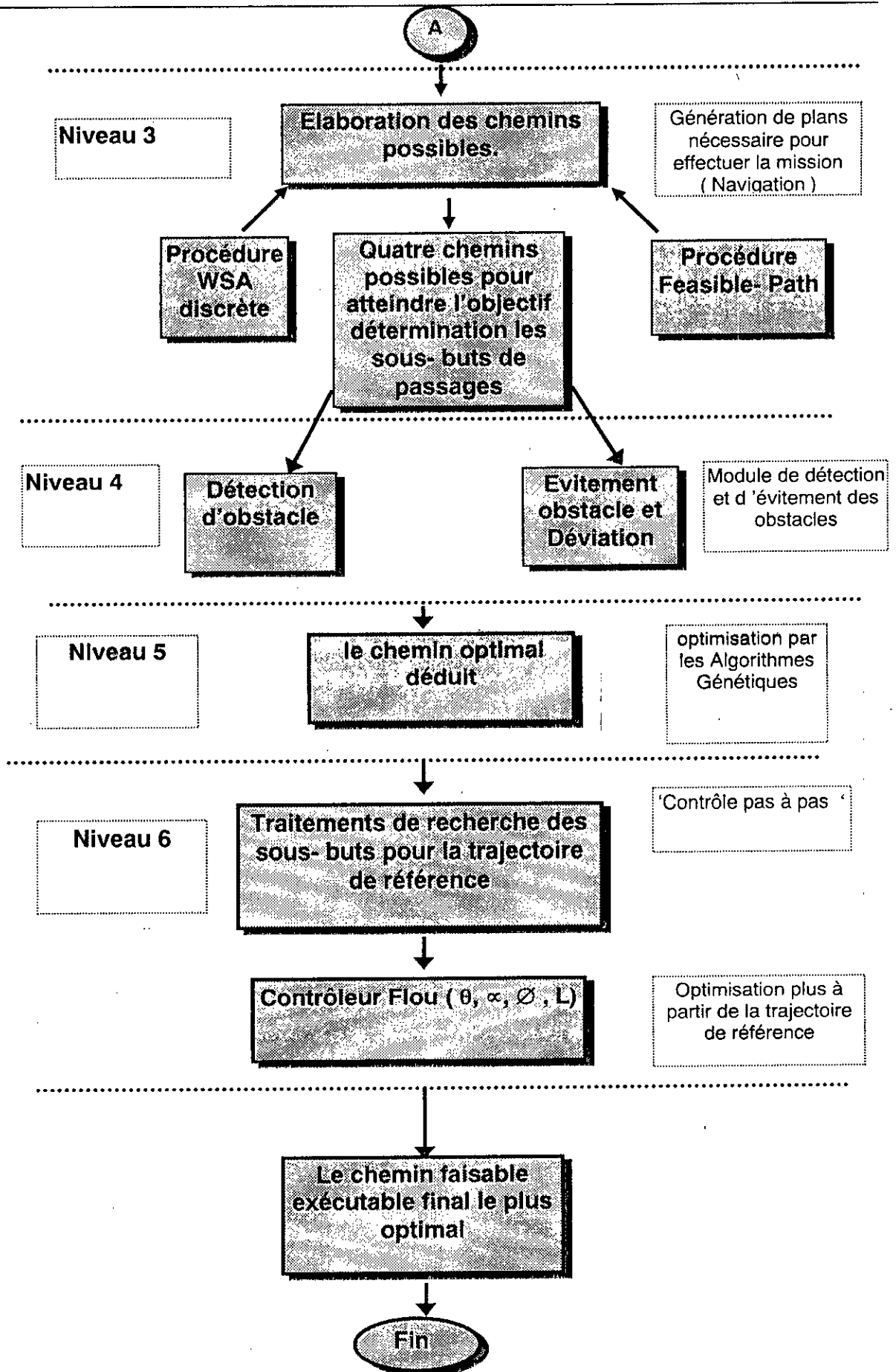
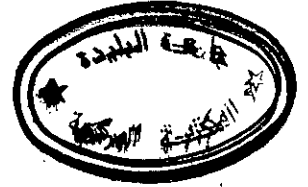


Figure 4.1 : Organigramme de travail



Les niveaux de l'organigramme sont détaillés dans ce qui suit :

Niveau 1

Le niveau 1 est réservé à l'utilisateur . Ce dernier pourra choisir et définir un environnement parmi les 16 qui ont été proposés pour la navigation. Ce choix n'est pas fixe, car la structure de l'environnement n'est pas connue donc l'utilisateur pourra déplacer les objets de la scène de navigation, ainsi qu'il pourra créer d'autres environnements selon la tâche désirée pour accomplir la mission du robot.

Cette création est peut être accompagnée par un changement de dimensions des obstacles, le déplacement (décalage), l'enlèvement et l'ajout des objets possibles pour réaliser la tâche désirée, rien ne change du moment qu'aucune information n'est fournie dès le départ sur la scène de navigation. Cette flexibilité est accompagnée par la prise en compte de certain ordre de complexité de l'environnement (nombre maximal).

La tâche intelligente démarquée par notre progiciel *AG_LF* peut réaliser le traitement en temps réel et répond en même temps aux exigences de l'autonomie.

Niveau 2

Ce niveau est consacré à la modélisation de l'environnement, dont on obtient un graphe composé d'un ensemble de chemins libres et de chemins non atteignables. L'analyse de l'espace permet d'avoir un graphe finalisé dit " *graphe de travail* ". Comme nous l'avons déjà exposé dans le premier chapitre, nous avons utilisé le modèle de grille qui était bien convenable dans notre cas pour structurer l'espace d'évolution, un graphe de travail est alors obtenu pour analyser l'espace libre d'évolution et l'espace occupé par les obstacles.

Niveau 3,4

Ce niveau correspond à l'élaboration des chemins possibles pour atteindre l'objectif en touchant deux algorithmes similaires WSA et Feasible Path . Ces deux techniques sont bien convenables du moment que l'environnement n'est pas connu. Ces méthodes qui ont été développées simultanément marquent une génération de plans importante dans la mesure elle peut générer tous les chemins possibles en évitant les obstacles. Ces deux méthodes sont décrites ci-dessous :

Procédure WSA discrète [52]:

1. Si la ligne directe à l'objectif n'est pas interrompue par un obstacle, exécuter la trajectoire et terminer avec succès.
2. Déterminer les candidats probables pour un *PWE* (*Passage Way Entrances*) sur les deux cotés de la ligne directe et les arranger dans l'ordre croissant de préféralité, si aucun n'existe, terminer avec échec.

3. Sélectionner le *PWE* avec le maximum de préférabilité, rendre ce point le prochain point de *SP* (*StandPoint*).
4. Si ce point existe déjà dans la liste des solutions, prendre le prochain point candidat comme prochain *SP*.
5. Exécuter le chemin dans la direction du prochain *SP* et mettre le *SP* courant dans la liste des solutions.
6. Aller à 1.

Procédure *FEASIBLE – PATH*

1. Initialisat : $A = IP$ (point intermédiaire), $B = \text{But}$. *Path - Node - List* (liste des nœuds du chemin) = NUL, $PRED (IP)$ (ascendant de IP) = NUL.
2. Si le segment AB n'est pas interrompu par un obstacle (B est visible à partir de A), exécuter le chemin directement à B .
3. *Candidat - List* (liste des nœuds candidats) = { nœuds N : AN est un chemin ininterrompu ; N est un point tangentiel local par rapport à A }.
 - 3.1. N est un point tangentiel local par rapport à A , si N est un nœud de l'obstacle tel qu'au voisinage de ce nœud l'obstacle se trouve entièrement d'un seul cote de la ligne AN .
 - 3.2. Les nœuds de la liste *Candidat - List* sont probablement candidats pour un *PWE*.
4. Si « *Candidat- List* » = NUL, alors « *Next - Node* » (prochain nœud) = $PRED (A)$; sinon le nœud le plus préférable dans « *Candidat - List* » ; et $PRED (\text{Next- Node}) = A$.
5. Si « *Next - Node* » = NUL, terminer l'exécution avec échec (pas de chemin possible au but) ; sinon « *Path- Node- List* » = A ; exécuter le chemin A à « *Next - Node* », et $A = \text{Next - Node}$, aller à l'étape 2.

Cette génération de plan a donc pu déterminer toutes les trajectoires possibles pour atteindre l'objectif en tenant compte les contraintes qui devaient être embarquées par le robot lors d'accomplissement d'une certaine tâche intelligente.

Pour résumer, nous pouvons dire que nous avons déterminé les candidats les plus probables pour la navigation en évitant les obstacles.

Niveau 5

Cet étage est consacré à l'optimisation par l'algorithme génétique : cet algorithme permet de trouver la trajectoire la plus optimale (en basant sur le paramètre *distance*) sous forme de points de passage sans collisions entre le point Source et le point But. Le procédés a été déjà expliqué dans le deuxième

chapitre. Nous avons pu établir une stratégie de tri des coûts des chemins pour trouver le chemin le plus optimal au sens génétique.

Niveau 6

Pour une bonne exécution de la trajectoire de référence (issue de l'étage précédent), nous pourrions encore plus optimiser cette référence en appliquant le contrôleur logique flou, ce dernier peut encore nous offrir une bonne assurance d'atteindre la cible en évitant les obstacles et réaliser un bon lissage à fin de gagner et réaliser les exigences d'autonomies. Le résumé récapitulatif du principe de contrôleur flou appliqué au mouvement du robot (expliqué déjà dans le chapitre 2) montre que la logique floue est un outil très puissant pour fournir la meilleure orientation à notre robot, cet outil peut faciliter la modélisation des problèmes basés sur le raisonnement humain sans avoir recours à des constructions mathématiques complexes.

A.2. Progiciel et Résultats de la simulation

Afin de refléter les comportements d'un robot mobile autonome dans des environnements inconnus, et de démontrer la capacité de naviguer sans crainte ou risque de collisions, la navigation de notre robot est simulée dans différents environnements non structurés (sauf le point source et le point but).

Notre progiciel offre à l'utilisateur l'accomplissement de la tâche désirée dans les seize (16) environnements proposés pour la navigation dont la structure est complètement inconnue (sauf la position de l'objectif et la source), cette configuration est accompagnée par une flexibilité inhérente dans la mesure que les seize scènes de navigation ne sont pas fixées, l'utilisateur peut déplacer les positions des objets (obstacles) comme il veut pour accomplir la tâche, rien ne change du moment qu'aucune structure n'est définie dès le départ. Cette flexibilité est envisagée dans la mesure de respecter un certain ordre de complexité de l'environnement, cela signifie que, nous commençons par une scène vide jusqu'à atteindre le nombre maximal de la complexité de la scène (le nombre maximum des obstacles).

Cette flexibilité offre une opportunité importante qui se démarque par un caractère intelligent du fait que le robot peut se familiariser avec un autre environnement en dehors de ceux proposés pour la navigation et trouvera parfaitement son chemin le plus optimal pour atteindre l'objectif malgré que le monde externe est changé et les objets sont déplacés ou enlevés. De cela, un nombre infini d'environnements peut être exploité .

Le robot est représenté par un carré de 8X8 pixels. Il perçoit l'environnement à l'aide d'une caméra. On balaie pixel par pixel à partir d'un point d'origine S, et on effectue le test sur la couleur renvoyée par le pixel. Si sa couleur est la même pendant le long chemin (la couleur permise pour la navigation) : effectuer le mouvement et avancer sans crainte. Mais si la couleur est différente de celle habituelle pour la navigation (la couleur de la zone de sécurité), alors un obstacle est détecté, on arrête le balayage (les candidats les plus probables pour la navigation) et on détermine à nouveau les autres candidats

à partir du point où l'anomalie est détectée (pixel par pixel) jusqu'à avoir un côté complet d'un obstacle et on se retrouve avec les nouveaux sous points du passage.

L'utilisateur fait rentrer un nombre qui revient sur un indice représentant un environnement proposé pour la navigation (il peut choisir un parmi ceux proposés pour la navigation ou de créer d'autres environnements pour effectuer la mission demandée). Par la suite, après avoir analysé l'espace libre et l'espace occupé, notre système génère un certain nombre de trajectoires possibles pour atteindre l'objectif. Ce nombre est fixé à quatre (4).

La première trajectoire prend le côté gauche (par rapport au robot) de la navigation où l'obstacle est détecté. La deuxième trajectoire prend le côté droit de déviation où l'obstacle est détecté. La troisième prend l'extrémité gauche de la scène de navigation (par rapport au robot) et la quatrième trajectoire prend l'extrémité droite (par rapport au robot) de la scène de navigation. La figure 4.2 montre ces différentes trajectoires.

Le concept génétique est alors appliqué, à chaque génération on sauvegarde le coût minimal de chaque chemin. Après progéniture on se retrouve avec la trajectoire la plus optimale au sens génétique (le paramètre est la distance).

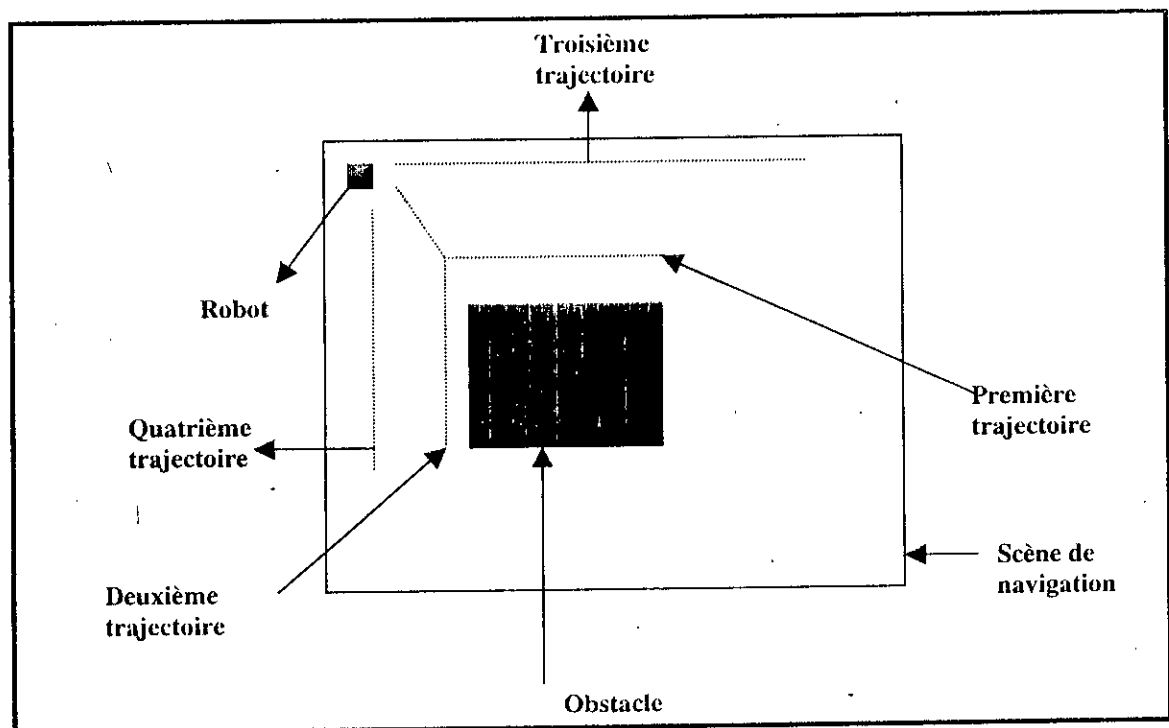


Figure 4.2 : Différentes trajectoires

La figure 4.4 présente un cas d'environnement simple. Cet environnement illustre une scène de navigation vide (non parsemée par des obstacles). Les résultats de l'analyse sont illustrés dans la figure suivante :



| <i>Différentes trajectoires</i> | <i>Distances parcourues (pixels)</i> |
|---------------------------------|--------------------------------------|
| <i>Première trajectoire</i> | 432 |
| <i>Deuxième trajectoire</i> | 432 |
| <i>Troisième trajectoire</i> | 488 |
| <i>Quatrième trajectoire</i> | 488 |

Figure 4.3 : Tableau des résultats.

Ayant la trajectoire la plus optimale de référence (issue du principe génétique, le paramètre est la distance), nous allons appliquer le contrôleur flou pour réaliser une bonne optimisation et un bon lissage, notre contrôleur délivre la meilleure orientation pour que le robot puisse rejoindre son objectif sans risque de collisions.

Dans ce cas, Le chemin est retrouvé facilement pour mener le robot d'un point de départ vers un point d'arrivée. C'est le cas d'une navigation non intelligente.

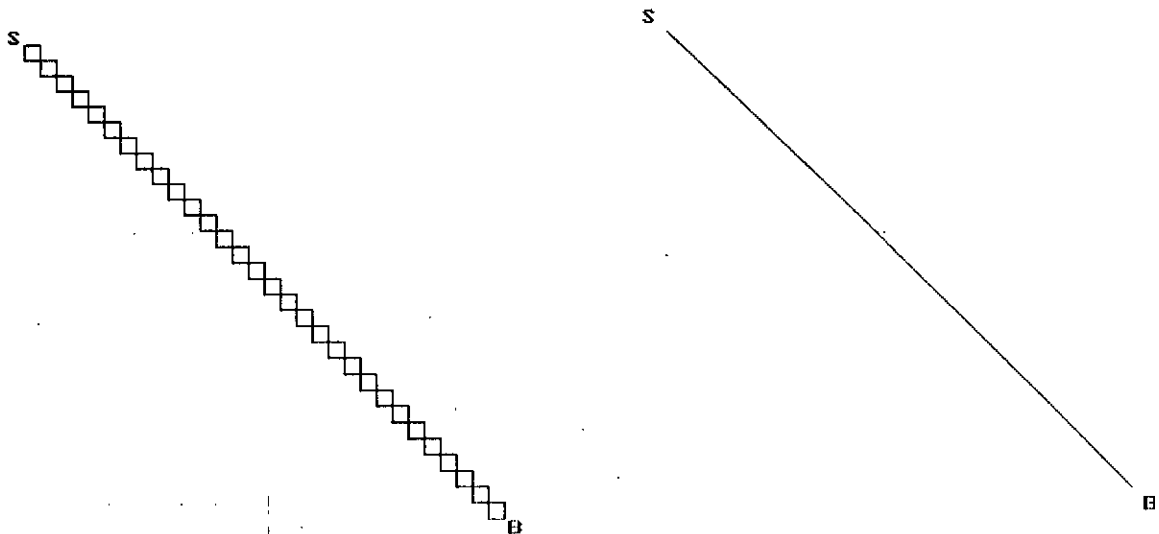


Figure 4.4 : Cas d'une scène de navigation vide (navigation non intelligente).

Notons que la trajectoire finale à exécuter permet à l'utilisateur de changer l'encombrement du robot. Si une forme sphérique est introduite par exemple, l'utilisateur prend la trajectoire finale et exécute la mission demandée.

Si le même coût apporté (distance) pour deux trajectoires différentes, le système choisit au hasard une trajectoire parmi les deux et applique le principe du contrôleur flou.

Testé dans un autre environnement plus complexe, le robot trouve son chemin parfaitement sans crainte ou risque de collisions (voir la figure 4.5) .

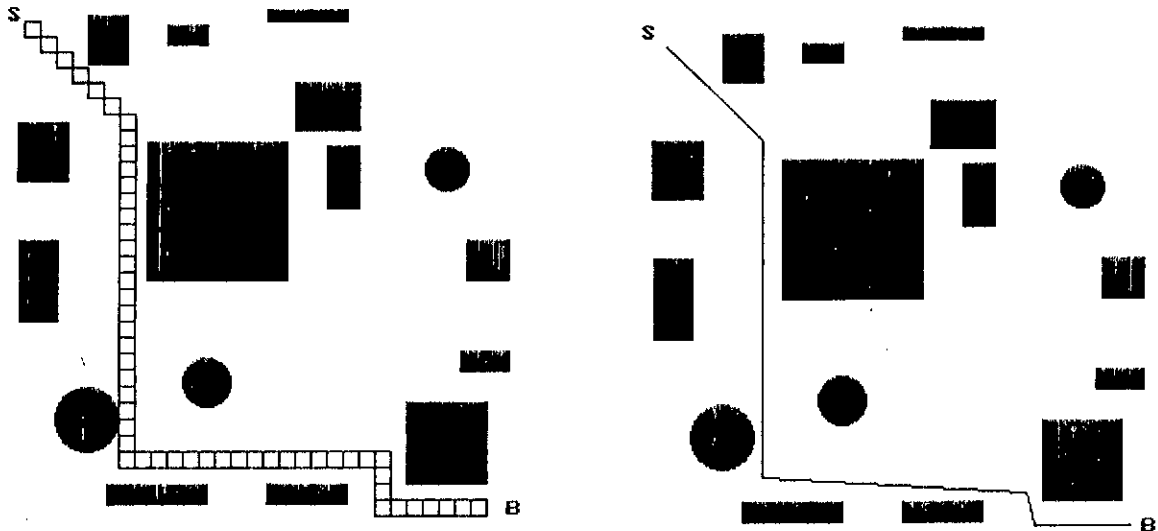


Figure 4.5 : Cas d'un environnement complexe

La figure 4.6 illustre un cas d'environnement moins complexe que celui proposé dans la figure 4.5 (on diminue l'ordre de complexité). Dans ce type de configuration, le robot trouve son chemin avec succès pour rejoindre son objectif.

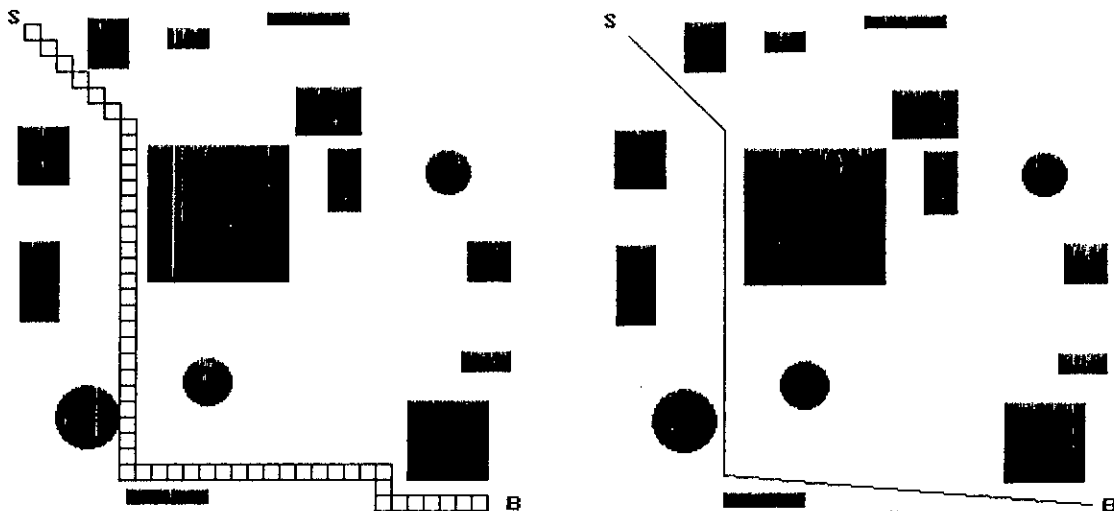


Figure 4.6 : Cas d'un environnement moins complexe.

| | | |
|--|---|-----------|
| 1.10.2 | Planificateur local appliqué à l'approche. | 15 |
| 1.10.2.1 | Approche développée. | 16 |
| 1.10.2.2 | Génération de plan | 16 |
| 1.10.2.3 | Architecture hiérarchisée. | 17 |
| 1.11 | Conclusion. | 18 |
| CHAPITRE 2 : PERFORMANCES DES ALGORITHMES GENETIQUES ET LA LOGIQUE FLOUE. | | 19 |
| A ALGORITHMES GENETIQUES. | | 19 |
| A.1 | Introduction. | 19 |
| A.2 | Définition. | 20 |
| A.3 | Le croisement « <i>CROSSOVER</i> ». | 20 |
| A.4 | Sélection. | 21 |
| A.5 | Mutation. | 22 |
| A.6 | Fonctions d'adéquation « <i>FITNESS</i> ». | 22 |
| A.7 | Algorithmes Génétiques appliqués au problème de la navigation | 22 |
| B LOGIQUE FLOUE. | | 25 |
| B.1 | Introduction. | 25 |
| B.2 | Définition. | 25 |
| B.3 | Les bases de la logique floue. | 25 |
| B.3.1 | Théorie des ensembles flous. | 26 |
| B.3.2 | Définition. | 26 |
| B.3.2.1 | Définition d'un ensemble flou. | 26 |
| B.3.2.2 | Définition d'un ensembles flou vide. | 26 |
| B.3.2.3 | Définition de deux ensembles flous égaux. | 26 |
| B.3.2.4 | Définition de l'intersection de deux ensembles flous. | 26 |
| B.3.2.5 | Définition de l'union de deux ensembles flous. | 26 |
| B.3.2.6 | Définition du complément d'un ensemble flou. | 27 |
| B.3.2.7 | Définition de l'implication de deux ensembles flous. | 27 |
| B.3.2.8 | Définition d'un sous ensemble flou. | 27 |
| B.4 | L'inférence floue. | 27 |
| B.5 | Définition d'un contrôleur flou. | 27 |
| B.5.1 | Fuzzification. | 28 |
| B.5.2 | L'inférence floue. | 28 |
| B.5.3 | Défuzzification. | 28 |
| B.6 | Contrôleur floue appliqué au mouvement du robot. | 29 |
| 2 | Conclusion. | 33 |
| CHAPITRE 3 : LES ASICs ET LES FPGAs | | 35 |
| A CIRCUITS ASICs | | 35 |
| A.1 | Introduction | 35 |
| A.2 | Définition. | 36 |
| A.3 | Styles de conception. | 36 |
| A.3.1 | Circuits sur mesure « <i>FULL CUSTOM</i> ». | 37 |
| A.3.2 | Circuits pré caractérisés « <i>Standard Cells</i> ». | 37 |
| A.3.3 | Circuits pré diffusés « <i>Gate Array</i> ». | 37 |
| A.3.4 | Réseaux logiques programmables « <i>FPGA, PLD</i> ». | 37 |
| A.4 | Avantages et inconvénients des ASICs. | 37 |

Notre architecture fait une étude complète pour calculer et tester les différents chemins à prendre pour atteindre l'objectif , et comme pour le système naturel (étude comportementale) qui s'expose par une capacité logique qui diffère d'un être humain à un autre pour répondre à un problème donné. Notre architecture *AG_LF* répond parfaitement mais prenant un peu du temps pour délivrer le chemin le plus adéquat, et cela même si on choisit le meilleur outil informatique pour exécuter les instructions de progiciel *AG_LF*.

A cet effet, nous avons développé une architecture hardware adéquate à la navigation d'un robot mobile autonome, pour planifier et contrôler une trajectoire donnée dans un environnement inconnu dans les meilleures conditions. Le concept software fait ressortir une étude complète sur la faisabilité de cette implémentation , donc on réduit encore plus la tâche de contrôle effectuée par la partie contrôle de notre circuit FPGA. Il était donc nécessaire d'implémenter l'architecture *AG_LF* sur une carte électronique dotée de circuit FPGA afin de répondre encore plus aux exigences d'autonomies et surtout si c'était le cas d'une application demandée dans des environnements réels où la contrainte du temps est le facteur le plus important pour répondre à un problème donné (opératoire) .

B. CONCEPTION HARDWARE

Dans la section suivante, on propose notre architecture implémentée sur le circuit FPGA. Nous avons utilisé le langage VHDL pour décrire l'architecture de conception. Pour la synthèse, nous avons utilisé l'outil Galileo, notons que celui-ci peut estimer parfaitement la surface du silicium de notre circuit en tenant compte un compromis entre le temps et la vitesse. Le résultat doit être un fichier de type Xnf qui va être fourni à l'entrée XACT, ce dernier fait le placement et le routage délivrant ainsi le circuit final *FPGA*.

Nous voulons que la navigation s'effectue grâce au circuit FPGA pour répondre encore plus aux exigences d'autonomie. Ce circuit répond parfaitement au problème posé. Le processeur ainsi développé peut dégager les meilleurs systèmes développés au Soft-Computing.

B.1 PROBLEME POSE

Les problèmes que nous avons rencontré lors de la conception sont les suivants :

- Comment concevoir l'architecture de la navigation ?
- Par quel moyen nous pouvons passer de l'architecture développée en software à une architecture similaire mais en utilisant le concept hardware ?

Pour développer un progiciel donné, nous savons très bien que le microprocesseur est l'organe essentiel qui exécute les instructions du ce progiciel. Il est clair que le traitement des données est une des fonctions essentielles du microprocesseur. Le traitement des données comprend à la fois les calculs et la manipulation des données.

Cependant, le microprocesseur est composé de deux parties : la partie opérative (chemin de données) et la partie contrôle. La décomposition d'un système digital en partie opérative et partie contrôle permet de simplifier la conception et aussi de mieux représenter un microprocesseur complexe. A cet effet, nous avons eu l'idée de baptiser notre architecture en deux parties essentielles : la partie opérative et la partie contrôle.

La partie opérative (exposée déjà dans le troisième chapitre section 3.B.8) utilise des unités de traitement comme : des additionneur, des soustracteurs, des comparateurs ou d'autres composants pour effectuer les calculs ou traitement.

La logique de contrôle est une autre partie importante de l'architecture. Son rôle est de permettre à tous les éléments qui constituent la partie opérative de travailler ensemble et dans le bon ordre, et c'est là que l'on trouve le plus d'erreurs quand on met au point une architecture conçue. La partie contrôle indique au chemin de données ce qu'il doit faire à chaque cycle d'horloge.

Pour implémenter notre modèle AG-LF sur le circuit FPGA, nous avons utilisé une architecture descendante. Nous allons décrire cette architecture en VHDL en utilisant des modèles ou des composants de base. Ces composants (

additionneurs, soustracteurs, multiplexeurs, ...) doivent être vérifiés dans toutes les étapes de conception.

Un simulateur en VHDL doit vérifier si le modèle est syntaxiquement correct. Une fois la syntaxe est vérifiée, le simulateur procède à la vérification fonctionnelle du modèle. C'est à partir des stimulus fournis par les entrées que le simulateur évalue les expressions d'un processus via les pilotes.

Le VHDL est un langage qui est conçu avec une sémantique de simulation. A cet effet, nous l'avons utilisé pour développer notre architecture. A chaque étape, nous vérifions la fonctionnalité avant d'entamer la nouvelle étape et ceci jusqu'à la vérification complète de l'architecture .

Une fois que la vérification de la simulation est effectuée, nous allons décrire d'une manière structurelle ou physique. Cette dernière définit la synthèse. Comme l'outil de synthèse nous avons utilisé l'outil Galileo, ce dernier réalise automatiquement le mapping du VHDL en technologie FPGA. L'outil de synthèse Galileo estime la surface des circuits en termes de bloc logiques configurables (CLB) avec une bonne optimisation (temps et vitesse), c'est pourquoi nous l'avons choisi lors de la conception.

B.2 OBJECTIFS DE LA CONCEPTION

Les modèles et composants de notre architecture doivent répondre aux objectifs suivants :

- Simplicité de conception : une simplicité d'une architecture est basée sur des copies de quelques cellules simples.
- Régularité : la régularité d'une structure a pour objectif de réduire les interconnexions.
- Réduction et extension de l'architecture sont possible.

Ces objectifs présentent une bonne implémentation VLSI, qui peut intégrer un haut degré de complexité, une haute performance, une répétition (ou copie) de cellules simples et une diminution de la surface du silicium. Notre architecture développée en VHDL peut répondre à ces objectifs.

B.3.CARACTERISTIQUES DE L'ARCHITECTURE DEVELOPPEE

B.3.1 Description flexible

Nous nous sommes fixés comme objectif une description flexible de l'architecture. La capacité du langage VHDL peut supporter de telles descriptions et qui peut s'adapter à plusieurs spécifications. Le VHDL est riche par ces clefs de conceptions, nous pouvons citer : l'encapsulation, l'héritage, la généricité et la réutilisation à l'intérieur d'une description.



Lors de la conception, nous avons constaté que l'encapsulation réduit le nombre de détails que le concepteur doit confronter, cette ambiguïté est dépassée par la représentation d'une conception comme un ensemble d'unités de conceptions itératives. En plus, le concepteur ici n'a pas à savoir comment ces unités fonctionnent à l'intérieur mais, il doit concentrer ces efforts plutôt sur l'utilisation des paquetages, les appels de fonctions, les procédures et la déclaration des entités, ceci est défini comme l'encapsulation dans le langage VHDL.

Comme pour la conception Software, nous pouvons choisir un environnement parmi les seize (16) inconnus pour que le robot peut naviguer sans craintes ou risques de collisions, ainsi qu'on peut donner d'autres environnements pour effectuer la mission demandée en dehors de seize proposés pour la navigation.

Cette flexibilité offre une richesse à notre circuit, notre circuit FPGA n'est pas dédié uniquement pour les seize proposés pour la navigation, l'extension du nombre est encore plus large et qui peut être infinie (mais en tenant compte de l'ordre de complexité). Nous pouvons appliquer donc notre description à un autre environnement de paramètres différents. Une vraie tâche intelligente ainsi obtenue est embarquée alors par les véhicules pour réaliser les perspectives de l'IA et répondre encore plus aux critères de l'autonomie.

A cette flexibilité, le contrôle du robot est alors obtenu avec moins de consommations en *puissance énergie*. Notre circuit FPGA est rapide, il peut présenter certains avantages par rapport au développement en software tels : délais, tops d'horloges et d'autres paramètres qui sont optimisés par ce type de circuit.

B.3.2 Description physique de l'architecture

Jusqu'à présent l'architecture est décrite uniquement d'une manière comportementale (un niveau abstrait). Mais si nous voulons décrire cette architecture d'une manière physique ou structurelle, nous devons passer par l'outil de synthèse. A ce stade la on réalise le mapping du VHDL dans le circuit FPGA. Comme il a été déjà expliqué dans la section (3.B.5), La synthèse donne la description physique d'une architecture décrite en VHDL. L'outil de la synthèse que nous avons utilisé est Galileo, cet outil nous offre les avantages suivants :

- Réduction de la surface du silicium.
- Vitesse performante.
- Un délai du temps convenable.

Notons que la synthèse de notre architecture se fait de façon hiérarchique. Cela veut dire qu'on effectue la synthèse de sous-composants (modèles utilisés lors de la conception comme : multiplexeur, mémoire,...) avant de faire la synthèse globale de l'architecture. La façon hiérarchique offre une meilleure synthèse d'une architecture décrite en VHDL.

B.4. RESULTAT DE LA SYNTHESE

Durant cette phase, les différentes descriptions VHDL sont synthétisées par l'outil de synthèse Galileo, ce dernier peut estimer la surface des circuits en terme de bloc logique configurable (CLB). Pour notre part, nous avons choisi de cibler la famille XC4000 car elle permet d'intégrer une haute densité d'intégration et elle mieux convenable à la complexité de la circuiterie. Le tableau da la figure 4.6 présente les résultats de la synthèse en terme de CLB, ceci indique que l'architecture a été synthétisée. Ces résultats montrent que l'architecture existe sous forme d'une matrice (bloc logique) et elle peut être utilisée par l'outil XACT pour faire le placement et routage (c'est comme un circuit imprimé ou` on veut placer et implémenter quelques composants électroniques). Le circuit FPGA obtenu (après le placement et le routage de l'outil XACT) est décrit dans la figure 4.7.

| <i>Design</i> | Estimation en terme de CLB |
|---------------|----------------------------|
| <i>AG_LF</i> | 184 |

Figure 4.6 : Les résultats de la synthèse.

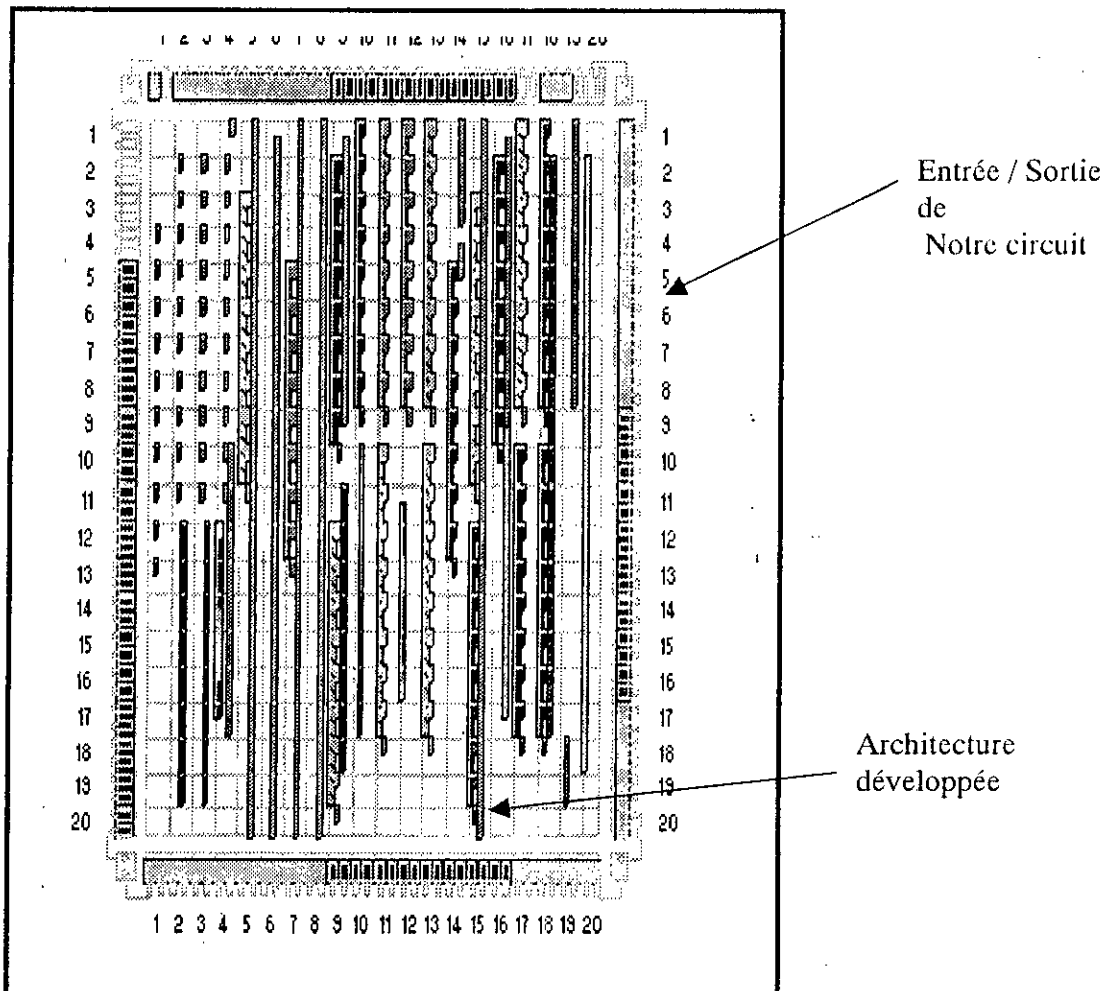


Figure 4.7 : Le circuit FPGA obtenu.

4. CONCLUSION

Dans ce chapitre, nous avons proposé les outils et les étapes utilisés pour la simulation. Les résultats de la simulation illustrent les capacités intelligentes ainsi que les comportements logiques optimaux quand le robot est ordonné à rejoindre son objectif en trouvant le chemin le plus optimal pour accomplir la mission désirée.

Les techniques utilisées (la logique floue et les algorithmes génétiques) s'avèrent des solutions pertinentes pour délivrer le chemin le plus adéquat à notre robot pour rejoindre son objectif. L'émergence de telles techniques offre un système robuste et rigoureux pour répondre à une mission donnée (navigation).

Le réflexe obtenu montre que le véhicule répond parfaitement aux exigences d'autonomie. Ce système se termine par l'exécution des mouvements intelligents capable de fournir aux véhicules plus d'autonomie et plus d'intelligence, notamment, l'adaptation à des environnements inconnus parsemés de plusieurs obstacles sans connaissance de tailles et de dispositions différentes. Ce réflexe s'expose par l'arrivée avec succès en se dirigeant vers la cible.

Pour répondre encore plus aux exigences d'autonomie et pour concrétiser la discipline de la micro-robotique : le royaume de la technologie de demain dans plusieurs domaines et utilisations, nous avons proposé une nouvelle méthodologie pour l'implémentation de la stratégie de navigation d'un robot mobile sur une carte électronique dotée de circuit FPGA basée sur la synthèse de haut niveau et en utilisant le langage de description VHDL. Comme l'originalité de notre approche illustre des algorithmes complexes et ait en besoin un temps énorme pour exécuter la mission désirée : l'utilisation d'un outil de synthèse permet de réduire considérablement ces contraintes et offre une facilité de mise en œuvre de la fonctionnalité.

Nous avons proposé une nouvelle méthodologie de conception : implémentation d'une stratégie de navigation sur le circuit FPGA basée sur la synthèse de haut niveau comme Galileo et en utilisant le langage de description hardware comme VHDL. Nous avons proposé une architecture régulière et intéressante intégrant un haut degré de parallélisme. Une telle nouvelle approche décrite en VHDL est flexible et paramétrique pour des simplifications, généralisations et changement en admettant ainsi toute modification ou changement donné par l'utilisateur de cette architecture.

Le circuit a été synthétisé en ciblant la famille FPGA XC4000, cette synthèse permettant d'optimiser la surface du circuit est justifiée par un choix judicieux offert par les caractéristiques de la compagnie Xilinx, cet outil qui offre une performance en temps et en vitesse la surface du silicium et qui permet ainsi en une implémentation complète sur un seul circuit FPGA.

CONCLUSION GENERALE

Le travail que nous avons réalisé s'inscrit à la fois dans le cadre d'un projet de recherche du Laboratoire de Robotique et d'Intelligence d'Artificielle et dans le cadre du laboratoire micro-électronique du centre de développement des technologies avancées CDTA à EL-Madania. Le but essentiel de ce présent travail se rapporte à la conception d'un planificateur local pour mener le robot d'un point de départ vers un point d'arrivée dans des environnements inconnus. Ce travail consiste, d'une part, en le développement d'un outil informatique (progiciel) permettant d'offrir une stratégie de navigation des véhicules autonomes, et d'autre part, en le développement d'un outil hardware (circuit FPGA) afin de répondre encore plus aux exigences d'autonomies et de gagner encore plus le temps avec le moins d'énergie pour accomplir la mission demandée au robot.

Nous nous sommes intéressé à une stratégie développée à partir d'étude et d'observation des comportements de l'être humain. Le comportement intelligent réside en reconnaissance des situations topologiques comme l'évitement des obstacles et en prise de décision sur la tâche à accomplir. Pour acquérir ces comportements intelligents, nous avons proposée des systèmes hybrides intelligents (SHI) exploitant les techniques suivantes :

- Les algorithmes génétiques.
- La logique floue .
- Les technique d'intelligence artificielle.

Les techniques citées ci- dessus s'avèrent des approches importantes pour le problème de la navigation dans des environnements inconnus. En effet, la robustesse de telles techniques peut être expliquée lorsque le robot peut se

retrouver parfaitement lorsqu'on propose d'autres environnements différents de ceux proposés pour la navigation. Notre système intelligent offre une flexibilité et une adaptation majeure dans la mesure que le robot peut accomplir la mission demandée dans d'autres scènes de navigation inconnues et différentes. Cette tâche est démarquée par une opportunité inhérente dans le vaste champ de la robotique mobile.

L'originalité de cette approche réside en la combinaison hybride de telles techniques citées ci-dessus. Les résultats que nous avons obtenu sont très satisfaisants et significatifs. Plusieurs applications et tests ont été testés avec succès.

Pour répondre encore plus aux exigences d'autonomie, et pour offrir un comportement pertinent, pour que le SHI soit capable de doter le véhicule de capacités de traitement en temps réel, rehaussant ainsi son « *opérability* » et sécurité avec une rapidité accrue : une nouvelle méthodologie s'est alors posée pour concevoir un système dédié à l'implémentation de système AG_LF sur une carte électronique dotée de circuit FPGA.

L'originalité de cette approche réside en la possibilité d'implémenter un algorithme de navigation complexe sur un seul circuit FPGA, une approche descendante de conception basée sur la synthèse de haut niveau s'est alors mise en œuvre. De plus en fixant comme objectif l'optimisation de la surface, nous avons réussi, grâce à un choix judicieux de composant à implémenter un algorithme entièrement sur un seul circuit. Cette nouvelle proposition d'une description VHDL permet d'offrir une flexibilité et reprogrammation possible, une grande densité d'intégration, haute performance et une grande rapidité avec une vitesse pertinente.

Les résultats auxquels nous avons aboutis sont très satisfaisants et significatifs. Plusieurs applications et tests ont été réalisées dans des environnements complètement inconnus, le chemin est retrouvé parfaitement et la mission est accomplie pour rejoindre l'objectif. Ces résultats offrent un traitement en temps réel de l'architecture qui peut être réutilisée pour couvrir un large domaine d'applications.

Ce projet nous a été d'un grand intérêt. Il nous a permis de nous introduire dans deux domaines importants sur l'échelle économique et technologique : celui de la robotique mobile et celui de la micro-électronique. C'était ainsi d'une opportunité importante pour bien manipuler des langages de programmation puissants destinés à la fois pour une conception software et hardware.

En perspective d'une éventuelle implémentation de navigation proposée sur « l'Autonomous Guided Véhicule (AGV) », nous espérons que notre travail soit un parchemin qui ouvrira de nouvelles perspectives et de nouveaux horizons à d'autres qui aimeront poursuivre cette voie en développant :

- 1- Une nouvelle stratégie de navigation dans le cas de cibles dynamiques
- 2- Un ensemble de véhicules se déplacent pour atteindre des cibles différentes ou une seule (statique ou en mouvement).

- 3- Il serait intéressant ainsi de baptiser les techniques précédentes sur les réseaux de neurones entraînés par les : algorithmes génétiques et la logique floue.
- 4- Il était aussi intéressant de développer une extension de *AG_LF* pour le cas des obstacles en mouvement. Ou` l'application peut être accomplie efficacement non limitée par l'ordre de complexité et qui s'adaptent parfaitement aux changements effectués
- 5- Il nous a été important de baptiser la génétique qui joue avec l'effet aléatoire et probabiliste sur l'outil mathématique statistique robuste : les chaînes de Markov cachées HMM, pour déterminer les candidats les plus probables possibles pour l'opération génétique avec un traitement plus robuste et qui offre une meilleure sélection des chromosomes et des populations cachées parmi les solutions proposées.
- 6- Une alternative aussi intéressante est d'implémenter les techniques citées ci dessus sur le circuit FPGA afin de gagner encore plus de rapidité et d'avoir mieux les pertinences

Le développement de telles approches de planification et de contrôle de la navigation ouvre une voie vers l'exploitation d'un nouveau concept « *micro-robot* ». Le fait de minimiser les cartes électroniques par de petits circuits peut minimiser l'architecture de notre robot. Ainsi on peut développer un système de taille ordinaire qui peut exécuter efficacement des tâches intelligentes dans des environnements réels.

BIBLIOGRAPHIE

- [1] Perebaskine , V.O : *Une architecture modulaire pour le contrôle d'un robot mobile autonome*, thèse de Doctorat, Spécialité : Robotique, laboratoire du CNRS, avril, 1992.
- [2] Losrios, J.A.S.D: *étude de déplacements d'un robot mobile dans un environnement peu contraint*, thèse de Doctorat, L'université de technologie De Comegne, option : contrôle des systèmes, septembre, 1993.
- [3] Korein, J.U. , Ish- Shalom, J. : *Robotics*, IBM Systems Journal, Vol.26, N.1,1987, pp.55-95.
- [4] Daclin , E. : *robots mobiles autonomes*, Techniques de l'ingénieur, Vol.R, Traité Mesures et Contrôle, pp.R7850 –1 à R7850-20
- [5] Chohra, A. : *Planification et contrôle de la navigation des véhicules autonomes intelligents (VAI) en environnement dynamiques*, Thèse de doctorat d'état en électronique, Magister Cybernétique, option : Robotique, 16 Mars, 1999.
- [6] Gleizes, M.P. and Glize, P.: *Les systèmes multi-experts*, Edition – Hermès, Paris,1990.
- [7] Bennila, N.: *Contribution au développement d'un robot mobile autonome et simulation d'un planificateur réactif des trajectoires*, Mémoire de Magister, Spécialité : électronique, option :Communication, Université de Blida,1995.
- [8] Hewett, M. and Hewett , R. : *A language and Architecture for Efficient*

| | | |
|---------------------|--|-----------|
| A.4.1 | Avantages des circuits ASICs. | 37 |
| A.4.2 | Inconvénients. | 38 |
| A.5 | Cycle de conception d'un ASIC. | 38 |
| A.5.1 | Faisabilité. | 38 |
| A.5.2 | Cahier des charges. | 38 |
| B | CIRCUITS FPGA. | 39 |
| B.1 | Introduction. | 39 |
| B.2 | Définition. | 40 |
| B.3. | Circuits FPGAs de la famille Xilinx. | 40 |
| B.3.1 | Famille de FPGA. | 41 |
| B.3.2 | Ressources des interconnexions. | 42 |
| B.4 | Langage de description hardware VHSIC. | 44 |
| B.4.1 | Définition. | 45 |
| B.4.2 | Nécessité du VHDL. | 45 |
| B.4.3 | Organisation pour une standardisation. | 45 |
| B.5 | Synthèse. | 46 |
| B.5.1 | Définition. | 46 |
| B.5.2 | Les outils de synthèse. | 46 |
| B.5.3 | Synthèse avec le VHDL. | 47 |
| B.6 | Approche de conception. | 47 |
| B.7 | Approche développée. | 48 |
| B.8 | Description digitale de l'architecture développée | 49 |
| B.9 | Architecture de composant AG. | 49 |
| B.10 | Architecture de composant LF. | 50 |
| B.11 | Paramètres de flexibilité | 51 |
| B.11.1 | Généricité. | 51 |
| B.11.2 | Instanciation. | 52 |
| B.12 | Description de l'architecture développée et quelques composants utilisés en VHDL. | 53 |
| B.12.1 | Exemple d'un comparateur . | 53 |
| B.12.2 | Exemple d'un additionneur . | 54 |
| 3 | Conclusion. | 59 |
| CHAPITRE 4 : | SIMULATION | 60 |
| A | CONCEPTION SOFTWARE | 60 |
| 4.A.1 | Présentation de progiciel AG_LF. | 61 |
| 4.A.1.1 | L'organigramme de travail. | 61 |
| 4.A.2 | Progiciel et résultats de la simulation. | 65 |
| B | CONCEPTION HARDWARE | 70 |
| 4.B.1 | Problème posé. | 70 |
| 4.B.2 | Objectifs de la conception. | 71 |
| 4.B.3 | Caractéristiques de l'architecture développée. | 71 |
| 4.B.3.1 | Description flexibilité. | 71 |
| 4.B.3.2 | Description physique de l'architecture. | 72 |
| 4.B.4 | Résultat de la synthèse. | 73 |
| 4 | Conclusion. | 74 |
| | CONCLUSION GENERALE | 75 |
| | BIBLIOGRAPHIE. | 78 |
| | ANNEXE PROCEDURES. | 83 |



Blackboard Systems, in Proc, 9.th.Conf.Artificial Intelligence for applications, Orlando, Florida, March 1-5, 1993, pp.34-40.

- [9] Skhiri, F. and Dafaoui, M. : *Navigation et localisation d'un robot mobile autonome*, Proc Int, AMSE Confer, Communication, Signal & Systems, Rabat (Morocco), October, 9-11, Vol.2, pp.968-977, 1995.
- [10] Bidlack, c. , Hampapur, A., katkere, A., Feng, l., Kagalwala et Kraljevic, T. : *Visual robot navigation using flat earth obstacle projection*, Proc, IEEE, international Conference on Robotics and Automation , Vol .4, May 8-13, 1994.
- [11] Gonzàlez, A. and Pérez, R. : *SLAVE : A genetic learning System Based on an Iterative Approach*, IEEE, Trans on Fuzzy Systems, Vol 7, N.2, April 1999, pp.176-191.
- [12] Murakawa, M., Yoshizawa, S., Kajitani, I., Yao, X., Kajihara, N., Iwata, M. and Higuchi, T. : *The GRD chip : genetic reconfiguration of DSPs for Neural Network Processing*, IEEE Trans on Computers, Vol.48, N.6, June 1999, pp.628-639.
- [13] Srinivas, M. and Patnaik, L.M. : *Genetic Algorithms : a survey*, 0018-9162/97/\$4.00© 1994 IEEE, Computer, pp.17-26.
- [14] Filho, J.L.R., Treleaven, P.C. and Alippi, C. : *Genetic – Algorithm Programming Environments*, 0018-9162/94/\$4.00© 1994 IEEE, pp.28-73.
- [15] Burdzy , K. and Biszka, J.B. : *the reproducibility property of fuzzy control systems*, Fuzzy Sets and Systems 9(1983), North Holland, pp.161-177.
- [16] Koczy, L.T et Hirota , K. : *Ordering distance and closeness of fuzzy*, Fuzzy Sets and Systems 59(1993), North Holland, pp.281-293.
- [17] Ruan, D. and Kerre, E.E. : *Fuzzy implication operators and generalised Fuzzy method of cases*, Fuzzy Sets and Systems, 54(1993), North Holland, pp.23-37.
- [18] Yager, R.R. : *A characterisation of the extension principle*, Fuzzy Sets and Systems 18 (1986), 205-217.
- [19] Parris, C.p. & haggard, R.L. : *An architecture for a high speed Fuzzy logic Inference Engine in FPGAs* : 0-8186-7873-9/97\$10.00© IEEE, 1997, pp.179-182.
- [20] Cao, Z. and Kandel, A. : *Applicability of some fuzzy implication operators*, Fuzzy Sets and Systems 31(1989), North Holland, pp.151-186.
- [21] Masmoudi, N., Hachicha, M. and Kamouni, L. : *Hardware design of programmable fuzzy controller on FPPGA*, in Proc.Int IEEE Conf, Fuzzy Systems, August 22-25, 1999, Seoul, Korea, ppS.III-1675-III-1679.
- [22] Lago, E., Jiménez, C.J., Lopez, D.R, Sanchez-Solano, S. et Barriga, A. :

- XFVHDL : A tool for synthesis of fuzzy Logic controllers*, IEEE, 0-8186-8359-7/98 \$10.00©, 1998, pp.102-107.
- [23] Kawase, S. and Yanagihara, N. : *On the accuracy of fuzzy models of infinite dimensional system*, proceeding of the international conference on fuzzy logic, Neural Networks (Izuka, Japan, July 20-24, 1990), pp.523-526.
- [24] Norwich, A.M. et Turksen, I.B. : *The fundamental measurement of fuzziness*, Fuzzy Set and Possibility theory, pp.49-60.
- [25] Norwich, A.M. et Turksen, I.B.: *The construction of fuzziness system*, Fuzzy Set and Possibility theory, pp.61-67.
- [26] Norwich, A.M. et Turksen, I.B., *The meaningful of fuzzy system*, Fuzzy Set and Possibility theory, pp.68-74.
- [27] Pedrycz, W.: *Construction of fuzzy relational models*, *Mathematics and Systems Research 2*, R. Trappl (ed.), Elsevier Science Publishers B.V, (North- Holland), 1984, pp.545-549.
- [28] Pedrycz, W.: *Application of fuzzy relational equations for methods of reasoning in presence of fuzzy data*, *Fuzzy Sets Systems 16* (1985), North – Holland, pp.163-175.
- [29] Kiszka, J.B., Kochanska, M.E. et Sliwinska, D.S. : *the influence of some fuzzy implication operators on the accuracy of a Fuzzy Model*, Part 1, *Fuzzy sets and Systems 15* (1985), North Holland, pp.111-128.
- [30] Bandler, W. and KOHOUT, L. : *Fuzzy power sets and fuzzy implication operators*, *Fuzzy Sets and Systems 4* (1980), North Holland, pp.13-30.
- [31] Shih-Sen, C. : *Fixed point theorems for fuzzy mapping*, *Fuzzy Sets and Systems 17* (1985), 181-187.
- [32] Dinechin, F. : *The price of routing in FPGAs*, rapport de recherche, Septembre 1999, N.3772, pp.1-19.
- [33] Brown, s. , Rose, J. : *FPGA and CPLD architectures : A tutorial*, IEEE Design & Test of Computers, 1996, pp.42-97.
- [34] Lucazeau, B, Trézéguet, H. : *ASIC : Circuits intégrés pour applications spécifiques*, *Techniques de l'ingénieur, traité électronique, Vol.E*, pp. E 2418-1 à E 2419-4.
- [35] Cummings, M. and Haruyama, S. : *FPGA in the Soft radio*, IEEE Communication Magazine, Communication Systems and device , February 1999, Vol.37, N.2, pp.108-112.
- [36] Lesea, A. : *Many ASICs can easily be replaced with FPGA*, [http : // www.Xilinx.com](http://www.Xilinx.com).
- [37] Camilleri, N. & Lockhard, C. : *Improving XC4000 Design performance*,

- XCell Journal ,Xilinx , XAPP, 043.000, pp.8-21 à 8 –35.
- [38] Σ Xilinx, *XC4000E and XC4000X series FPGA*, Xcell Journal , Xilinx, May 14, version 1.6, 1999, pp.6-5 à 6-10.
- [39] Cartier, L.: *Using select Ram memory in XC4000 series FPGAs*, XCell Journal , Xilinx, XAPP 057 July 7, version 1.0, 1996, pp.1-16.
- [40] Cartier, L.: *Implementing FIFO in XC 4000 series RAM*, Xcell Journal, Xilinx, XAPP 053, July, version 1.1,1996,pp.1-16.
- [41] Brown, S.D., Francis , R.J., Rose, J. and Vranesic, Z.G. : *Field-Programmable Gate Array*, Kluwer Academic Publishers, 1997
- [42] Mintzer, L. : *The FPGA as FFT processor*, [Http://www. ICSPAT. com](http://www.ICSPAT.com).
- [43] Trimberger, m. : *Field Programmable Gate Array*, Kluwer Academic Publishers, 1995.
- [44] Airiau, R., Bergé, J.M., Olive, V., Rouillard,J. : *VHDL du langage à la modélisation*, presse Polytechniques et Universitaires Romandes et CNET- ENST.
- [45] Mazo, s. et Langstrat, P. : *A guide to VHDL* , Kluwer Academic Publishers, 1997.
- [46] Airiau, R., Bergé, J.M., Olive, V. : *Circuit synthesis with VHDL*, Kluwer Academic Publishers, 1994.
- [47] Pellerin, D. and Taylor, D. : *VHDL made easy!* Prentice Hall PTR, 1997.
- [48] Legenhausen, J., Wade, R., Wilner, C. and Wilson, B. : *VHDL for programmable logic*, Addison- Wesley, 1996.
- [49] GALILEO, *HDL Synthesis Manual*, Exemplar Logic,1995.
- [50] *XACT user manual*, © 1995.
- [51] Izeboudjen, N. : *Conception et implémentation en FPGA d'un classificateur neuronal des arythmies cardiaques*, Mémoire de Magister, spécialité : électronique, option : télécommunications, école polytechnique,1999.
- [52] Keirse, D.M., Koch, E., Mckisson, J., Meystel, A.M. and Mitchell, J.S.B. : *Algorithm for Navigation for a Mobile Robot*, CH2,1984,IEEE,pp. 574-583.
- [53] Delannoy, C. : *Apprendre à programmer en C*, Éd. Eyrolles, 1992.
- [54] Delannoy, C. : *Exercices en langage C* , Éd. Eyrolles, 1992.
- [55] LEVINE, A. : *Mathématiques en turbo C : Initiation*, Edition Berti, 1993.

- [56] The programmable logic DATA Book, Xilinx, 1994.
VHDL and Verilog Simulation for work stations , V- system/ plus work station, User's Manual. Version 4.6.