Utilisation des réseaux de neurones artificiels pour la commande d'un véhicule autonome

Eric Gauthier

THÈSE

présentée par

Eric GAUTHIER

pour obtenir le grade de DOCTEUR

de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(Arrêté ministériel du 30 mars 1992)

(Spécialité : Informatique, Systèmes et Communications)

==========

Utilisation des Réseaux de Neurones Artificiels pour la Commande d'un Véhicule Autonome

==========

Date de soutenance : 25 Janvier 1999

Composition du jury :

Président :	**Jean-Luc Schwartz**
Rapporteurs :	**Frédéric Alexandre** **René Zapata**
Examinateurs :	**Bernard Amy** **Philippe Jorrand** **Christian Laugier**

Thèse préparée au sein de l'équipe RÉSEAUX D'AUTOMATES du Laboratoire LEIBNIZ en collaboration avec le projet SHARP de l'INRIA Rhône-Alpes.

Remerciements

Je tiens à remercier en premier lieu les personnes qui ont accepté de juger cette thèse et d'y apporter leur caution :

- M. Jean-Luc Schwartz, Directeur de Recherche CNRS qui m'a fait l'honneur de présider mon jury ;
- M. René Zapata, Maître de Conférence à l'université de Montpellier et M. Frédéric Alexandre, Chargé de Recherche INRIA pour avoir accepté d'être rapporteurs de ma thèse ainsi que pour leurs remarques constructives ;
- M. Philippe Jorrand, Directeur de Recherche CNRS pour m'avoir acceuilli dans son laboratoire ;
- M. Bernard Amy, Ingénieur de Recherche CNRS et M. Christian Laugier Directeur de Recherche INRIA pour m'avoir encadré durant mon travail. Je tiens à les remercier pour leur aide et pour le temps qu'ils m'ont consacré, mais aussi pour la grande liberté qu'ils m'ont laissée durant ma thèse.

Ce travail n'aurait put être mené à bout sans l'aide et la présence à mes cotés de tous les membres des deux équipes de recherche au sein desquelles j'ai passé ces trois années.

Je tiens à remercier en particulier parmi eux Philippe Garnier, Abderahim Labbi et Cyril Novales pour leur encadrement et leurs précieux conseils, sans oublier Olivier Dulac et Thierry Fraichard qui se sont portés volontaires pour la tâche ingrate de relecture de mon mémoire.

Il me faut également remercier les membres des moyens robotiques de L'INRIA Rhône-Alpes pour l'aide qu'ils m'ont apportée dans mes tentatives desespérées pour faire bouger le robot.

Enfin, je voudrais bien sur remercier mes parents et ma famille pour leur aide et leur soutient sans lesquels ce travail n'aurait pu être réalisé.

Table des matières

Introduction générale

Problématique

La conception de robots mobiles, capables d'évoluer avec un minimum d'interventions extérieures, a été le but de nombreuses recherches menées depuis le début des années 80. Les travaux réalisés ont porté dans un premier temps sur des robots évoluant dans des environnements fortement structurés et parfaitement connus à l'avance. Ils ont permis de produire des systèmes limités à une évolution sur des voies dédiées, libres de toute présence humaine, que ce soit en milieu industriel ou dans des environnements hostiles (e.g. centrale nucléaire). Les robots qui y sont utilisés sont généralement spécialement conçus pour la tâche particulière qui leur est assignée.
L'automatisation d'un véhicule aussi complexe qu'une voiture évoluant à des vitesses importantes dans un environnement peu structuré ou changeant, potentiellement occupé par des êtres humains constitue un tout autre défi. Les applications aujourd'hui envisagées pour ce type de robot sont multiples. Il peut s'agir dans un premier temps d'une automatisation partielle ou totale du fonctionnement d'ensemble du robot dans des zones de parking dédiées, ou d'une aide à la réalisation de manœuvres particulières. Une conduite totalement autonome est également visée à plus long terme.
Les contraintes de sécurité et de robustesse inhérentes à ce type d'application imposent des capacités de perception, de décision et d'action particulièrement importantes de la part du véhicule, dont la trajectoire doit pouvoir être contrôlée de manière précise.
La taille et la cinématique particulière d'une voiture complexifient encore la commande d'un tel robot.

Les techniques basées sur l'emploi des réseaux de neurones artificiels suscitent aujourd'hui un intérêt croissant dans les domaines de la commande et de la robotique. La rapidité de traitement, les capacités d'apprentissage et d'adaptation, mais aussi la robustesse de ces approches motivent en grande partie ces travaux.
L'objectif de cette thèse, qui se situe à l'intersection des domaines de la robotique mobile et des réseaux de neurones artificiels, est d'étudier les solutions que peuvent apporter les techniques connexionnistes[1] aux problèmes particuliers posés par la commande automatique d'un robot de type voiture.

1. Techniques utilisant les réseaux de neurones artificiels

Notre contribution

La première contribution de cette thèse concerne l'établissement d'une double classification qui porte d'une part sur les différentes architectures de commande pour robots mobiles rencontrées dans la littérature, et d'autre part sur les méthodes de commande utilisant les réseaux de neurones artificiels (RNA).
La première de ces études nous permet de spécifier les différentes étapes qui doivent, à notre avis, être suivies pour construire un système de commande conférant une véritable autonomie à un robot de type voiture. La deuxième nous aide à définir les apports possibles des RNA dans un tel système.
Ceci nous permet de proposer plusieurs contrôleurs, utilisant les RNA, et autorisant un accroissement progressif de l'autonomie du véhicule.
Le premier d'entre eux s'applique au simple suivi d'une trajectoire de référence, issue d'un planificateur. Notre approche autorise une adaptation continue du système face à d'éventuels changements des paramètres du robot ou de son environnement.
Nous proposons également une méthodologie pour la réalisation de systèmes de commande, basés sur les informations délivrées par les capteurs du robot, et ayant pour but la réalisation de manœuvres spécifiques. Nous introduisons pour cela un modèle alliant des caractéristiques issues de la logique floue et des réseaux de neurones artificiels.
Enfin nous montrons comment des manœuvres plus complexes peuvent être réalisées à partir de l'enchaînement de plusieurs contrôleurs simples. Nous proposons d'utiliser un modèle de RNA récurrent introduit par l'équipe Réseaux d'Automates du laboratoire LEIBNIZ, pour réaliser le système de sélection de ces contrôleurs. Notre approche autorise des réactions très rapides face à l'ensemble des événements extérieurs qui doivent être pris en compte.

Organisation de ce mémoire

Ce mémoire est organisé en deux parties principales. La première d'entre elles aborde les aspects fondamentaux de la commande des robots mobiles, ainsi que de l'application des RNA dans le domaine de la commande. Elle se compose de deux chapitres :

- Dans le chapitre 1 nous exposons les principales architectures de commande pour robots mobiles rencontrées dans la littérature. Celles-ci sont séparées en trois grandes familles : l'approche délibérative, l'approche réactive et l'approche hybride. Nous présentons à la fin de ce chapitre l'architecture proposée par le projet SHARP de l'INRIA Rhône-Alpes, sur laquelle sont centrés nos travaux ;

- Dans le chapitre 2, après une brève présentation de certains modèles de RNA, ainsi que de leurs principales caractéristiques et propriétés, nous proposons une revue des schémas d'utilisation de ces modèles pour la commande.

La deuxième partie de ce rapport présente l'application de plusieurs modèles de RNA à la commande d'une voiture autonome. Elle comporte trois chapitres qui abordent la

réalisation de tâches de complexité croissante :

- Le chapitre 3 aborde le problème du suivi d'une trajectoire planifiée. Nous y présentons les approches courantes pour la résolution de ce problème, ainsi que le système que nous proposons ;
- Le chapitre 4 traite de la réalisation de manœuvres basées sur les capteurs du robot. Nous présentons ici aussi les approches courantes, ainsi que le modèle de RNA que nous proposons d'utiliser et les modalités de cette utilisation ;
- Le chapitre 5 expose le système que nous proposons pour enchaîner des manœuvres simples, afin de réaliser des missions plus complexes.

Mots-clés : Réseaux de neurones artificiels, robots mobiles, systèmes hybrides, logique floue, non-holonomie.

Première partie

La commande des robots mobiles et les réseaux de neurones artificiels : aspects fondamentaux

Introduction

Les travaux présentés dans ce mémoire ont pour but d'étudier les possibilités d'utilisation des réseaux de neurones artificiels dans le cadre de la commande d'un robot mobile de type voiture. Dans cette première partie nous tentons de présenter un double état de l'art qui concerne à la fois les architectures de commande pour robots mobiles rencontrées dans la littérature et les utilisations des réseaux neurones artificiels dans le domaine de la commande :

- le premier chapitre présente les différentes approches utilisées pour permettre à un robot de combiner des raisonnements de haut niveau avec une vitesse de réaction importante. Notre but est d'isoler les différents types de traitement impliqués dans une architecture de ce type, et ce afin de déterminer par la suite les points sur lesquels les RNA peuvent intervenir ;
- le deuxième chapitre expose les modalités d'application des réseaux de neurones artificiels dans le domaine de la commande. Nous ne nous limitons pas à des exemples liés à la robotique mobile. Notre but est plutôt de présenter une «boîte à outils», dont la deuxième partie de ce mémoire décrit l'utilisation.

Chapitre 1

Systèmes de commande pour robots mobiles

Au cours de ce chapitre, nous présentons les principales architectures de commande pour robots mobiles rencontrées dans la littérature. Il nous est impossible d'établir une liste exhaustive de ces travaux, c'est pourquoi nous nous limitons à des représentants significatifs des principales tendances. Ces travaux sont séparés historiquement en trois groupes dont les deux premiers se sont longtemps opposés :

- l'approche délibérative, d'une part, utilise une modélisation de l'environnement, connue *a priori* ou obtenue à partir des données capteurs, pour planifier à l'avance les commandes que le robot doit exécuter. Cette vision de la robotique autonome conduit à une décomposition séquentielle du traitement réalisé et à des systèmes fortement hiérarchiques ;
- l'approche réactive, d'autre part, s'appuie sur un couplage étroit entre les capteurs et les actionneurs, pour générer en continu les commandes. Cette méthode va généralement de pair avec une décomposition sous forme de comportements élémentaires réalisant, chacun en parallèle, une fonction simple ;
- l'approche hybride qui tente de combiner les deux approches précédentes afin de tirer partie des avantages respectifs de ces deux techniques. Ce dernier groupe représente aujourd'hui la grande majorité des systèmes étudiés.

1.1 Les systèmes purement délibératifs

Ces approches s'appuient sur les techniques de l'intelligence artificielle classique. Elles visent à reproduire le mode de raisonnement humain ou tout au moins une certaine vision du mode de raisonnement humain. Le traitement est décomposé en une série d'opérations successives décrites par la figure 1.1.

- la première opération consiste à traiter les données sensorielles qui fournissent au robot des informations sur son environnement ;

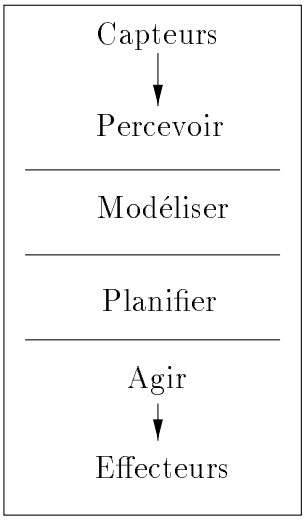

FIG. 1.1 – *L'architecture délibérative.*

- l'étape suivante consiste à construire ou à mettre à jour, à partir des données capteurs, un modèle du monde dans lequel évolue le robot. Ce modèle peut être par exemple une identification du type et de la position des obstacles que le robot doit éviter ;
- ce modèle est utilisé pour planifier une suite d'états conduisant le robot à effectuer la tâche recherchée ;
- la dernière opération a pour but de calculer puis d'exécuter les actions permettant au robot de suivre le plan généré par l'étape précédente.

La partie la plus importante, et la plus gourmande en calcul, est celle qui concerne la modélisation et la planification qui peuvent prendre un temps assez long, et ceci notamment en environnement peu structuré. Le temps disponible pour l'action est par conséquent beaucoup plus court. L'intervalle de temps entre les phases de perception et d'action peut de plus être relativement long.
On parle souvent pour cette approche de structure S.M.P.A. (pour Sense, Model, Plan, Act).
Les premières architectures de commande présentées dans la littérature sont basées sur un fonctionnement de ce type. C'est notamment le cas du premier exemple de robot mobile possédant une certaine autonomie : Shakey [114]. Ce robot, conçu au début des années 70 au Stanford Research Institute, utilisait une caméra T.V. comme capteur et était capable d'évoluer dans un environnement très fortement contraint. Le raisonnement de Shackey faisait intervenir une technique très proche de la démonstration automatique de théorème. Les tâches typiques sur lesquelles Shakey fut testé consistaient à trouver dans une pièce un objet de forme et de couleur fixées à l'avance, et à le pousser jusqu'à un point donné. Chaque mouvement simple demandait plus d'une heure de calcul sur un ordinateur extérieur et présentait une forte probabilité d'échec.

L'architecture du Stanford Cart développée à l'université de Carnegie Mellon [105] est également assez représentative des contrôleurs[1] délibératifs. Un système de vision 3D fournit au robot la position des objets dans son environnement. Le planificateur propose ensuite un chemin permettant au robot d'éviter les obstacles tout en se rendant à l'endroit voulu. Le robot se déplace alors le long de ce chemin sur une distance d'un mètre. Après ce mouvement, le cycle complet est exécuté de nouveau jusqu'à ce que le but soit atteint. Le traitement des images impose, lors de chaque déplacement d'un mètre, une attente de près d'un quart d'heure.
De manière générale les architectures purement délibératives posent un certain nombre de problèmes :

- leur principal inconvénient provient de leur faible vitesse de réaction. Elles sont en effet totalement incapables de prendre en compte des obstacles dynamiques ou non détectés lors de la modélisation. Ceci est dû à la lenteur des phases de modélisation et de planification qui ne peuvent généralement pas être exécutées en temps réel, même lorsqu'elles sont implantées sur de gros calculateurs extérieurs au robot ;

- la construction du modèle sur lequel se base la planification n'est pas une tâche aisée. Il est en pratique souvent difficile de relier les données sensorielles aux objets du monde réel. Ceci est dû en grande partie à l'imprécision de l'information délivrée par les capteurs utilisés en robotique. R. A. Brooks résume ce problème en affirmant qu'il existe une grande différence entre *capter* et *percevoir* ;

- les différences potentielles entre le modèle et le monde réel introduisent des incertitudes fortes sur la position du robot et des obstacles. La prise en compte de ces incertitudes, qui est indispensable pour obtenir un système robuste, rend la planification beaucoup plus complexe et pose de nombreux problèmes qui restent ouverts.

L'utilisation de ces approches semble donc limitée aux robots évoluant dans des environnements statiques et dont la structure est fortement contrainte et connue *a priori*.
Les systèmes délibératifs possèdent cependant un point fort important, qui est leur capacité à prendre en compte des raisonnements de haut niveau lors de la phase de planification. Il leur est possible en effet de gérer des missions complexes enchaînant plusieurs buts successifs. Ceci est comme nous le verrons dans la section suivante, impossible pour les architectures purement réactives actuelles.

1. Par abus de langage nous emploierons dans ce chapitre et dans le reste du mémoire le terme «contrôleur» comme équivalent de «système de commande». Bien que fréquemment rencontré dans la littérature francophone comme traduction de l'anglais «controller», ce terme devrait prendre un sens plus restreint en Français, et désigner plutôt un système ayant en charge la vérification ou la supervision [24].

1.2 Les systèmes purement réactifs

1.2.1 Principe

Cette approche, radicalement différente de la précédente se distingue par l'abandon des phases de modélisation et de planification. Elle est née à partir de considérations issues de l'éthologie et plus particulièrement de l'entomologie. Nous savons aujourd'hui que de nombreux êtres vivants, dont en particulier les insectes, ne possèdent, en raison de leur taille que très peu de capacités de modélisation et de raisonnement. Ils n'en sont pas moins capables de mener à bien des tâches complexes. Leur comportement semble même du point de vue d'un observateur extérieur résulter d'une intelligence avancée, similaire par certains côtés à la nôtre.

En partant de cette constatation, des chercheurs ont tenté de s'inspirer de cette autre vision de l'intelligence en proposant une architecture de commande nouvelle basée sur la composition de plusieurs modules implémentant des comportements simples[1]. Le comportement global du robot est alors le résultat de la composition des divers comportements élémentaires. Il *émerge* de l'interaction de ces derniers entre eux et avec l'environnement. Chacun des comportements de base, qui peuvent être par exemple l'évitement d'obstacle ou l'attirance pour un but, réalise un couplage étroit entre les capteurs et les effecteurs du robot. La simplicité du traitement réalisé, ainsi que la parallélisation des comportements, permettent une grande rapidité d'exécution.

On parle généralement de systèmes de commande réactifs pour qualifier ce type de fonctionnement. On peut toutefois distinguer deux définitions assez différentes de la réactivité. L'informaticien parlera de système réactif pour désigner un système *apte à réagir continûment à un environnement physique, à une vitesse déterminée par cet environnement* [53]. Dans le domaine des sciences cognitives, par contre, on dira d'un agent (i.e. robot) qu'il est réactif *s'il ne possède pas de représentation explicite du monde* [36]. Ces deux définitions se recoupent toutefois en ce qui concerne les systèmes de commande réactifs, puisque ceux-ci réagissent directement aux stimuli provenant du monde extérieur en évitant la construction d'un modèle forcement entaché d'erreurs en environnement réel, ce qui autorise des vitesses d'exécution importantes. Cette approche est justifiée par R. A. Brooks qui affirme que le meilleur modèle du monde est le monde lui-même.

Nous pouvons également noter que si l'on s'en tient à la première définition, l'opposition entre systèmes réactifs et systèmes délibératifs n'est plus aussi nette. Les systèmes délibératifs peuvent en effet être considérés comme réactifs dans une échelle de temps plus lente. Rien n'interdit de plus d'imaginer, la puissance des ordinateurs augmentant sans cesse, que des systèmes délibératifs beaucoup plus rapides pourront être réalisés dans le futur.

1. On parle d'architecture comportementale.

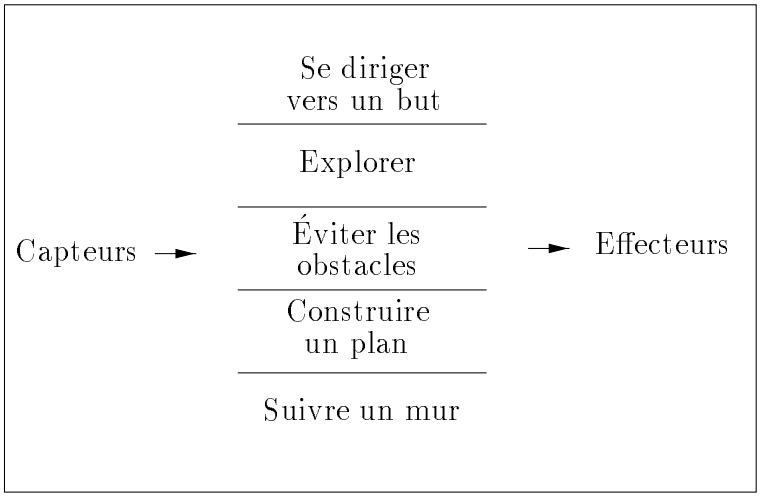

FIG. 1.2 – *Une architecture comportementale.*

1.2.2 L'architecture subsumption de Brooks

L'architecture proposée par Brooks [21, 22] est l'une des premières architectures comportementales (voir figure 1.3). C'est en tout cas la plus connue, et celle qui a donné naissance à de nombreux travaux.
Elle s'appuie sur une décomposition verticale en niveaux de compétences. Chacun de ces niveaux correspond à un comportement indépendant recevant des données capteurs et agissant sur les effecteurs. Ils sont réalisés par des machines d'états finis augmentées (prenant en compte le temps) communiquant entre elles par des messages. Chaque module est capable d'inhiber ceux de niveau inférieur au sien ou de les utiliser pour son propre traitement.
Cette architecture ne possède aucun arbitre central permettant de choisir l'un ou l'autre des comportements. Ce choix se fait par la réalisation des différents niveaux et le précâblage des communications possibles entre eux.

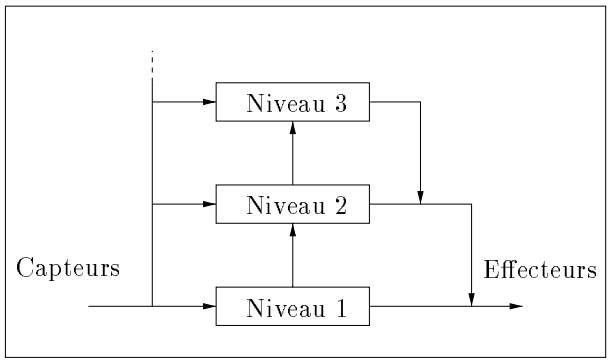

FIG. 1.3 – *La hiérarchie de niveau dans l'architecture Subsumption.*

Brooks définit, dans le cas d'un robot d'exploration, 8 niveaux de compétences qui sont

dans l'ordre :

0 Éviter les obstacles fixes ou mobiles.

1 Errer sans but.

2 Explorer le monde en cherchant les zones libres.

3 Construire une carte de l'environnement et des chemins possibles.

4 Noter les changements dans l'environnement.

5 Raisonner sur le monde en termes d'objets identifiables et de réalisations de tâches relatives à certains objets.

6 Formuler et exécuter un plan ayant pour but le changement de l'état du monde.

7 Raisonner sur le comportement des objets dans le monde et modifier les plans en conséquence.

Cette architecture a été implantée sur plusieurs robots du M.I.T. dont certains sont présentés sur la figure 1.4. Seuls les trois premiers niveaux de compétences ont cependant étés réalisés sur ces robots[1].

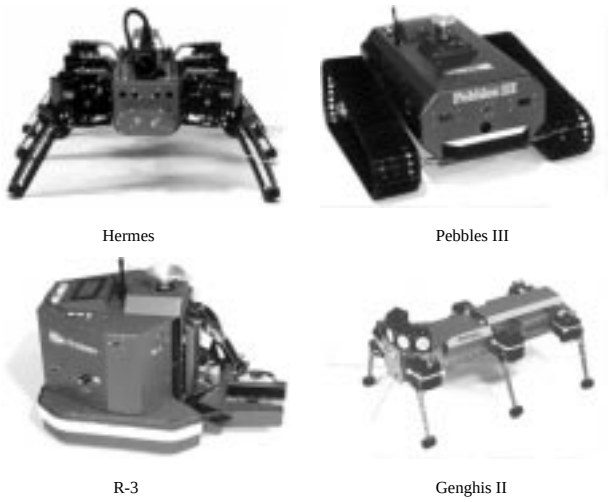

FIG. 1.4 – *Robots utilisant l'architecture subsumption (photos IRS).*

Cette architecture possède plusieurs points forts notables :

– sa rapidité de réponse qui lui donne la capacité d'évoluer dans des environnements fortement dynamiques. La simplicité des comportements permet d'accroître encore cette vitesse par une réalisation sur circuit imprimé ;

1. Nous pouvons noter que la réalisation des niveaux supérieurs, qui incluent les notions de modélisation et planification, classerait de fait cette architecture dans les systèmes hybrides.

- sa robustesse qui découle de la parallélisation et de l'indépendance relative des comportements qui permet au robot de continuer à fonctionner en cas de panne de l'un d'entre eux ;

- son caractère incrémental qui provient de la hiérarchie de comportements permet de rajouter aisément des niveaux supérieurs au système ;

- sa simplicité implique un faible coût de production et des possibilités de miniaturisation importantes. Ceci ouvre de nouveaux domaines d'application. Brooks s'intéresse notamment à l'utilisation d'un grand nombre de petits robots utilisant l'architecture subsumption dans le domaine spatial [23].

Le principal reproche fait à l'encontre de l'architecture subsumption de Brooks (et plus généralement des architectures purement réactives) vient de son manque de capacités de raisonnement de haut niveau. Même si celles-ci sont théoriquement présentes dans les niveaux de compétence les plus hauts, ces derniers n'ont jamais été implantés sur un robot, et on peut s'interroger sur les possibilités de raisonner aisément à haut niveau avec des machines d'états finis.
Les robots utilisant l'architecture subsumption réalisés par Brooks, n'ont de plus jamais été testés de manière poussée en environnement extérieur (hors d'un laboratoire spécialement aménagé) alors que la possibilité d'évoluer dans des milieux totalement inconnus et non contraints est une des justifications principales de l'absence de modélisation.

1.2.3 Autres architectures réactives

L'approche comportementale a inspiré plusieurs travaux concernant aussi bien la réalisation des comportements réactifs, que la mise en place d'architectures globales permettant leur composition. Nous nous intéressons ici à la deuxième catégorie de travaux, mais nous présenterons dans la section 4.2.3 ceux qui concernent l'utilisation de réseaux de neurones artificiels pour la réalisation de comportements par apprentissage.
L'architecture subsumption utilise une hiérarchie de comportements, dans laquelle à un moment donné un seul module dirige le robot (en utilisant éventuellement le résultat des autres comportements).
Une autre approche possible consiste à composer les commandes préconisées par plusieurs comportements. C'est notamment le principe des travaux de T. L. Anderson et M. Donath [6]. Chaque module de l'architecture qu'ils proposent représente un comportement réflexe simple ne possédant aucune capacité de mémorisation. La définition de tels comportements est donnée par D. McFarland [99] : *Un comportement réflexe est la forme la plus simple de réaction à une stimulation externe. Des stimuli tels qu'un changement soudain dans le niveau d'illumination, ou un contact à un endroit donné du corps, génèrent une réponse automatique, involontaire et stéréotypée.* Ces modules génèrent des champs de potentiels (voir §4.2.1) dont la composition permet de calculer la direction à suivre par le robot (en suivant le gradient de la fonction de potentiel

évalué pour la position actuelle du robot). Un mécanisme permet de plus de sélectionner les comportements à prendre en compte en fonction de la tâche à accomplir ou de l'environnement actuel du robot.
D. W. Payton et J. K. Rosenblatt [141, 127] proposent d'utiliser un réseau de neurones artificiels (voir chapitre 2) pour fusionner les commandes préconisées par les différents comportements. Le système est cependant construit «à la main» de manière totalement empirique.

Certains auteurs ont également cherché à réaliser par apprentissage un système de sélection des différents comportements. Ces travaux font généralement intervenir une forme d'apprentissage appelé apprentissage par renforcement que nous présenterons dans la section 2.5.3. Dans le système proposé par M. Humphrys [66] par exemple, chaque module comportemental apprend et met à jour une fonction donnant une évaluation de l'importance de l'action qu'il propose dans l'état courant. Le comportement présentant l'action de plus grande importance est sélectionné pour commander le robot.
L. J. Lin [95] utilise un système similaire dans lequel il construit par apprentissage une fonction donnant le comportement le plus adapté à l'état courant. Ces deux travaux se basent sur l'algorithme du Q-Learning [163] sur lequel nous reviendrons dans la section 2.5.3.2.
De manière générale, on retrouve dans ces différents travaux les points faibles (i.e. manque de capacités de raisonnement de haut niveau) et les points forts (i.e. rapidité de réaction, robustesse, modularité) de l'architecture de Brooks. Ces caractéristiques sont communes à l'ensemble des architectures purement réactives.

1.3 Les approches hybrides

Les approches purement réactives ou purement délibératives représentent deux extrêmes que de nombreux auteurs ont naturellement tenté de combiner. Le but de leur démarche est de conserver les capacités de raisonnement de haut niveau des approches délibératives, tout en s'assurant la robustesse et la rapidité d'utilisation des approches réactives. On parle généralement *d'architectures hybrides* pour décrire ces travaux.
Cette notion est à distinguer de celle de système hybride en intelligence artificielle (ou système hybride neuro-symbolique), qui désigne un système alliant des composantes utilisant des techniques issues de l'intelligence artificielle symbolique classique et des réseaux de neurones artificiels.
Nous verrons toutefois plus loin que ces deux concepts peuvent être rapprochés puisque les réseaux de neurones artificiels sont, à notre avis, plus adaptés à la réalisation de systèmes réactifs, alors que la planification fait plutôt intervenir des processus symboliques.
Les solutions adoptées pour combiner réactivité et délibération sont assez variées. Le choix d'un critère permettant de classifier les architectures de commande hybrides est donc relativement délicat. Nous avons choisi de séparer ces travaux en trois classes

principales [1] :

- la première correspond à des systèmes délibératifs, sur les couches basses desquels ont été greffés des mécanismes réactifs ;
- la deuxième regroupe au contraire, des architectures pour lesquelles l'aspect réactif est prédominant ;
- la dernière classe présente les travaux les plus récents qui s'orientent vers des architectures à trois niveaux incluant une «interface» entre les composantes délibératives et réactives du système.

1.3.1 Architectures à forte composante délibérative

Nous commençons notre présentation par les systèmes qui se basent sur des architectures délibératives classiques, ou plus généralement pour lesquels la partie délibérative est prédominante.

1.3.1.1 L'architecture de commande du robot d'exploration du JPL

Ce système proposé par E. Gat [44] a pour but de donner une autonomie partielle au robot. Il a été conçu pour permettre l'évolution de véhicules d'exploration sur d'autres planètes, le temps mis par les communications avec la terre interdisant dans ce cas une commande en temps réel par un opérateur. Cette architecture décrite par la figure 1.5 se compose d'un système de perception, d'un planificateur de chemins, d'un planificateur de contrôle d'exécution et d'un système de vérification de l'exécution du plan.
Le système de perception a pour buts de construire une carte locale de l'environnement et de faire le lien avec des données globales obtenues en orbite. En se basant sur cette carte, le planificateur de chemins propose un déplacement d'une dizaine de mètres. Un simulateur du véhicule permet de prévoir les valeurs attendues pour les capteurs le long de ce chemin. Le planificateur de contrôle d'exécution utilise ces prédictions pour spécifier les capteurs à surveiller lors du mouvement ainsi que les valeurs limites acceptables qui leurs sont associées. Une action réflexe est liée à chaque dépassement dans un sens ou dans l'autre de ces valeurs limites. Le chemin ainsi paramétré est fourni au système d'exécution qui prend en charge le contrôle de l'exécution du plan.
Les actions réflexes qui sont pré-calculées consistent principalement à stopper le robot puis à réaliser une marche arrière jusqu'au point où les valeurs capteurs sont devenues anormales. Un nouveau cycle modélisation, planification, action est exécuté après chacune de ces actions.
Cette architecture suit toujours le cycle S.M.P.A. et conserve un fort caractère séquentiel. La composante réactive se résume à l'exécution de manœuvres réflexes lorsque le véhicule est en danger. Ce système représente en quelque sorte le degré d'intégration minimum d'une composante réactive dans une architecture délibérative.

1. Cette classification pourra être contestée, mais elle nous apparaît comme la plus adaptée.

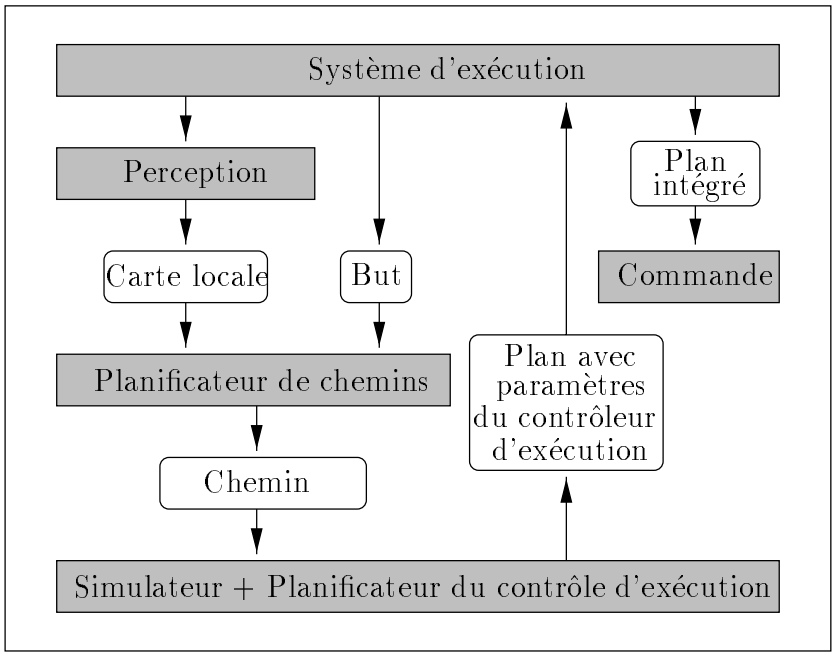

FIG. 1.5 – *L'architecture du JPL.*

1.3.1.2 L'architecture de Payton

L'architecture proposée par D. W. Payton [126] se base sur une décomposition verticale en quatre modules, et conserve un fort caractère hiérarchique. La décomposition se fonde sur le niveau de prétraitement des données capteurs (modélisation) demandé par chaque module. On retrouve ainsi successivement :

- un planificateur de missions chargé de définir une série de buts géographiques et de spécifier les contraintes sur le déplacement ;
- un planificateur de chemins chargé de relier les buts géographiques par des chemins à partir d'un modèle global de l'environnement. Le temps de réponse de ce niveau peut être de quelques minutes ;
- un planificateur local dont le rôle est de sélectionner les manœuvres qui doivent permettre au véhicule de suivre les chemins issus du niveau supérieur. Le temps de réponse de ce niveau ne doit pas excéder quelques secondes ;
- un planificateur réflexe chargé de la commande en temps réel du véhicule.

Cette architecture est décrite par la figure 1.6. Chaque module est à son tour décomposé en un certain nombre d'agents experts communiquant entre eux par l'intermédiaire d'un tableau noir. L'activité d'un niveau peut être commandée par le niveau supérieur à travers la sélection des experts à prendre en compte. Pour cela, Payton regroupe les agents en plusieurs ensembles qui ne s'activent que lorsqu'ils en reçoivent l'ordre

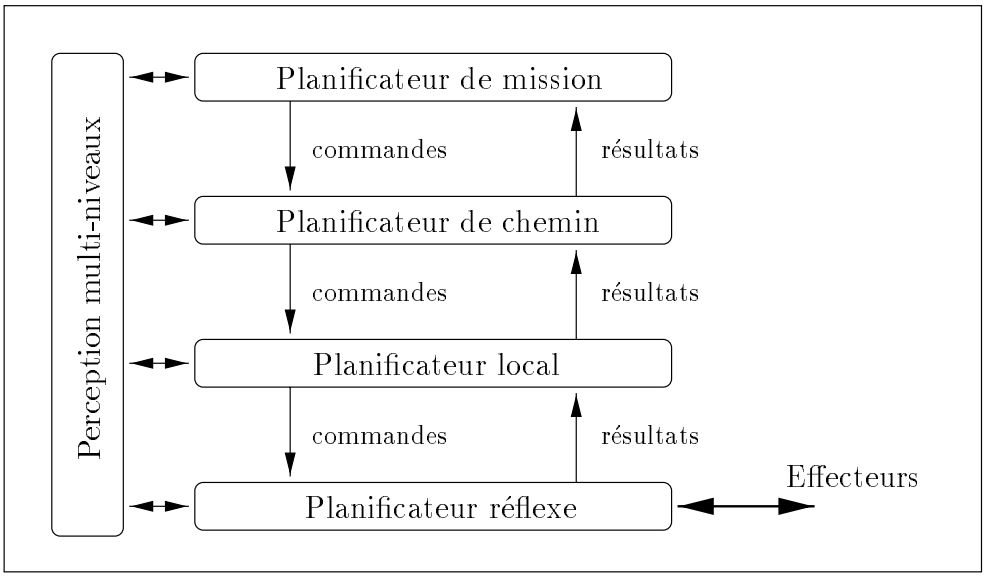

FIG. 1.6 – *L'architecture de Payton.*

du niveau supérieur. Dans le sens inverse, un module peut communiquer au module supérieur des informations sur son état ou sur l'échec de sa tâche.
Seul le niveau de planification réflexe est décrit en détail par l'auteur. Les agents experts y sont des comportements réflexes (e.g. suivi de mur, évitement d'obstacle) associés à des capteurs virtuels. Ces derniers sont définis comme des fonctions de prétraitement des données capteurs chargées d'extraire des informations spécialisées. Il peut s'agir par exemple de détection et de reconnaissance d'obstacle ou de calcul de position. Lorsqu'un de ces capteurs obtient une information pertinente, il la communique au ou aux comportements associés qui envoient une commande sur un tableau noir. Un arbitre sélectionne une de ces commandes à partir d'une priorité associée à l'avance à chaque expert.
Cette architecture permet de planifier des missions complexes qui sont exécutées en conservant une forte réactivité. Son principal point faible provient du mode de communication entre les différents niveaux. Le regroupement des comportements réalisé une fois pour toutes offre en effet très peu de flexibilité.

1.3.1.3 L'architecture TCA de Simmons

L'architecture TCA (Task Control Architecture) présentée par R. G. Simmons [148] rompt avec l'aspect hiérarchique des modules des architectures présentées jusqu'ici. Elle se compose d'un nombre arbitraire de modules, spécialisés chacun dans une tâche précise, dialoguant avec un module de gestion central.
Les modules spécialisés réalisent les tâches spécifiques au robot, alors que le module central est responsable principalement de l'acheminement des messages entre les modules et du maintien des informations internes.

Le système met à la disposition des modules plusieurs types de messages afin de demander des informations, d'envoyer des commandes ou encore de demander la décomposition d'une tâche en sous-tâches (planification). Chacun de ces messages transite par le module central, ce qui donne au système un fort caractère centralisé.

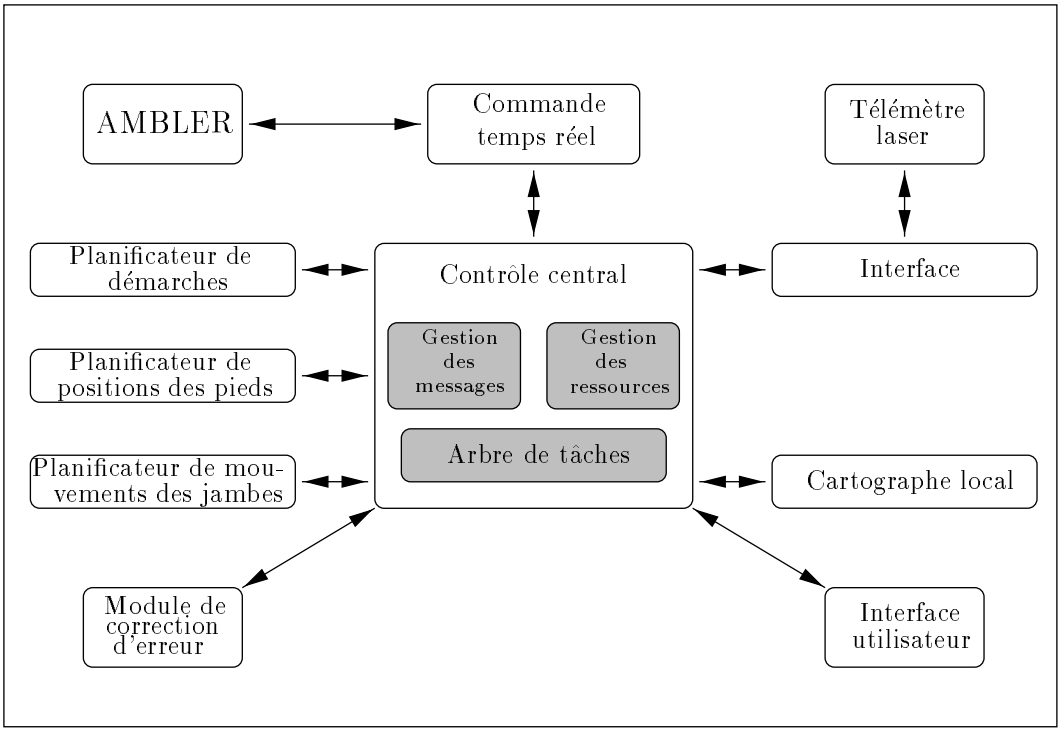

FIG. 1.7 – *L'architecture TCA appliquée au robot à pattes AMBLER.*

Au cœur de l'architecture TCA se trouve une représentation hiérarchique des relations entre les tâches/sous-tâches. Cette structure appelée arbre des tâches, est construite automatiquement par le module central au fur et à mesure de la réception des messages et de la réalisation des sous-tâches.
La figure 1.7 représente l'application de l'architecture TCA à la gestion du robot à pattes AMBLER. Dans cette application, l'accent est mis beaucoup plus sur la planification que sur la réactivité. Les différents modules sont spécialisés dans des tâches de planification particulières (démarche, position des pieds, mouvement des jambes). Les seuls traitements réellement réactifs concernent des actions entreprises lorsque des exceptions sont déclenchées. Il s'agit principalement de stabiliser le robot en cas de problèmes (e.g. glissement d'une patte).
Le reproche généralement fait à l'encontre de cette architecture provient de la centralisation du traitement et du mécanisme de communication par messages qui peuvent impliquer des temps de réponse importants incompatibles avec une application sur des robots rapides. Dans le cas du robot à pattes AMBLER, des mécanismes réflexes d'urgence (stabilisation) sont d'ailleurs implémentés en dehors de cette architecture pour obtenir une vitesse de réaction suffisante.

1.3.2 Architectures à forte composante réactive

Au contraire de celles de la section précédente, les architectures que nous présentons ici se distinguent par une partie comportementale prédominante sur laquelle des aspects délibératifs sont surajoutés.

1.3.2.1 L'architecture AuRA D'Arkin

L'approche de R. C. Arkin [7, 8] s'appuie sur la notion de schémas moteurs et de schémas perceptifs. Cette notion, issue de l'éthologie, est utilisée pour décrire le lien entre l'action et la perception. Un schéma moteur est la spécification d'un comportement générique devant être instancié pour donner un mouvement particulier du robot. Il peut s'agir par exemple de déplacement en ligne droite, d'évitement d'obstacle[1], d'un mouvement vers un but ou encore de simples déplacements aléatoires. A chacune de ces briques de base est associé un schéma perceptif fournissant uniquement les données nécessaires au traitement. Arkin parle à ce propos de *perception orientée action.*
L'architecture AuRA (pour Autonomous Robot Architecture) se compose de cinq sous-systèmes principaux (voir figure 1.8) : perception, cartographie, planification, moteur et contrôle homéostatique.

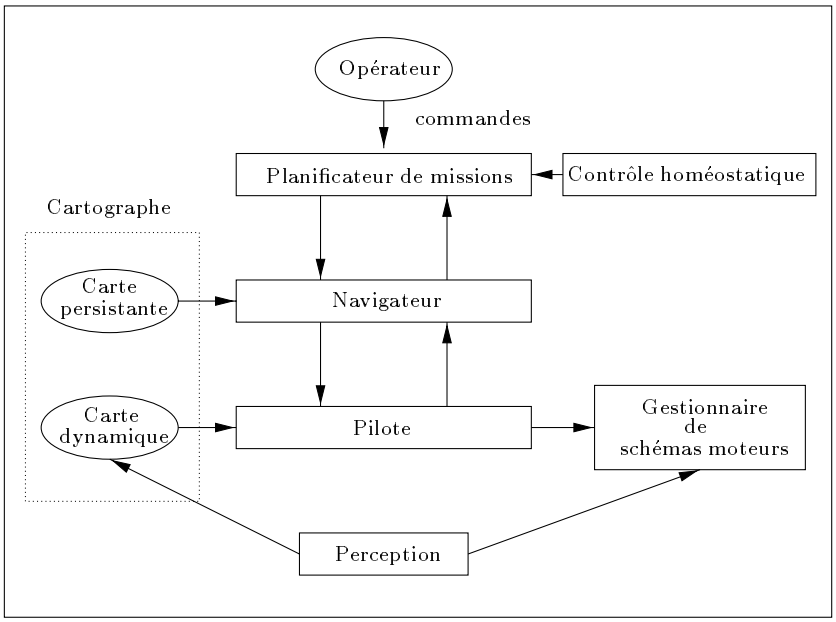

FIG. 1.8 – *L'architecture AuRA.*

– Le sous-système de perception traite les données capteurs (vision, ultrasons et capteurs internes) et fournit les résultats au cartographe et au gestionnaire de schémas moteurs.

1. L'instanciation correspond dans ce cas à la spécification de la position de cet obstacle.

- Le cartographe utilise deux types de représentations du monde. Il maintient d'une part une mémoire à long terme sur la base d'informations connues *a priori*, et d'autre part une mémoire à court terme représentant un modèle dynamique de l'environnement obtenu à partir des données capteurs.

- Le sous-système de planification se compose lui-même de trois modules : un planificateur de missions, un navigateur et un pilote. Le premier fournit un certain nombre de sous-buts à partir des spécifications de la mission. Le deuxième délivre un chemin permettant de les relier. Enfin le troisième a en charge la sélection et l'instanciation des schémas moteurs.

- Le système de contrôle homéostatique a pour but de maintenir les variables internes du robot (e.g. niveau de charge) dans un domaine acceptable. Il peut intervenir sur le traitement du planificateur de missions si nécessaire.

- Le système moteur, enfin, fait le lien avec le niveau physique du robot.

Chaque schéma moteur activé propose un mouvement sous forme vectorielle. C'est la somme de ces différents vecteurs qui détermine la commande appliquée au robot.
La partie délibérative (sous-système de planification) ne produit au final qu'une série de points que le robot doit rejoindre. C'est le pilote qui a en charge la sélection des schémas permettant d'atteindre ces points en évitant les obstacles. L'un des inconvénients de cette approche est qu'elle ne permet pas de planifier des missions complexes impliquant par exemple des successions de manœuvres.
De plus la combinaison des différents schémas moteurs, dont découle suivant l'auteur la réalisation de comportements de plus haut niveau, ne permet pas de spécifier précisément le type de mouvements qui seront obtenus. Des «conflits» peuvent également apparaître entre des schémas spécifiant des mouvements opposés (e.g. attraction pour un but et évitement d'obstacle).

1.3.2.2 L'architecture SSS de Connel

L'architecture SSS (pour Symbolic, Subsumption, Servo) proposée par J. H. Connel [26] se compose de trois niveaux successifs. Ce découpage décrit par la figure 1.9 se base sur une discrétisation successive de l'espace des états du robot, puis du temps. Le plus bas niveau, chargé des asservissements et de la perception, opère dans un domaine de temps et d'espace continu. Le second niveau basé sur l'architecture subsumption de Brooks travaille lui aussi de manière continue dans le temps, mais il utilise une représentation discrète de l'espace des états puisque chaque comportement y est spécialisé dans un traitement bien particulier comme le suivi de mur ou le passage de porte. C'est ce niveau qui assure la partie réactive du système. Enfin, le dernier niveau réalise un traitement symbolique et discrétise également le temps sur la base d'événements significatifs. Cette discrétisation successive est mise en avant par l'auteur mais se retrouve dans la majeure partie des architectures hybrides.

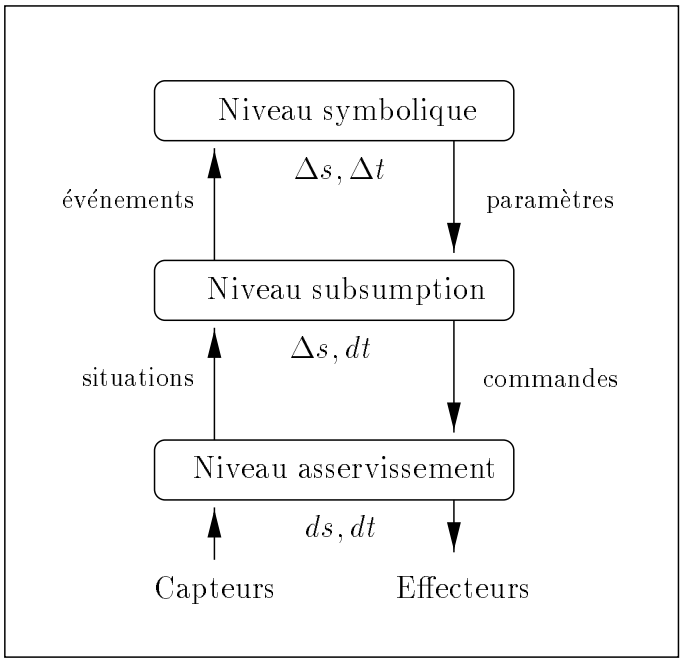

FIG. 1.9 – *L'architecture SSS.*

Les communications entre ces différents niveaux existent dans les deux sens. Le niveau symbolique reçoit de la part du niveau subsumption un certain nombre d'événements et agit sur lui à travers la sélection des comportements ainsi que le choix de leurs paramètres. Ce module possède une carte de l'environnement contenant des «balises» reliées par des chemins (segments de droites).

Le niveau le plus bas (asservissement et perception) envoie au niveau subsumption des informations capteurs sur la situation dans laquelle se trouve le robot. Il reçoit en retour les commandes à appliquer sur les moteurs.

De manière générale, chaque module peut faire remonter des informations vers le niveau supérieur et reçoit de celui-ci les commandes à effectuer. Connel a testé avec succès cette architecture dans un environnement intérieur structuré.

Le niveau le plus bas, bien que rarement évoqué, est toujours présent dans les autres architectures que nous présentons. Le niveau symbolique se résume en pratique à la construction d'une table d'éventualités[1] indiquant dans quelles conditions les comportements de la couche subsumption doivent être activés ou non.

L'approche utilisée dans ce système, est donc principalement basée sur l'architecture subsumption. L'auteur y rajoute uniquement un mécanisme plus élaboré de sélection ou d'inhibition des comportements.

1. Contingency table.

1.3.3 Architectures à 3 niveaux

Nous regroupons dans cette section les travaux les plus récents qui mettent en avant la nécessité de disposer d'un niveau intermédiaire entre les composantes réactives et délibératives du système[1].

1.3.3.1 Les architectures ATLANTIS et 3T

Les architectures ATLANTIS de E. Gat [42, 43] et 3T de R. P. Bonasso [19] présentent de nombreuses similitudes.
On retrouve dans ces deux architectures trois parties principales : un planificateur pour la partie délibérative, un ensemble de mécanismes de commande réactifs et un niveau intermédiaire chargé de gérer les séquences d'actions. Ces trois niveaux sont appelés *controller, sequencer* et *deliberator* par Bonasso et *skill layer, sequencing layer* et *planning layer* par Gat.
La couche basse rassemble plusieurs algorithmes implémentant des comportements réactifs simples tels que le suivi de mur ou l'évitement d'obstacle. Ces derniers ne gèrent aucune mémoire sur l'état interne du robot. Ce niveau constitue une sorte de bibliothèque de *talents*, utilisés à la demande des niveaux supérieurs.
Le rôle du niveau intermédiaire est de sélectionner quel comportement ou quel groupe de comportements doit être actif à un moment donné. Il peut également fournir les paramètres utiles à l'exécution de ces derniers. Le principe utilisé est celui de *l'ordonnancement conditionnel*[2] [43]. Cette méthode autorise le déroulement de plusieurs plans parallèles en interaction, et permet la gestion de plusieurs solutions choisies en fonction des évènements internes ou externes. Le but peut être atteint par plusieurs voies différentes en fonction de ces événements. Les résultats de la planification sont utilisés pour guider les mouvements du robot sans pour autant proposer une séquence linéaire unique d'actions.
La réalisation de ce niveau s'appuie sur l'utilisation des langages ESL pour ATLANTIS et RAP pour 3T (Gat a également utilisé ce dernier dans un premier temps). Le langage RAP (Reactive Action Packages) proposé par R. J. Firby [38] permet la spécification d'un ensemble de procédures (séquences d'actions) activées lorsque certaines conditions sont vérifiées. Une RAP est une représentation groupant un but, et tous les moyens connus de l'atteindre en fonction du contexte d'exécution. Elle intègre également une procédure de test permettant de vérifier si le but recherché a été atteint. Le système maintient un agenda des tâches à réaliser et active successivement les RAP correspondant à ces tâches. Afin d'atteindre le but fixé une RAP propose plusieurs voies choisies en fonction du contexte courant. Elle peut se décomposer à son tour en plusieurs RAP inscrites sur l'agenda ou demander l'activation de systèmes de commande du niveau inférieur. Un interpréteur tourne de manière continue et gère l'agenda. Il intègre égale-

1. On retrouve une préoccupation similaire dans les systèmes hybrides en intelligence artificielle [119].

2. Conditional sequencing.

ment une mémoire contenant les faits connus sur l'environnement et sur le robot, ainsi que les sous-buts déjà atteints.
Le fonctionnement du langage ESL est assez semblable à celui des RAP. Dans les deux cas, la spécification des procédures exploite l'idée suivante introduite par Noreils [115] : *Plutôt que d'essayer de construire des algorithmes qui n'échouent jamais (ce qui est impossible sur des robots réels), il vaut mieux construire des algorithmes qui n'échouent jamais à détecter un échec. Ceci permet à d'autres composantes du système d'entreprendre des actions permettant de corriger ces échecs.* Gat [43] parle à ce propos d'échec instructif[1].
La couche supérieure s'exécute dans une échelle de temps différente des deux autres niveaux. Elle a pour but de fournir les séquences d'actions utilisées par la couche intermédiaire. Elle fonctionne de deux manières différentes suivant le système. Dans 3T, elle produit à l'avance le plan qui est utilisé par la couche d'ordonnancement, alors que dans ATLANTIS elle s'active en réponse à des requêtes de cette dernière.
Les trois niveaux fonctionnels de ces systèmes se retrouvent dans l'ensemble des architectures de commande pour robots mobiles les plus récentes. On parle généralement d'architectures à trois niveaux pour décrire ce type de systèmes. Le niveau intermédiaire d'ordonnancement peut être vu comme une interface entre les composantes réactives et délibératives du système. Il doit intégrer lui-même à la fois une capacité de réaction rapide aux évènements internes ou externes, et une possibilité de représenter des informations de haut niveau.

1.3.3.2 L'architecture du LAAS

Le système proposé par le LAAS [4] pour la plate-forme de robot HILARE se compose également de trois niveaux principaux appelés niveau fonctionnel, niveau d'exécution et niveau décisionnel :

- le niveau fonctionnel inclut toutes les fonctionnalités de base nécessaires pour la perception et l'action. Les éléments constitutifs de ce niveau peuvent donc être des fonctions de traitement des données, des boucles de commande ou encore des observateurs chargés de réagir à un événement précis. Chacun d'entre eux est encapsulé dans un module pouvant recevoir des commandes. Les communications entre ces modules et le niveau d'exécution, ou entre deux modules différents, s'effectuent selon un protocole client/serveur. Lorsqu'une requête (qui peut inclure des paramètres d'initialisation) est envoyée, le module concerné s'active jusqu'à l'achèvement de la tâche demandée ou jusqu'à son blocage dans un état d'échec. Il peut à son tour envoyer des requêtes aux autres modules dont les services lui sont nécessaires. L'organisation de cette couche fait donc apparaître un véritable réseau de modules interagissant les uns avec les autres. Après l'achèvement de sa tâche, le module ayant reçu la requête émet un rapport d'exécution pouvant contenir des données résultats.

1. Cognizant failure.

Les modules possèdent tous la même structure. Ils sont instanciés à partir d'un module générique à l'aide d'un outil appelé $G^{en}oM$ (Generator of Modules).

De manière à rendre ce niveau le plus portable possible, les modules sont interfacés avec les ressources physiques, à travers un *niveau robot logique* ;

- le niveau d'exécution est un système purement réactif, sans capacités de planification. Il reçoit du niveau décisionnel les séquences d'actions à exécuter. Il sélectionne et paramètre dynamiquement les modules du niveau fonctionnel. Pour cela, il utilise un automate généré automatiquement à partir d'un ensemble de règles logiques. Les paramètres des différents modules sont généralement le résultat du traitement d'autres modules du niveau fonctionnel ;

- le niveau décisionnel prend en charge tous les traitements qui demandent une connaissance globale de la tâche et du contexte d'exécution. Il réalise les traitements délibératifs tels que la planification et la prise de décision. Pour permettre à ce niveau de rester réactif malgré les temps de traitement importants requis, celui-ci est décomposé en deux éléments : un planificateur et un superviseur. Le planificateur produit les séquences d'actions nécessaires pour réaliser le but recherché. Il est utilisé comme une ressource par le superviseur qui interagit avec le niveau d'exécution et contrôle le déroulement du plan. En plus de ce plan, le planificateur spécifie des modalités d'exécution qui sont principalement la description des événements à surveiller et des actions réflexes à entreprendre lorsqu'ils se produisent.

  En fonction des tâches à accomplir, le niveau de décision peut se composer de plusieurs ensembles planificateur + superviseur. C'est le cas sur la figure 1.10 où le traitement est décomposé en une planification puis un raffinement des tâches.

1.3.3.3 L'architecture du projet SHARP

Nous terminons ce tour d'horizon par la présentation de l'architecture de commande proposée par le projet SHARP de l'INRIA Rhône-Alpes [90]. Le but de cette approche est de prendre en compte les contraintes particulières aux véhicules de type voiture afin d'obtenir des séquences de mouvements «sécurisés» et «réguliers». C'est autour de ce système que tournent les travaux réalisés durant cette thèse.
Le système (voir figure 1.11) se compose d'un planificateur de trajectoire travaillant hors-ligne, d'un noyau décisionnel générant en temps réel un plan de mouvements utilisant des manœuvres basées sur les données capteurs du robot (les *Manœuvres Basées Capteurs*) et d'un contrôleur de mouvements utilisant un ensemble de *talents*.
La notion de Manœuvre Basée Capteurs (MBC) est centrale dans cette architecture. Ces manœuvres, regroupées dans une bibliothèque, décrivent l'ordonnancement des actions à entreprendre pour réaliser une tâche particulière. Chacune de ces actions implique un ou plusieurs contrôleurs spécialisés (i.e. les talents). Les MBC peuvent être implémentées sous la forme d'un script (voir figure 1.12) ou d'une base de règles.

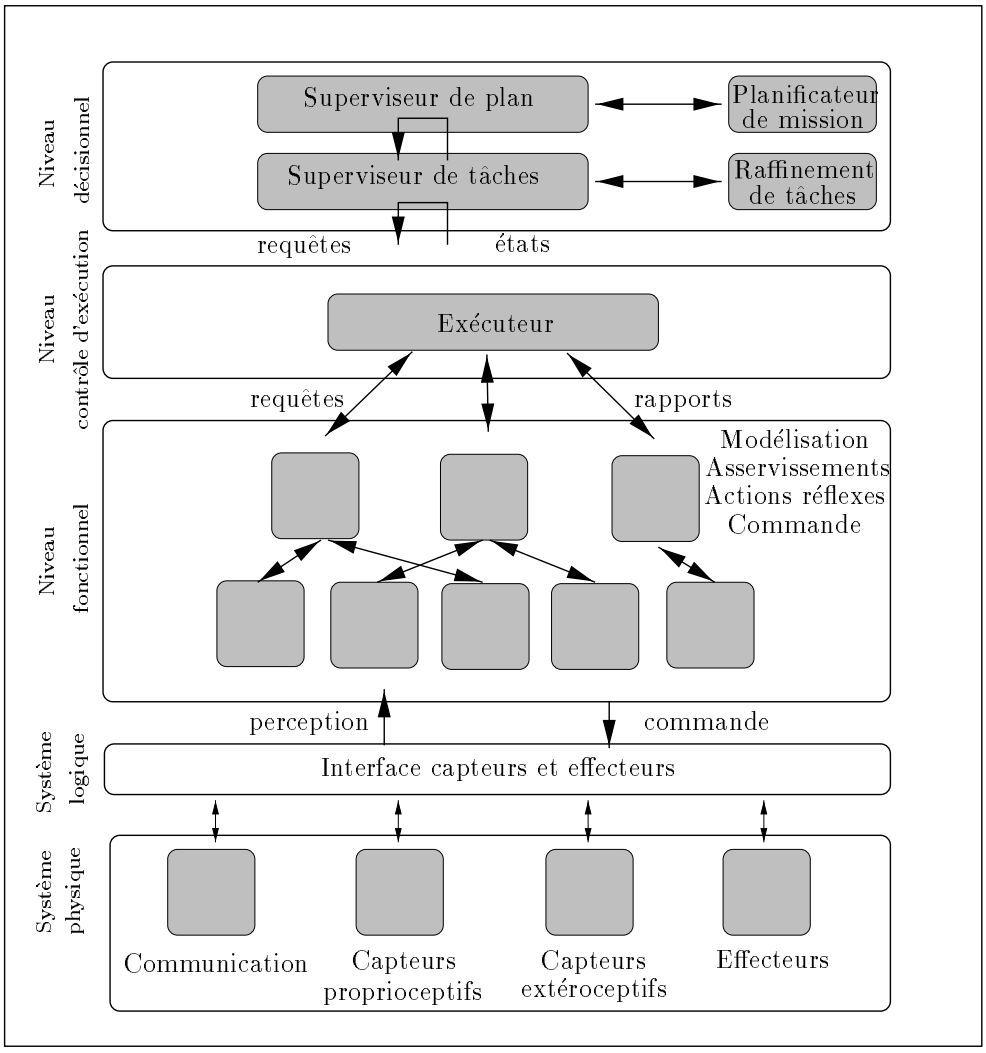

FIG. 1.10 – *L'architecture développée au LAAS.*

Les opérations réalisées par le système lors de la réception d'une requête de tâche sont les suivantes :

- Le superviseur de missions génère un *Plan de Mouvement Paramétré* (PMP) en combinant un ensemble de manœuvres guidées capteurs avec une trajectoire nominale lorsque celle-ci est nécessaire. Cette trajectoire nominale peut être soit un mouvement en avant en ligne droite, si le modèle du monde est trop pauvre ou si des contraintes de temps empêchent l'utilisation du planificateur, soit une suite de configurations du robot représentant un mouvement théoriquement sûr et exécutable. Le planificateur de trajectoires utilise trois types d'informations : un modèle du monde connu *a priori*, des données provenant des capteurs du véhicule (position et vitesse des obstacles) et une prédiction donnant les comportements probables des obstacles en mouvement. Il produit dans un premier temps

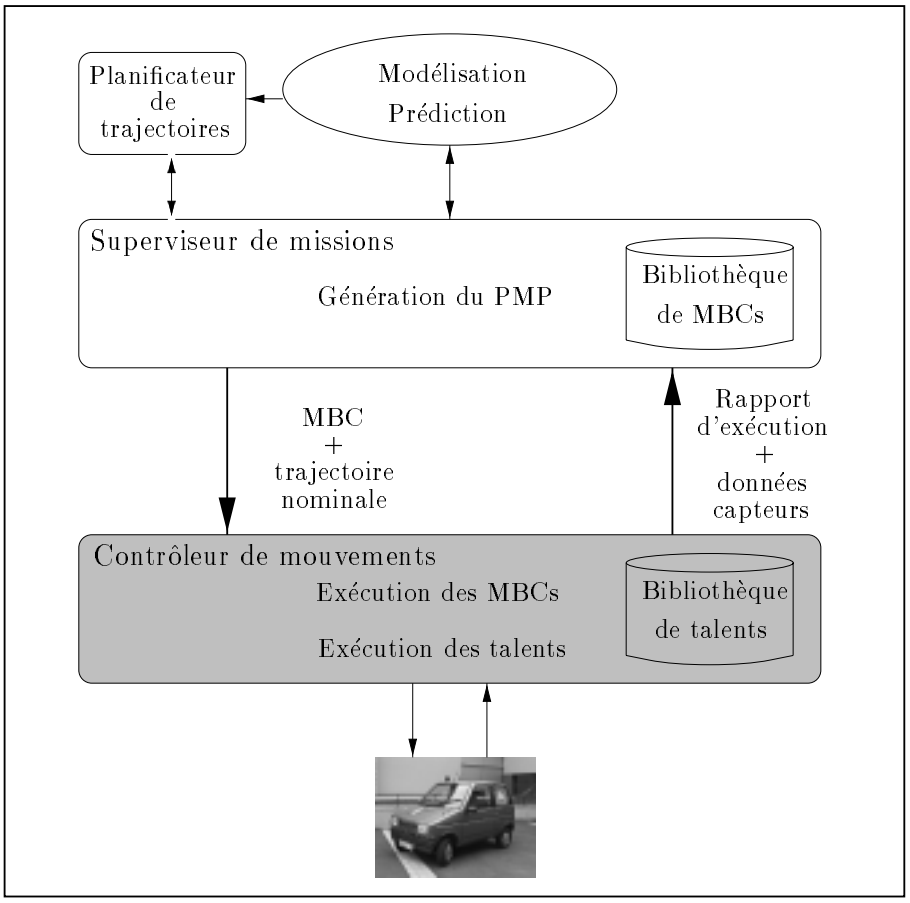

FIG. 1.11 – *L'architecture de commande du projet SHARP.*

un chemin respectant les contraintes cinématiques du véhicule (voir §A.2.2), puis plaque sur ce chemin un profil de vitesse de manière à respecter les contraintes dynamiques. Une description plus complète de ce planificateur pourra être trouvée dans [145] pour ce qui concerne la planification de chemin et dans [39] pour ce qui concerne la vitesse.

– Le contrôleur de mouvements se charge d'exécuter de manière réactive la manœuvre courante. Pour cela il fixe les paramètres des talents utilisés. Ces paramètres peuvent être connus à l'avance (e.g. courbure de la route, accélération ou vitesse limite) ou provenir des capteurs du véhicule (e.g. distance des obstacles, espace disponible). En cas d'échec dû à un événement impossible à traiter par la manœuvre (e.g. intrusion d'un nouvel obstacle), le contrôleur de mouvements rapporte cette information au noyau décisionnel qui décide soit de recommencer une procédure de planification, soit d'insérer en temps réel une autre manœuvre basée capteurs.

La figure 1.12 montre le script correspondant à une MBC permettant à la voiture de se garer en créneau. Cette manœuvre a été testée avec succès sur un véhicule réel.

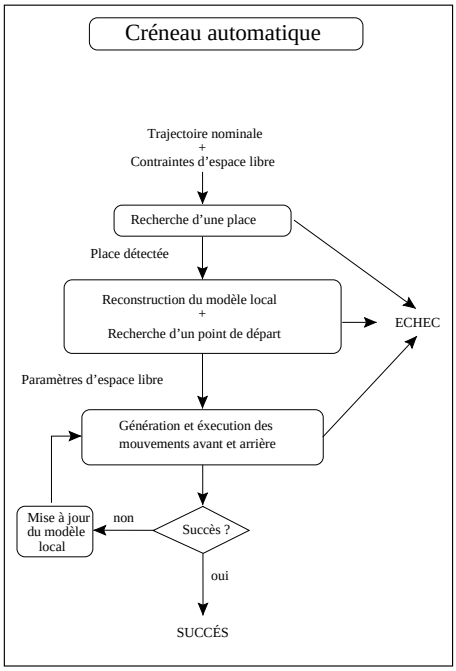

FIG. 1.12 – *Script d'une manœuvre basée capteurs.*

On retrouve dans cette architecture les trois niveaux des architectures 3T, ATLANTIS ou du LAAS. La bibliothèque de talents regroupe les systèmes de commande réactifs utilisés. Le niveau d'exécution correspond à notre contrôleur de mouvement. La principale nouveauté se trouve dans le fonctionnement du gestionnaire de missions. L'introduction des MBC qui constituent en quelque sorte des talents de plus haut niveau permet en effet de gagner un temps très important lors de la phase de planification. Le nombre de situations que peut rencontrer un véhicule dans un contexte particulier n'est en effet pas très élevé, et il ne sert à rien de replanifier la même séquence d'actions chaque fois que le véhicule doit, par exemple, se garer ou dépasser une autre voiture.

1.4 Conclusion

Au cours de ce chapitre nous nous sommes intéressés au fonctionnement de différentes architectures de commande pour robots mobiles proposées dans la littérature. Nous avons vu que les solutions adoptées par les systèmes les plus récents concordent sur la définition de trois niveaux principaux impliquant chacun des traitements différents. Ceci ne se rapporte pas forcement à un découpage physique des architectures de commande mais plutôt à une décomposition fonctionnelle.
Le premier de ces niveaux correspond à une collection de systèmes de commande purement réactifs, spécialisés dans des manœuvres ou des comportements particuliers, et utilisés comme des ressources par les niveaux supérieurs. Les systèmes de commande composant ce niveau peuvent être combinés en réalisant la fusion des consignes qu'ils

proposent comme dans le cas des schémas moteurs de l'architecture AuRA d'Arkin. Les contraintes liées à l'utilisation d'un véhicule rapide et de taille importante, imposent plutôt une exécution en exclusion mutuelle des contrôleurs afin de pouvoir spécifier plus précisément le comportement attendu pour le robot. Le comportement émergeant de la composition de plusieurs systèmes de commande est en effet généralement plus complexe à spécifier de manière précise. Si nous prenons le cas du dépassement d'un véhicule, nous ne désirons pas uniquement obtenir un mouvement sans collisions mais nous voulons aussi pouvoir garantir l'absence d'oscillations dans le mouvement ou pouvoir spécifier par exemple la distance qui séparera les deux véhicules durant la manœuvre, ou la manière dont la voiture se rabattra après le dépassement. Ces critères que nous qualifierons de critères de qualité sont très importants dans le cas d'un véhicule de grande taille évoluant à des vitesses importantes dans un environnement potentiellement occupé par des êtres humains. Ce choix se retrouve également dans les architectures 3T et ATLANTIS, ainsi que dans celle du LAAS.

Le second niveau ou niveau intermédiaire correspond au contrôle de l'exécution des missions. Il doit permettre de suivre un plan définissant une succession de sous-buts par lesquels le robot doit transiter pour atteindre son but final. Chacune de ces situations peut être associée à un contrôleur particulier ou à une instanciation des paramètres de ce contrôleur. Cette étape doit permettre la spécification des réactions attendues face aux événements extérieurs qui peuvent survenir. Les plans exécutés ne doivent donc pas être linéaires. Ils doivent au contraire autoriser plusieurs cheminements possibles conduisant au même but. Dans le cas par exemple du dépassement d'un véhicule, le système doit pouvoir réagir rapidement à l'intrusion d'un nouvel obstacle empêchant la manœuvre et proposer une solution de remplacement. On retrouve par exemple ce type de traitement, dans le contrôleur de mouvement du système proposé par le projet SHARP, dans le niveau contrôle d'exécution du LAAS, ou dans l'utilisation des RAPs dans 3T.

Le dernier niveau correspond à la programmation et à la planification de missions. Il implique des traitements qui peuvent s'effectuer dans des échelles de temps différentes de celles des deux autres niveaux. Il est généralement également chargé de la mise à jour d'un modèle plus ou moins précis du monde dans lequel évolue le robot. On le retrouve dans l'ensemble des architectures hybrides présentées (noyau décisionnel et niveau décisionnel des architectures du projet SHARP et du LAAS par exemple).

Au cours du prochain chapitre, nous présentons les possibilités offertes par les réseaux de neurones artificiels dans le cadre de la réalisation de systèmes de commande. Notre but est de tenter d'identifier leurs utilisations possibles dans le cadre d'une architecture de commande pour robots mobiles respectant les différents points que nous venons d'aborder.

Chapitre 2

Réseaux de neurones artificiels et utilisation en commande

L'utilisation de réseaux de neurones artificiels (RNA) pour la réalisation de systèmes de commande a connu un essor important au cours des dernières années. Dans ce chapitre nous tentons de présenter les possibilités offertes par ces techniques, ainsi que les modalités de leur utilisation, dans ce domaine.

Après une brève présentation de quelques notions générales sur les RNA, nous présentons les modèles de réseaux utilisés dans cette thèse. Ceci nous amène également à aborder dans ce chapitre un autre paradigme fréquemment employé dans le domaine de la commande : la logique floue.

Nous exposons dans un deuxième temps les motivations pour l'emploi des RNA en commande, ainsi que le problème principal qui rend cette utilisation non triviale. Enfin, nous proposons une classification des principales approches de commande utilisant les RNA, rencontrées dans la littérature. Notre but n'est pas d'en dresser une liste complète, mais plutôt de dégager les lignes directrices communes. Certaines architectures sont ainsi absentes de cette classification mais l'on pourra généralement les rattacher à l'une des classes proposées. Pour une revue plus complète, le lecteur pourra se reporter par exemple aux travaux de M. Agarwal [3].

De même nous ne cherchons pas à dresser une liste des différents modèles de RNA rencontrés ni même des algorithmes d'apprentissage, mais nous nous intéressons plutôt à la manière dont ces RNA sont utilisés pour la réalisation d'un système de commande. Pour une description des principaux modèles de réseaux de neurones artificiels, le lecteur pourra se reporter entre autres aux ouvrages de S. Haykin [56] ou R. Hecht-Nielsen [57].

Les exemples choisis dans ce chapitre ne se limitent pas, volontairement, à la robotique mobile. Le terme système de commande pourra donc désigner dans ce qui suit tout dispositif ayant en charge la commande d'un processus quelconque, que ce dernier soit un robot, une usine ou tout autre appareil.

2.1 Quelques notions sur les réseaux de neurones artificiels

Le principe des réseaux de neurones artificiels est né dans les années 40 à partir d'une analogie avec le système nerveux humain. Le terme désigne aujourd'hui un très grand nombre de modèles, dont beaucoup n'ont plus grand chose à voir avec le fonctionnement des neurones biologiques, et doit donc être pris comme une métaphore. Ces différents modèles ont en commun l'utilisation d'automates, appelés neurones ou unités, capables de réaliser chacun un traitement très simple et d'échanger des informations entre eux. On associe généralement aux RNA un algorithme «d'apprentissage» permettant de modifier de manière plus ou moins automatique le traitement effectué afin de réaliser une tâche donnée. Nous n'aborderons ici les RNA qu'avec la vision que pourrait en avoir un ingénieur en oubliant les aspects liés aux sciences cognitives ou à la neurobiologie.

2.1.1 Généralités

Nous présentons rapidement dans cette section, le formalisme que nous utiliserons tout au long de ce mémoire pour décrire les RNA.

2.1.1.1 Le neurone formel

Le neurone formel qui représente la brique de base des RNA est un automate dont le modèle s'inspire de celui d'un neurone biologique. On peut le décrire par les éléments suivants (pour un neurone d'indice i) :

- son état (aussi appelé activation) a_i, qui peut être une valeur réelle ou booléenne. Cet état est généralement également choisi comme valeur de sortie du neurone ;
- ses connexions d'entrée auxquelles sont associés des poids w_{ij} (j est l'indice du neurone partageant la connexion) ;
- sa fonction d'entrée réalisant un prétraitement (généralement une somme pondérée) des entrées ;
- sa fonction d'activation (ou de transfert) $g_i(x)$, qui calcule à partir du résultat de la fonction d'entrée l'activation du neurone.

La figure 2.1 représente un neurone formel appliquant une fonction de seuil sur la somme pondérée de ses différentes entrées.

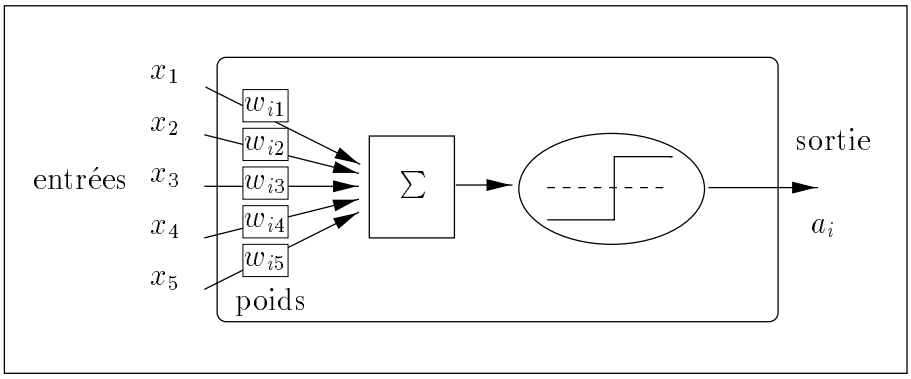

FIG. 2.1 – *Un neurone formel.*

2.1.1.2 Réseaux de neurones artificiels

Un réseau de neurones[1] consiste en un ensemble de neurones reliés entre eux par des connexions pondérées. Il se caractérise principalement par le type des unités utilisées et par sa topologie. On distingue souvent deux types de neurones particuliers dans un réseau : les neurones d'entrée recevant les données du monde extérieur, et les neurones de sortie fournissant le résultat du traitement effectué. Les autres unités sont généralement qualifiées de cachées. Cette distinction n'est toutefois pas obligatoire et tous les neurones peuvent très bien communiquer dans les deux sens avec l'extérieur.

Il est fréquent de différencier les réseaux suivant la présence ou non de cycles dans le graphe orienté des connexions entre les neurones [118]. On parle dans le cas positif de *réseaux récurrents*. Il est à noter que ces connexions cycliques, dont les valeurs dépendent des activations passées des unités du réseau, permettent de mieux traiter des problèmes comportant un aspect temporel. L'apprentissage est cependant généralement assez complexe dans ces réseaux, et leurs propriétés sont souvent moins bien connues que celles des réseaux non récurrents.

Une autre distinction importante est généralement faite entre codage *local* de l'information d'entrée et codage *distribué*. Dans les réseaux réalisant un codage local, chaque unité est spécialisée dans le traitement d'une petite partie de l'espace d'entrée, alors que dans les réseaux réalisant un codage distribué tout traitement fait intervenir l'ensemble des neurones.

L'un des principaux attraits des RNA concerne les capacités d'apprentissage que possèdent certains modèles. On entend généralement par apprentissage la modification automatique des poids des connexions ou plus rarement du nombre et de l'organisation des neurones, afin d'adapter le traitement effectué par le réseau à une tâche particulière. On distingue trois familles d'apprentissage en fonction de la nature des informations

1. Par abus de langage nous omettrons désormais parfois le terme artificiel. Sauf précision contraire, tous les neurones évoqués seront donc artificiels. Le terme réseau employé seul désignera de même un RNA.

disponibles et du but recherché :

- *l'apprentissage supervisé* pour lequel il est nécessaire de disposer d'un ensemble de couples de données {entrées du réseau ; sorties désirées correspondantes}, appelées exemples ou patrons. La différence entre la sortie du réseau et la sortie désirée donne ainsi une mesure d'erreur quantitative sur le calcul effectué par le réseau, qui est utilisée pour réaliser l'adaptation ;

- *l'apprentissage semi-supervisé* ou *apprentissage par renforcement* pour lequel seule une mesure d'erreur qualitative (échec ou réussite) est disponible ;

- *l'apprentissage non supervisé* pour lequel il n'y a pas de réponse désirée. La tâche du réseau peut être par exemple dans ce cas de créer des regroupements de données selon des propriétés communes (catégorisation).

2.1.2 Quelques modèles de RNA

Nous présentons dans cette section plusieurs modèles de réseaux, importants pour la suite de notre travail. Nous limitons cet exposé à des réseaux non récurrents. Nous présenterons toutefois dans le chapitre 5 un modèle récurrent particulier.

2.1.2.1 RNA multi-couches

Structure Nous présentons ici une des architectures de réseaux les plus utilisées. Elle correspond à une organisation des neurones en n couches successives ($n \geq 3$). La première couche, dont les neurones voient leur activation forcée à la valeur des données d'entrée, est appelée couche d'entrée. La dernière est appelée couche de sortie. Les seules connexions présentes dans ce type de réseau relient chaque neurone avec l'ensemble de ceux de la couche suivante (voir figure 2.2). La propagation de l'information se déroule ainsi en sens unique depuis la couche d'entrée vers la couche de sortie.
Les réseaux de cette classe sont souvent appelés perceptrons multi-couches (par analogie avec un des premiers modèles de réseau proposé [142]) ou plus simplement réseaux multi-couches.
La fonction d'activation utilisée pour les neurones peut être n'importe quelle fonction croissante et dérivable. On utilise souvent une fonction sigmoide telle que :

$$g(x) = \frac{1}{1 + \exp(-x)}$$

Elle prend pour paramètre la somme pondérée des entrées du neurone :

$$e_i = \sum_j W_{ij} a_j + b_i,$$

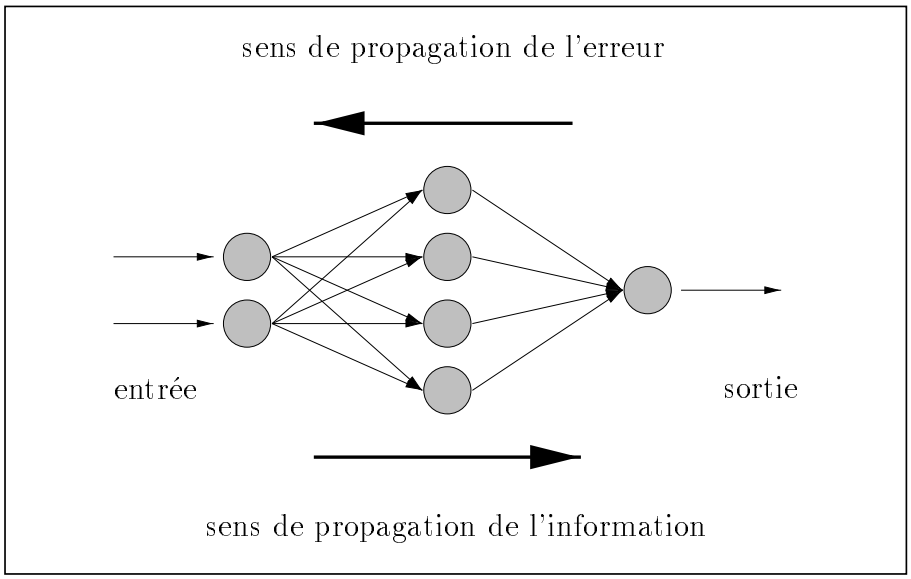

FIG. 2.2 – *Un réseau multi-couches comportant 2 neurones d'entrée, 4 neurones cachés et un neurone de sortie.*

où j parcourt l'ensemble des neurones envoyant une connexion vers le neurone i, W_{ij} est le poids de la connexion entre le neurone j et le neurone i et b_i est un paramètre optionnel appelé biais [1].

Apprentissage Nous présentons ici l'algorithme de rétropropagation du gradient qui est le plus connu pour réaliser l'adaptation des réseaux multi-couches. C'est à sa découverte que l'on doit le renouveau d'intérêt pour les RNA apparut au début des années 80. Il s'agit d'une méthode d'apprentissage supervisé, fondée sur la modification des poids du réseau dans le sens contraire à celui du gradient de l'erreur par rapport à ces poids. Nous allons présenter brièvement la méthode d'obtention de ce gradient, qui se base sur le calcul des dérivées partielles successives de fonctions composées.
La mesure de performance utilisée est l'erreur quadratique :

$$Q = \frac{1}{2}\sum_i [a_i - s_i]^2, \tag{2.1}$$

où i parcourt les indices des neurones de sortie, et a_i et s_i représentent respectivement l'activation mesurée et l'activation désirée pour ces neurones. Les poids du réseau sont modifiés en suivant la règle :

$$\Delta W_{ij} = -\eta \frac{\partial Q}{\partial W_{ij}}, \tag{2.2}$$

1. Il peut être représenté par le poids d'une connexion provenant d'un neurone dont l'activité reste fixée à un (neurone biais).

où η est une constante positive appelée pas du gradient. Le calcul de la quantité $\frac{\partial Q}{\partial W_{ij}}$ se fait en partant de la couche de sortie et en se déplaçant vers la couche d'entrée. Cette propagation, suivant le sens inverse de celui de l'activation des neurones du réseau, justifie le nom de l'algorithme. Le calcul est décomposé de la manière suivante :

$$\frac{\partial Q}{\partial W_{ij}} = \frac{\partial Q}{\partial a_i}\frac{\partial a_i}{\partial e_i}\frac{\partial e_i}{\partial W_{ij}}$$

En posant, $\delta_i = \frac{\partial Q}{\partial a_i}\frac{\partial a_i}{\partial e_i}$ on obtient :

$$\frac{\partial Q}{\partial W_{ij}} = \delta_i \frac{\partial e_i}{\partial W_{ij}},$$

et puisque $\frac{\partial e_i}{\partial W_{ij}} = a_j$:

$$\Delta W_{ij} = -\eta \delta_i a_j \tag{2.3}$$

La quantité δ_i est appelée contribution à l'erreur du neurone i. Dans le cas où i est l'indice d'un neurone de sortie, on obtient :

$$\frac{\partial Q}{\partial a_i} = (a_i - s_i),\ \frac{\partial a_i}{\partial e_i} = g'(e_i),$$

et donc :

$$\delta_i = g'(e_i)(a_i - s_i) \tag{2.4}$$

Dans le cas où i est l'indice d'un neurone caché, on pose :

$$\frac{\partial Q}{\partial a_i} = \sum_k \frac{\partial Q}{\partial a_k}\frac{\partial a_k}{\partial a_i},$$

où k parcourt les indices de tous les neurones vers lesquels le neurone i envoie une connexion.
Le calcul nous donne :

$$\frac{\partial Q}{\partial a_k}\frac{\partial a_k}{\partial a_i} = \frac{\partial Q}{\partial a_k}\frac{\partial a_k}{\partial e_k}\frac{\partial e_k}{\partial a_i} = \delta_k \frac{\partial e_k}{\partial a_i} = \delta_k W_{ki}$$

Nous obtenons donc :

$$\frac{\partial Q}{\partial a_i} = \sum_k \delta_k W_{ki},$$

et :

$$\delta_i = g'(e_i)\sum_k \delta_k W_{ki} \tag{2.5}$$

Cet algorithme, présenté ici dans sa version la plus simple, possède de nombreuses variantes. Elles correspondent généralement à l'utilisation de valeurs variables pour la constante η [139], ou à l'utilisation de méthodes du deuxième ordre pour le calcul du gradient [35].

On utilise souvent une version légèrement différente de l'équation 2.2 pour calculer la quantité dont doivent être modifiés les poids :

$$\Delta W_{ij}(t) = -\eta \frac{\partial Q}{\partial W_{ij}} + \mu \Delta W_{ij}(t-1), \tag{2.6}$$

où μ est une constante appelée momentum, et t représente le temps. Cette version introduit un deuxième terme proportionnel à la dernière adaptation de W_{ij}.
Les modifications des poids peuvent intervenir après chaque présentation d'un patron, ou après la présentation de l'ensemble de la base d'exemples. L'apprentissage nécessite dans tous les cas un grand nombre de présentations de la totalité de ces exemples pour obtenir un résultat satisfaisant.

Propriétés Il a été démontré que moyennant le choix d'une architecture appropriée (i.e. nombre de neurones cachés), les réseaux multi-couches sont capables d'approcher n'importe quelle fonction [30, 64]. On parle *d'approximateurs universels de fonctions.*
Une propriété fondamentale de l'apprentissage réalisé concerne les capacités de généralisation de ces réseaux. Dans le cas où l'architecture initiale est correctement choisie (ce point sera développé dans la section 2.1.3), on constate généralement que les exemples ne sont pas appris «par cœur» mais que le réseau est capable d'étendre les connaissances acquises à des exemples proches ou intermédiaires.
Cependant un certain nombre de problèmes sont liés à l'utilisation de ce type de réseaux (et de manière plus générale de la majeure partie des RNA). On peut remarquer notamment la lenteur de l'apprentissage et surtout l'absence de résultats théoriques garantissant sa convergence. On constate souvent des problèmes liés au blocage de l'apprentissage dans des minima locaux de la fonction d'erreur. Ce point qui est souvent cité comme l'inconvénient principal des réseaux multi-couches doit cependant être tempéré par l'existence de nombreuses approches permettant d'éviter de tels minima [48].
Il est également difficile une fois la phase d'apprentissage terminée de faire acquérir au réseau des connaissances supplémentaires sans repartir de zéro et sans réutiliser l'ensemble de la base d'exemples.
L'utilisation d'un petit nombre d'exemples, cantonnés dans une partie seulement de l'espace d'entrée, conduit souvent à une trop grande spécialisation et à l'oubli des connaissances préalables. On parle dans ce cas d'apprentissage catastrophique.

2.1.2.2 RNA à fonctions de base radiales

Structure Nous présentons ici un deuxième modèle de réseau que nous utiliserons plus loin dans le mémoire : les réseaux à fonctions de base radiales (RBF) ou plus simplement réseaux à bases radiales proposés par J. Moody et C. Darken [104]. On retrouve comme dans le modèle précédent une organisation comportant une couche d'entrée, une couche cachée et une couche de sortie. La principale différence vient du

fait que chaque neurone caché ne réagit ici qu'à une petite partie de l'espace d'entrée (sa zone d'influence).
Pour un réseau comportant n entrées et m unités cachées, l'activation des neurones cachés est donnée par une fonction de type gaussienne (les fonctions d'entrée et d'activation sont confondues) :

$$a_i = \exp(-\frac{1}{2}\sum_{k=1}^{n}\frac{(e_k - c_{k,i})^2}{\sigma_{k,i}^2}) = \prod_{k=1}^{n}\exp(-\frac{1}{2}\frac{(e_k - c_{k,i})^2}{\sigma_{k,i}^2}) \tag{2.7}$$

où i désigne l'indice du neurone, k parcourt l'ensemble des entrées notées e_k, et $c_{k,i}$ et $\sigma_{k,i}^2$ sont des paramètres appelés respectivement centres et variances des gaussiennes [1].
La figure 2.3 présente la forme de cette fonction d'activation pour un neurone possédant une seule entrée.

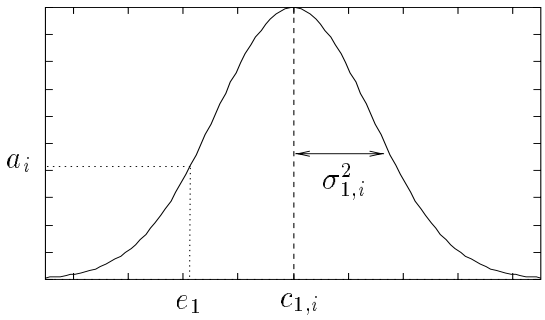

FIG. 2.3 – *Fonction d'activation d'un neurone caché possédant une seule entrée.*

Chacun de ces neurones ne s'active donc de manière significative que pour des valeurs d'entrée relativement proches des centres des gaussiennes.
Les connexions provenant des neurones d'entrée ne sont pas pondérées. L'activation d'un neurone de sortie d'indice i est donnée par :

$$a_i = \frac{\sum_{j=1}^{m} w_{ij} a_j}{\sum_{j=1}^{m} a_j}, \tag{2.8}$$

où j parcourt l'ensemble des indices des neurones cachés. Les neurones de ce type réalisent donc une somme pondérée des valeurs d'activation des neurones cachés.
Le terme $\sum_{j=1}^{m} a_j$ appelé facteur de normalisation n'est pas obligatoire. On parle de réseau normalisé lorsqu'il est employé.

Apprentissage L'apprentissage se fait dans ces réseaux par modification des poids des connexions entre les neurones cachés et les neurones de sortie, et des centres et des variances des gaussiennes. On réalise comme précédemment une descente de gradient ayant pour but de minimiser l'erreur quadratique, dont l'expression est donnée par

1. Le paramètre $\sigma_{k,i}^2$ est appelé variance et $\sigma_{k,i}$ est appelé écart type.

l'équation 2.1. Les modifications des différents paramètres sont données par les règles suivantes :

$$\Delta W_{ij} = -\eta \frac{\partial Q}{\partial W_{ij}},\ \Delta c_{i,k} = -\eta \frac{\partial Q}{\partial c_{i,k}} \text{ et } \Delta \sigma_{i,k} = -\eta \frac{\partial Q}{\partial \sigma_{i,k}} \tag{2.9}$$

Il est également possible comme dans le cas de l'équation 2.6 d'introduire un terme proportionnel à la dernière modification de ces paramètres.
En posant $R = \sum_{j=1}^{m} a_j$ (facteur de normalisation) on obtient :

$$\frac{\partial Q}{\partial W_{ij}} = \frac{\partial Q}{\partial a_i}\frac{\partial a_i}{\partial W_{ij}} = (a_i - s_i)\frac{a_j}{R}, \tag{2.10}$$

où i est l'indice d'un neurone de sortie et j est l'indice d'un neurone caché.

$$\frac{\partial Q}{\partial c_{j,k}} = \frac{\partial Q}{\partial a_j}\frac{\partial a_j}{\partial c_{j,k}} \text{ et } \frac{\partial Q}{\partial \sigma_{j,k}} = \frac{\partial Q}{\partial a_j}\frac{\partial a_j}{\partial \sigma_{j,k}},$$

où j est l'indice d'un neurone caché.

$$\frac{\partial Q}{\partial a_j} = \sum_i \frac{\partial Q}{\partial a_i}\frac{\partial a_i}{\partial a_j} = \sum_i (a_i - s_i)\frac{w_{ij}R - \sum_k w_{ik}a_k}{R^2}, \tag{2.11}$$

où i parcourt les indices des neurones vers lesquels le neurone j envoie des connexions (neurones de sortie).

$$\frac{\partial a_j}{\partial c_{j,k}} = a_j \frac{e_k - c_{j,k}}{\sigma_{j,k}^2} \tag{2.12}$$

$$\frac{\partial a_j}{\partial \sigma_{j,k}} = a_j \frac{(e_k - c_{j,k})^2}{\sigma_{j,k}^3} \tag{2.13}$$

Il est fréquent de n'utiliser qu'un apprentissage sur les poids W_{ij} des connexions entre la couche cachée et la couche de sortie. Nous pouvons noter que la sortie du réseau est linéaire par rapport à ces paramètres, ce qui diminue le risque de blocage dans un minimum local de la fonction d'erreur lors de l'apprentissage [138]. Les variances et les centres des gaussiennes sont dans ce cas choisis de manière à couvrir uniformément le domaine d'entrée désiré.
D'autres algorithmes d'apprentissage adaptés aux réseaux à fonctions de bases radiales peuvent être trouvés dans l'article de M. T. Musawi et al. [106].

Propriétés Tout comme dans le cas des réseaux multi-couches il a été démontré que les réseaux à bases radiales sont des approximateurs universels de fonctions. Les deux modèles possèdent cependant des propriétés différentes.
Le calcul de l'activation d'un réseau de type RBF ne fait intervenir qu'un petit nombre de neurones, dont la zone d'influence comprend les données d'entrée.
Le traitement de l'information n'est plus distribué entre l'ensemble des unités comme dans les réseaux multi-couches. La modification d'un paramètre du réseau n'a qu'une

influence locale. Elle n'affecte que le traitement d'une partie du domaine d'entrée. L'apprentissage de nouveaux exemples après la phase initiale d'adaptation présente donc moins de risque de dégrader les performances globales du réseau. On parle de *généralisation locale* pour décrire cette propriété.
Ce modèle souffre cependant d'un inconvénient par rapport aux réseaux multi-couches puisque contrairement à ceux-ci, son domaine d'approximation (i.e. domaine dans lequel il réalise une approximation satisfaisante) est strictement borné. Ce dernier se limite en effet aux zones d'influence des neurones cachés, en dehors desquelles le réseau est incapable d'extrapoler. Lorsque la dimension ou la taille du domaine d'entrée est trop importantes, le nombre de neurones nécessaires peut devenir considérable et l'emploi de réseaux multi-couches peut être plus approprié.

2.1.2.3 RNA de type CMAC

Nous terminons cette présentation par les réseaux de type CMAC (pour Cerebellar Model Articulation Controler), introduits par J. S. Albus [5].

Structure Le fonctionnement de ce modèle est très simple et relativement éloigné de celui des autres modèles que nous avons présentés.
Il s'appuie sur plusieurs discrétisations différentes de l'espace d'entrée sous forme de grilles légèrement décalées les unes par rapport aux autres (voir figure 2.4).

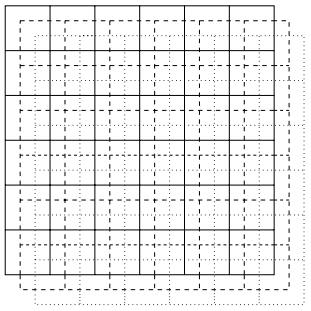

FIG. 2.4 – *Discrétisation d'un espace d'entrée à deux dimensions par un réseau de type CMAC.*

Chacun de ces «découpages» définit un ensemble de cellules (aussi appelées champs récepteurs) auxquelles sont associées les neurones d'entrée du réseau (chaque cellule est associée à un neurone).
Chacun de ces neurones possède un degré d'activation binaire (i.e. 0 ou 1) et s'active lorsque la valeur d'entrée présentée au réseau se situe dans la cellule qui lui correspond.
Le réseau se compose également d'un ensemble de neurones de sortie qui reçoivent des connexions pondérées depuis chaque unité d'entrée, et réalisent la moyenne des valeurs transmises par ces connexions.
L'apprentissage se fait simplement, dans ce réseau, en modifiant les poids des connexions d'une quantité proportionnelle à l'erreur de sortie.

Propriétés L'organisation des discrétisations est telle que les ensembles de cellules activés par deux valeurs d'entrée voisines possèdent un grand nombre d'éléments communs. Les sorties du réseau correspondant à ces deux entrées sont donc également assez proches puisqu'une partie des poids intervenant dans leur calcul est la même dans les deux cas.
Au contraire, les neurones activés par deux entrées très dissemblables sont tous différents, et les deux valeurs de sortie correspondant pourront donc être très éloignées.
Cette caractéristique est obtenue grâce au décalage successif des grilles (voir figure 2.5).
Cette topologie donne au réseau une propriété de généralisation locale identique à celle des réseaux de type RBF. La modification du poids associé à une connexion provenant d'un neurone donné, n'agit en effet sur la sortie calculée par le réseau que pour les valeurs d'entrée activant ce neurone.

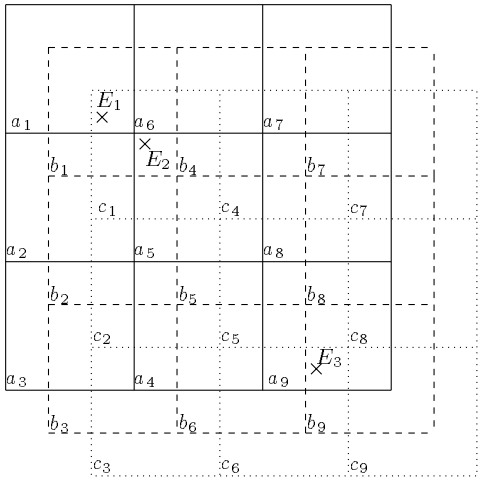

FIG. 2.5 – *Propriété de généralisation locale d'un réseau CMAC. Les calculs des valeurs correspondant aux entrées E_1 et E_2 font tous deux intervenir les poids associés aux cellules b_1 et c_1 mais différent par l'utilisation des poids associés aux cellules a_1 et a_5. Les entrées E_1 et E_3 n'activent par contre aucune cellule commune.*

Nous pouvons noter que le formalisme des RNA n'est pas du tout essentiel pour ce type de réseau dont le fonctionnement peut être vu comme l'application d'un simple mécanisme d'interpolation entre différentes tables de mémoire.

2.1.3 Le dilemme du biais et de la variance

Cette section présente un problème central dans l'utilisation des différents modèles de RNA.
Comme nous l'avons vu, ceux-ci sont essentiellement des procédures permettant d'approcher n'importe quelle fonction linéaire ou non, à partir d'exemples de valeurs de cette fonction. La réussite de cet apprentissage suppose cependant qu'un certain nombre de

conditions, portant sur le choix et le nombre de ces exemples ainsi que sur l'architecture initiale du réseau, soient vérifiées.
Cet aspect est particulièrement important si l'on cherche à garantir des capacités de généralisation au réseau.
Soit $F(X)$ la fonction que l'on cherche à approcher et $F_W(X)$ la fonction réalisée par le réseau utilisé (W est l'ensemble des paramètres libres du réseau). Il a été prouvé pour plusieurs modèles de réseaux que moyennant le choix d'un nombre suffisant de neurones, il existe un jeu de paramètres W^* tel que $F_{W^*}(X)$ est une approximation de $F(X)$, au degré de précision désiré [64, 30].
Dans la plupart des applications, $F(X)$ n'est connu que pour un ensemble de valeurs de X limité (les exemples d'apprentissage). La détermination exacte de W^* est donc le plus souvent impossible, même si le nombre de neurones du réseau est suffisant.
L'utilisation d'un algorithme d'apprentissage tel que par exemple la rétropropagation du gradient permet toutefois, à partir de l'observation d'un certain nombre de valeurs d'entrée/sortie, $S = \{(X_i, F(X_i)), i = 1...N\}$, d'estimer W^* en minimisant une mesure d'erreur $E_S(W)$[1] sur l'ensemble S.
Cette approche permet d'obtenir un jeu de paramètres W_S donnant un minimum local ou global de $E_S(W)$. La différence entre la fonction $F(X)$ et la fonction $F_{W_S}(X)$ réalisée par le réseau, appelée erreur de généralisation, peut être décomposée en deux termes nommés *biais* et *variance* [87] (voir figure 2.6).
Le premier terme (i.e. le biais) ne dépend que de l'architecture du réseau et décroît lorsque le nombre de paramètres libres et donc le nombre d'unités augmentent.
Le deuxième terme (i.e. la variance) dépend de la taille de l'ensemble S. Pour garantir un bon résultat, ce dernier doit être d'autant plus grand que le nombre de paramètres libres est important.
La réalisation d'une bonne approximation demande donc le choix d'une architecture appropriée en terme de nombre d'unités, afin de diminuer la contribution du biais à l'erreur. D'un autre coté si le nombre de paramètres libres du réseau (i.e. le nombre d'unités) est important, la taille de la base d'exemples d'apprentissage doit également être importante afin de diminuer la variance.
Cet aspect est particulièrement important si la taille de la base d'exemples est fixe et relativement faible. Privilégier la réduction du biais en augmentant la taille du réseau peut conduire alors à des systèmes *sur-paramétrés*, pour lesquels les exemples d'apprentissage sont appris «par cœur» aux dépens des capacités de généralisation [13] (voir figure 2.7). Au contraire, en réduisant la taille du réseau, on risque d'obtenir une architecture *sous-paramétrée* ne permettant pas d'approcher une fonction complexe et pour laquelle même un grand nombre d'exemples n'autorise pas un apprentissage satisfaisant[2]. L'équilibre entre taille du réseau et nombre d'exemples disponibles est donc relativement difficile à obtenir. Ce problème est connu sous le nom de *dilemme du biais et de la variance* en statistique.

1. Il peut s'agir par exemple de l'erreur quadratique décrite par l'équation 2.1.

2. Une démonstration plus détaillée des points abordées dans cette section peut être trouvée dans le thèse de A. Labbi [87].

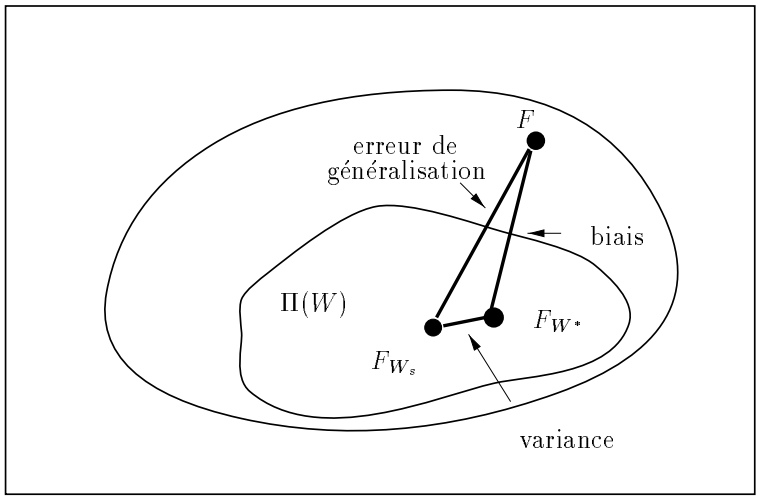

FIG. 2.6 – *Décomposition de l'erreur de généralisation en biais et variance.* $\Pi(W)$ *est l'ensemble des fonctions implémentées par le réseau (paramétrées par* W*). Cette figure est tirée de [88].*

Il est lié au fait que les modèles de RNA sont totalement indépendants du type de la fonction que l'on cherche à approcher. Tout ce qu'ils apprennent provient des exemples. L'utilisation de connaissances préalables sur le type de fonction que l'on cherche à

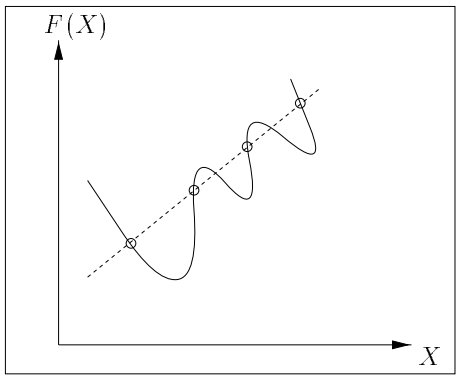

FIG. 2.7 – *Un exemple d'approximation par un réseau sur-paramétré. Les exemples, notés par des cercles sont parfaitement appris, mais la fonction réalisée par le réseau (ligne continue) est très différente de celle que l'on cherche à approcher (tirets).*

modéliser a donc été la motivation de nombreux travaux sur les *systèmes hybrides neuro-symboliques* (voir par exemple [153]). L'utilisation d'une architecture de réseau contrainte et adaptée à une application particulière peut en effet permettre de diminuer le biais sans augmenter le nombre de paramètres libres du réseau [2]. C'est cette approche que cherchent à exploiter les systèmes hybrides neuro-flous qui sont décrits dans la section suivante.

2.2 Logique Floue et RNA neuro-flous

Nous présentons dans cette section certains modèles de RNA qui intègrent des aspects issus de la logique floue. On parle généralement de *systèmes hybrides neuro-flous* pour décrire ce type de modèles. Nous débutons par un exposé des principes de base de la logique floue.

2.2.1 Logique floue

2.2.1.1 Notion d'ensemble flou

La Logique floue introduite par L. A. Zadeh [170] est particulièrement séduisante en raison de sa capacité à produire un raisonnement sur des termes proches du langage courant, et à manipuler des informations souffrant d'imprécisions et/ou d'incertitudes. Il s'agit d'une logique multivaluée basée sur la notion d'ensemble flou[1], qui est une extension de la théorie classique des ensembles. Dans cette dernière tout sous-ensemble X d'un univers U peut être défini par une fonction caractéristique binaire, associant la valeur 1 aux éléments de U appartenant à X et la valeur 0 à tous les autres. Un sous-ensemble flou[2] X est défini par une fonction d'appartenance qui associe à tout élément u de U un nombre réel $\mu_X(u)$ compris entre 0 et 1, appelé degré d'appartenance de u à X. Autrement dit un élément peut appartenir à un sous-ensemble flou de manière graduelle, ce qui rompt avec le tout ou rien de la théorie ensembliste classique.

2.2.1.2 Proposition floue

L'association de sous-ensembles flous à des termes linguistiques définis sur un univers de discours quelconque, autorise la représentation d'informations plus ou moins spécifiques et précises [41].
Une variable linguistique peut être définie comme l'association d'une variable classique (par exemple la distance des obstacles détectés par un capteur) et de plusieurs sous-ensembles flous caractérisant les valeurs possibles de celle-ci. On appelle alors *proposition floue* élémentaire une proposition du type X **est** A, où X est une variable linguistique et A un sous-ensemble flou. Une telle proposition possède un degré de vérité $\mu_A(x)$, compris entre 0 et 1, où μ_A est la fonction caractéristique de A et x est la valeur réelle de la variable X (voir figure 2.8). Il est ainsi possible de définir des assertions du type : des obstacles sont proches, où proche est un sous-ensemble flou.
Ces propositions élémentaires peuvent être combinées à l'aide d'opérateurs logiques binaires (**conjonction**, **disjonction**, **implication**) ou unaire (**négation**). Le degré

1. Théorie ensembliste et logique sont étroitement liées.

2. Nous emploierons par abus de langage indifféremment les termes ensemble flou et sous-ensemble flou. C'est plutôt ce dernier terme qui devrait être employé en permanence puisque l'on cherche à désigner une partie d'un univers plus vaste.

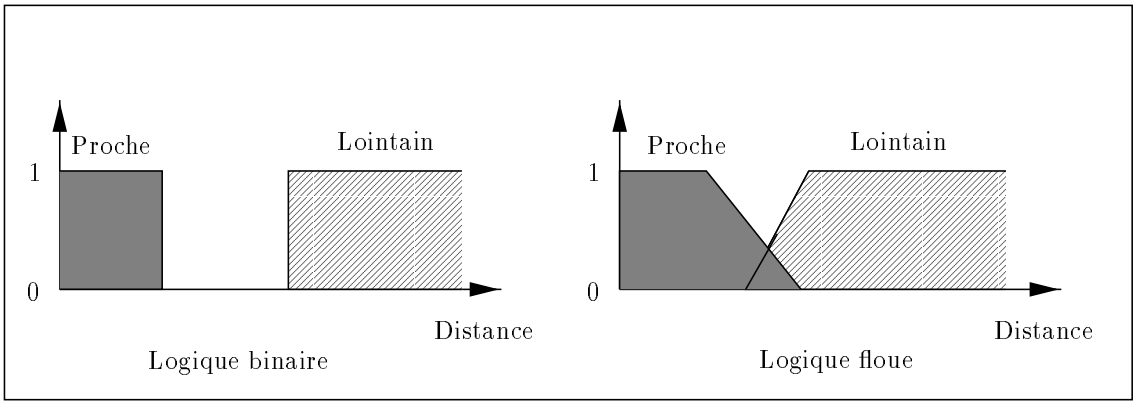

FIG. 2.8 – *Exemple de fonctions caractéristiques de deux ensembles en logique binaire (gauche) et en logique floue (droite).*

de vérité des nouvelles propositions obtenues peut être calculé entre autres par les équations suivantes :

- conjonction : (X est A) et (Y est B)
  $minimum(\mu_A(x), \mu_B(y))$ (Logique de Zadeh)
  $\mu_A(x) \times \mu_B(y)$ (Logique probabiliste)
  $maximum(\mu_A(x) + \mu_B(y) - 1, 0)$ (Logique de Lukasiewicz)
- disjonction : (X est A) ou (Y est B)
  $maximum(\mu_A(x), \mu_B(y))$ (Logique de Zadeh)
  $\mu_A(x) + \mu_B(y) - \mu_A(x) \times \mu_B(y)$ (Logique probabiliste)
  $minimum(\mu_A(x) + \mu_B(y), 1)$ (Logique de Lukasiewicz)
- négation : (X est non A)
  $1 - \mu_A(x)$

Dans ces expressions, x et y représentent les valeurs réelles des variables linguistiques X et Y, et μ_A et μ_B sont les fonctions d'appartenance des sous-ensembles A et B.

L'opérateur d'implication permet d'introduire la notion de règle floue qui caractérise les relations de dépendance entre plusieurs propositions floues quelconques :

$$(X_1 \text{ est } A_1) \text{ et } (X_2 \text{ est } A_2) \text{ implique } (Y \text{ est } B),$$

où X_1, X_2 et Y sont des variables linguistiques et A_1, A_2 et B sont des sous-ensembles flous. Cette règle peut également être exprimée sous une forme plus classique :

$$\text{Si } (X_1 \text{ est } A_1) \text{ et } (X_2 \text{ est } A_2) \text{ alors } (Y \text{ est } B)$$

Dans cette dernière formulation, la partie (X_1 est A_1) et (X_2 est A_2) est appelée prémisse de la règle et la partie (Y est B) est appelée conclusion.

2.2.1.3 Système d'inférence flou

La notion de règle floue permet de définir un système expert flou comme une extension d'un système expert classique, manipulant des propositions floues. Une des principales applications de ce type de système concerne le domaine de la commande. Il est en effet aisé, en définissant des sous-ensembles flous sur les variables d'état du système commandé ainsi que sur les variables de commande, de traduire la connaissance que peut avoir un expert humain sur la manière dont le processus doit être commandé.

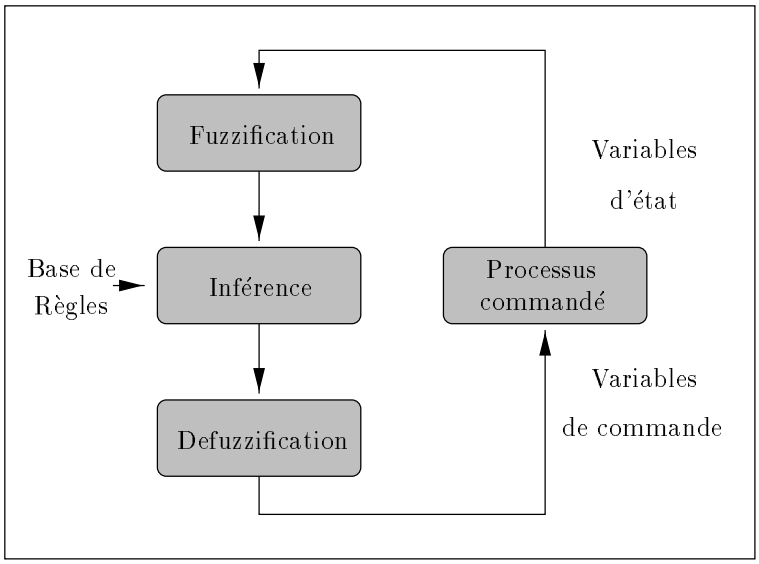

FIG. 2.9 – *Système de commande flou.*

Un système de commande flou (ou contrôleur flou), tel qu'il est défini par E. H. Mamdani [97] procède en trois étapes principales (voir figure 2.9) :

- la «fuzzification»[1] qui consiste à associer à chaque valeur d'entrée un ou plusieurs sous-ensembles flous ainsi que les degrés d'appartenance associés. Cette étape réalise la transformation de valeurs numériques en informations symboliques floues ;

- la phase d'inférence qui consiste à calculer le degré de vérité des différentes règles du système, en utilisant les formules données précédemment, et à associer à chacune de ces règles une valeur de sortie. Cette valeur de sortie dépend de la partie conclusion des règles qui peut prendre plusieurs formes. Il peut s'agir d'une proposition floue, et l'on parlera dans ce cas de *règle de type Mamdani*:

$$\text{Si } \ldots. \text{ alors } Y \text{ est } B$$

1. Il n'existe pas à notre connaissance de traduction admise pour cet anglicisme que nous utiliserons donc dans ce mémoire.

Il peut également s'agir d'une fonction réelle des entrées, et l'on parlera dans ce cas de *règle de type Sugeno*[1] :

$$\text{Si .... alors } Y \text{ est } f(x_1, x_2, ..., x_n),$$

où $x_1, .., x_n$ sont les valeurs réelles des variables d'entrée.

Dans ce dernier cas la valeur de sortie de la règle est tous simplement donnée par : degré de vérité de la prémisse $\times$ $f(x_1, x_2, ..., x_n)$.

Dans le cas d'une règle de type Mamdani, la sortie est un sous-ensemble flou obtenu à partir de celui présent dans la partie conclusion de la règle, soit en lui appliquant un facteur d'échelle égal au degré de vérité de la prémisse, soit en le tronquant à la valeur de ce degré de vérité (voir figure 2.10). On parle dans le premier cas d'inférence PRODUIT et dans le deuxième d'inférence MINIMUM.

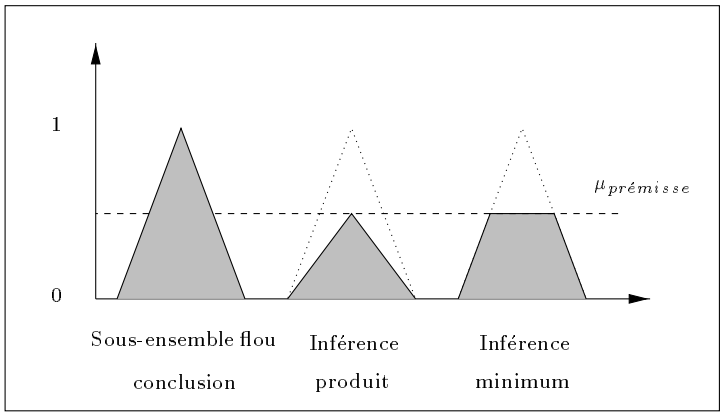

FIG. 2.10 – *Inférence PRODUIT et inférence MINIMUM. Le degré de vérité de la prémisse de la règle est noté* $\mu_{prémisse}$.

– la «défuzzification»[2] qui a pour but l'obtention d'une valeur numérique pour chaque variable de sortie à partir des valeurs de sortie des différentes règles. Dans le cas de règles de type Sugeno, le calcul se fait simplement par une somme normalisée des valeurs associées aux règles. Dans le cas de règles de type Mamdani, une valeur numérique doit être obtenue à partir de l'union des sous-ensembles flous correspondant aux différentes conclusions. Parmi les nombreuses possibilités pour réaliser cette étape, nous pouvons citer entre autres la méthode du centre de gravité, la moyenne des maxima, la bissectrice de la zone ou encore le plus petit ou le plus grand maximum, qui sont présentés sur la figure 2.11.

Il est également possible de définir un poids pour chaque règle de manière à leur donner des influences différentes dans le calcul de la valeur numérique de sortie.

1. On parle également de règle de type Sugeno d'ordre 0 lorsque la sortie est une constante.

2. Tout comme dans le cas de la Fuzzification, nous n'avons pas trouvé de traduction satisfaisante pour ce terme.

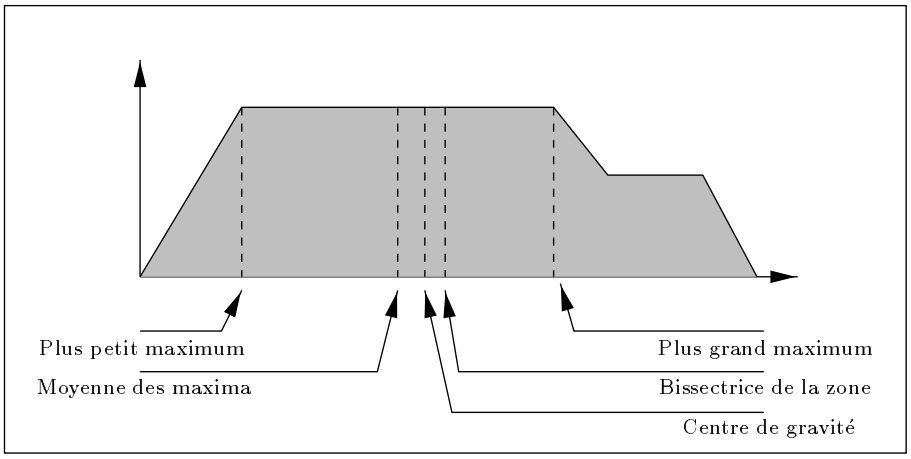

FIG. 2.11 – *Stratégies de défuzzification à partir de l'union de plusieurs sous-ensembles flous.*

Une description plus complète des systèmes de commande flous pourra être trouvée dans la thèse de P. Garnier [41] dont s'inspire en partie cette section, ou dans les articles de référence de C. C. Lee [92, 93].

2.2.2 Systèmes hybrides neuro-flous

Il apparaît au travers des présentations que nous venons de faire que les réseaux de neurones artificiels et la logique floue peuvent être complémentaires sur plusieurs points. La logique floue permet une spécification rapide des tâches à accomplir à partir de la connaissance symbolique disponible. Le réglage précis du système obtenu et l'optimisation de ses différents paramètres reste néanmoins beaucoup plus difficile dans de nombreux cas (voir §4.2.2 pour des exemples en robotique mobile).
Les modèles les plus courants de RNA, au contraire, n'autorisent pas l'incorporation de connaissance *a priori* mais permettent de régler par apprentissage le comportement précis du système.
De nombreux auteurs ont donc tout naturellement cherché à combiner ces deux paradigmes depuis le début des années 90 et ceci de plusieurs manières [100, 96].
Nous portons ici notre attention sur les approches permettant de représenter sous forme de RNA les règles d'un système d'inférence flou. Cette technique consiste à utiliser des fonctions d'activation particulières pour les unités du réseau, ainsi qu'une organisation spécifique de ces dernières et ce afin de reproduire les différents éléments constitutifs d'un système d'inférence flou.
La structure du réseau est ainsi choisie en fonction de la forme de la fonction que l'on cherche à approcher. Les dépendances entre les variables d'entrée et de sortie sont en effet spécifiées par le choix des règles floues. Les différents paramètres de ces règles (i.e. forme et position des sous-ensembles flous, sortie et poids des règles), peuvent ensuite être modifiés par un algorithme d'apprentissage supervisé. Nous pouvons noter que cela

ne restreint en rien l'ensemble des fonctions approchables puisqu'il a été démontré que la logique floue présente comme les RNA des propriétés d'approximation universelle de fonction [80].
Si l'on adopte le point de vue des systèmes flous, cette approche permet de régler de manière précise, par apprentissage, le comportement du système réalisé. Si l'on se place au contraire du coté des RNA, la connaissance sous forme de règles floues permet de choisir l'architecture du réseau en fonction de la tâche à accomplir.
On retrouve dans la littérature principalement deux modèles de RNA utilisés pour le codage de systèmes d'inférence flous : les réseaux multi-couches et les réseaux à fonctions de base radiales.

2.2.2.1 Réseaux neuro-flous à fonctions de base radiales

L'équivalence entre un système d'inférence flou utilisant des règles de type Sugeno et un réseau de type RBF est assez intuitive [69].
Les fonctions gaussiennes définissant les activations des unités d'entrée du réseau peuvent également être utilisées pour représenter les sous-ensembles flous. En utilisant la logique probabiliste (voir §2.2.1.2), le degré de vérité de la prémisse d'une règle ne contenant que l'opérateur conjonction peut être calculé par un neurone caché d'un réseau RBF (voir équation 2.7). La prémisse Si (X est A) et (Y est B) et (Z est C), est dans ce cas représentée par un neurone possédant la fonction d'activation suivante :

$$a(x,y,z) = \exp(-\frac{(x-c_a)^2}{\sigma_a^2}) \times \exp(-\frac{(y-c_b)^2}{\sigma_b^2}) \times \exp(-\frac{(z-c_c)^2}{\sigma_c^2}),$$

où x, y et z sont respectivement les valeurs des variables X, Y et Z, et $c_a, c_b, c_c, \sigma_a, \sigma_b$ et σ_c sont les paramètres des fonctions d'appartenance des sous-ensembles flous A, B et C.
Chaque sous-ensemble flou est ainsi codé par une dimension de l'entrée du neurone.
Si les règles utilisées sont de type Sugeno d'ordre 0 [1], leurs conclusions sont tout simplement représentées par les poids des connexions vers la couche de sortie. Les unités de cette couche concordent avec les sorties du système flou et réalisent par leur fonction d'activation une somme normalisée qui correspond à la défuzzification [2] :

$$sortie = \frac{\sum_j w_j a_j}{\sum_j a_j}, \tag{2.14}$$

où j parcourt toutes les règles dont a_j représente le degré de vérité et w_j la sortie.
Cette équivalence parfaite permet d'étendre l'utilisation des algorithmes d'apprentissage définis pour les réseaux de type RBF aux systèmes d'inférence flous.
Cette approche impose cependant plusieurs restrictions. Si le même sous-ensemble flou apparaît dans les prémisses de deux règles différentes, il est représenté par deux neu-

1. Règles dont la sortie est une valeur réelle constante.
2. On parle de défuzzification par la méthode du centre de gravité.

rones distincts dont les paramètres peuvent évoluer différemment lors de l'apprentissage. De plus l'utilisation de règles plus complexes comportant notamment des disjonctions ou des négations est impossible avec ce formalisme [58]. Leur représentation demande la modification de la structure du réseau et l'on tend alors vers le modèle présenté dans la section suivante.

2.2.2.2 Réseaux neuro-flous multi-couches

le deuxième choix possible consiste à utiliser un réseau multi-couches pour lequel chaque couche correspond à la réalisation d'une étape d'un système d'inférence flou. Une architecture classique peut être décrite de la manière suivante (voir figure 2.12) :

- la couche d'entrée reçoit les valeurs des variables d'entrée ;
- les unités de la ou des premières couches cachées calculent par leur fonction d'activation le degré de vérité des différents sous-ensembles flous (fuzzification) ;
- la couche suivante correspond au calcul du degré de vérité des prémisses des différentes règles ;
- la ou les dernières couches du réseau réalisent la phase de défuzzification et fournissent les valeurs de sortie.

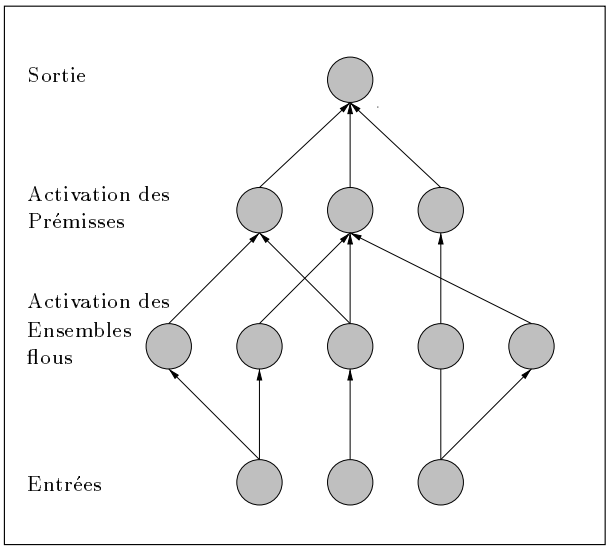

FIG. 2.12 – *Un exemple d'architecture de réseau neuro-flou.*

Plusieurs réalisations de ces différentes étapes sont possibles. Il est important de noter que l'utilisation d'un algorithme d'apprentissage de type descente de gradient (e.g. la rétropropagation du gradient) impose l'emploi de fonctions d'activation dérivables pour l'ensemble des unités du réseau.

Codage des sous-ensembles flous La première couche d'une architecture de ce type comporte autant de neurones qu'il y a de sous-ensembles flous dans le système d'inférence représenté. Chaque unité calcule le degré de vérité d'un sous-ensemble particulier par sa fonction de transfert. La seule restriction sur le choix de cette fonction concerne sa dérivabilité. On retrouve généralement dans la littérature, l'utilisation de fonctions gaussiennes similaires à celles des réseaux RBF. C'est notamment le cas dans l'architecture ANFIS de J-S. R. Jang [70] ou dans les travaux de P-Y. Glorennec [45]. Afin de conserver le formalisme des RNA, les paramètres modifiables (i.e. centre et pente de la gaussienne) sont codés par les poids de connexions provenant d'un neurone dont la sortie reste fixée à 1.
Certains auteurs choisissent également d'utiliser plusieurs unités réparties sur deux couches [63, 52]. La fonction d'appartenance est dans ce cas codée par le minimum ou la différence de deux sigmoides. Elle présente toujours la forme d'une gaussienne mais présente l'avantage de pouvoir utiliser deux pentes moyennes différentes à gauche ou à droite du centre.

Calcul du degré d'activation des prémisses Les neurones de la deuxième couche cachée représentent chacun la prémisse d'une règle. Ils reçoivent en entrée le degré de vérité des différents sous-ensembles flous composant cette prémisse et ont en charge le calcul de son propre degré de vérité. Les fonctions d'activation utilisées pour ces neurones dépendent des opérateurs présents dans les règles (conjonction ou disjonction) et de la logique utilisée :

- logique de Zadeh : les opérateurs minimum et maximum ne sont pas dérivables. Ils peuvent cependant être approchés par des fonctions dérivables, appelées Softmin et Softmax, que l'on retrouve dans [52, 111, 14] :

$$SOFTMIN(x,y) = \frac{x\exp(-Kx) + y\exp(-Ky)}{exp(-Kx) + exp(-Ky)}$$

$$SOFTMAX(x,y) = \frac{x\exp(Kx) + y\exp(Ky)}{exp(Kx) + exp(Ky)}$$

  Dans ces deux équations, K est une constante qui doit être choisie la plus grande possible. La figure 2.13 compare les fonctions minimum(x,y) et softmin(x,y) pour $K = 10$;

- logique de Lukasiewicz : l'opérateur de conjonction peut être obtenu à l'aide de l'approximation de la fonction $maximum(0,x)$ par une sigmoide [45](voir figure 2.14) :

$$Maximum(0,x) \approx \frac{1}{1+\exp(\frac{-(x-0.5)}{0.227})}$$

  L'opérateur de disjonction peut être obtenu par une méthode similaire ;

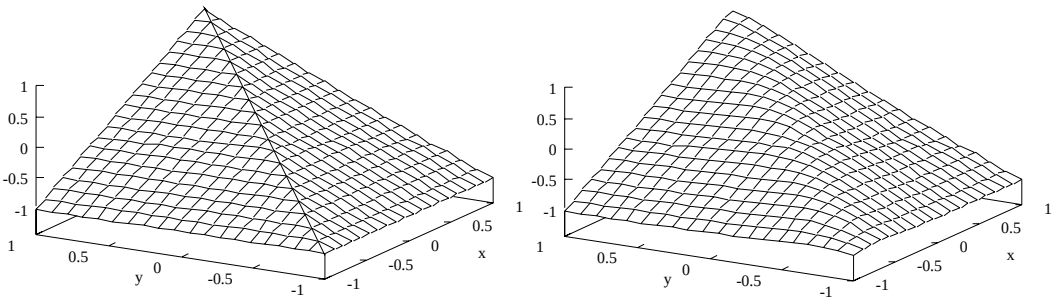

FIG. 2.13 – *Fonctions minimum(x,y) et softmin(x,y).*

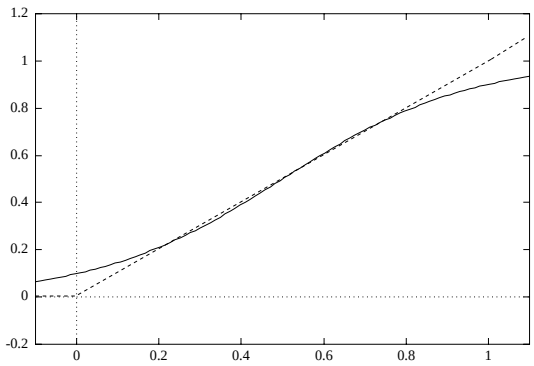

FIG. 2.14 – *Approximation de la fonction* $Maximum(0, x)$ *(en pointillés) par une sigmoide (en ligne continue).*

– logique probabiliste : les opérateurs de la logique probabiliste sont dérivables et ne posent donc pas de problèmes. On les retrouve dans [70, 54].

Inférence et défuzzification Ces deux phases sont généralement réalisées par les unités de la ou des dernières couches du réseau. Les fonctions de transfert utilisées dépendent du type de règles choisi :

– règles de type Mamdani : les nombreuses méthodes de défuzzification proposées pour les règles de ce type ne peuvent généralement pas être exprimées par une expression dérivable. H. R. Berenji et P. Khedar [14] fournissent une solution particulière dans le cas où les fonctions d'appartenance des sous-ensembles flous apparaissant dans les conclusions sont des fonctions triangulaires. Une première couche de neurones calcule une valeur numérique pour chaque règle en utilisant l'expression suivante :

$$sortie = S + \frac{1}{2}(L_d - L_g)(1 - \mu_{prémisse}),$$

où S est l'abscisse du sommet de la fonction d'appartenance triangulaire, L_d et L_g sont respectivement les longueurs des parties du sous-ensemble situées à

droite et à gauche de S, et $\mu_{prémisse}$ est le degré de vérité de la prémisse de la règle. Cette valeur correspond à la moyenne de la zone pour laquelle le sous-ensemble flou possède une valeur maximum (voir figure 2.15).

Une dernière couche de neurones (la couche de sortie) réalise une somme normalisée des valeurs ainsi obtenues pour les différentes règles. Cette méthode de défuzzification est baptisée moyenne locale des maxima (LMOM).

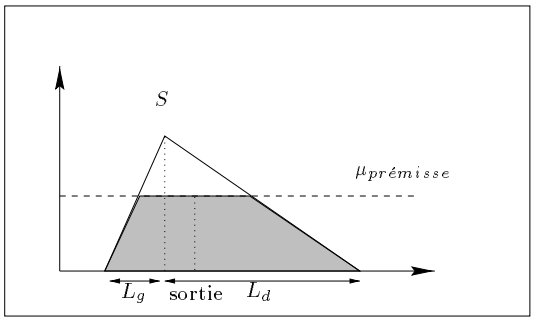

FIG. 2.15 – *Défuzzification par la moyenne locale des maxima. Le sous-ensemble flou correspondant au résultat de la règle est représenté en grisée.*

- règles de type Sugeno d'ordre 0 : ces règles sont les plus fréquemment rencontrées dans les applications neuro-floues [70]. Puisque la sortie est uniquement une constante réelle, il suffit de coder sa valeur par le poids de la liaison entre le neurone calculant le degré de vérité de la prémisse et un neurone calculant la valeur de sortie. Ce dernier calcule encore une fois une somme normalisée des valeurs des différentes règles comme décrit par l'équation 2.14.

- règles de type Sugeno d'ordre supérieur : lorsque la sortie des règles utilisées est une fonction des entrées, il suffit de rajouter une couche de neurones calculant cette fonction [70].

- règles de type Tsukamoto [161] : ces règles sont un cas particulier des règles de type Mamdani pour lesquelles les sous-ensembles flous utilisés en conclusion utilisent des fonctions d'appartenance monotones. Il suffit ici d'utiliser des neurones codant l'inverse de ces fonctions et recevant en entrée le degré de vérité des prémisses pour obtenir une valeur numérique pour chaque règle. Encore une fois les neurones de sortie réalisent une somme normalisée des valeurs des différentes règles. On retrouve notamment ce type de règle dans le système NEFCON de D. Nauck et R. Kruse [111].

Apprentissage : Dans la majorité des cas rencontrés la stratégie d'apprentissage repose sur une adaptation de l'algorithme de rétropropagation du gradient. On trouve cependant quelques autres exemples. S. M. Sulzberger et al. [151] utilisent une méthode de perturbation aléatoire des poids du réseau et ne gardent que les modifications qui

améliorent les performances. Si une telle approche ne garantit absolument pas la convergence vers un système optimal, elle possède l'avantage de n'imposer aucune contrainte sur la structure du réseau.
Berenji et Khedar [14] utilisent dans le système GARIC un algorithme d'apprentissage par renforcement similaire à ceux décrits dans la section 2.5.3.

2.3 Motivations pour l'emploi des réseaux de neurones artificiels en commande

La commande des systèmes (processus) linéaires, ou pouvant être facilement approchés par des systèmes linéaires, est aujourd'hui un domaine bien maîtrisé par les automaticiens. Les outils de l'algèbre linéaire permettent d'obtenir pour ces systèmes, des contrôleurs possédant des propriétés de stabilité[1] et d'optimalité. Dans le monde réel cependant un grand nombre de processus sont caractérisés par un comportement dynamique non linéaire complexe, et rendent impossible l'utilisation des outils classiques de l'automatique. Il en est de même pour les systèmes pour lesquels les modèles mathématiques connus sont incomplets ou de mauvaise qualité. Il n'existe pas aujourd'hui de théorie systématique et applicable de manière générale à la commande de tels processus. Pour résoudre ce problème une des solutions proposées consiste à avoir recours à une phase d'apprentissage pour identifier le modèle du processus ou le contrôleur. Le terme «apprentissage» désigne ici le fait de modifier la structure et/ou les paramètres du système, de manière à améliorer ses performances futures, en se basant sur des observations expérimentales passées [10]. Nous avons vu plus haut que les réseaux de neurones artificiels sont principalement des procédures permettant d'approcher n'importe quelle fonction linéaire ou non [56]. C'est cette propriété qui motive leur utilisation pour la réalisation de systèmes de commande non linéaires par apprentissage.

2.3.1 Apprentissage en commande et commande adaptative

De nombreuses méthodes de *commande adaptative* ont été développées par les automaticiens pour permettre au contrôleur d'évoluer (de s'adapter) en fonction de l'évolution de la tâche à résoudre (voir par exemple [109]). Ces méthodes permettent, une fois choisie la structure du contrôleur, de régler un certain nombre de paramètres de ce dernier. Si le principe général mis en œuvre dans ces algorithmes est similaire à l'apprentissage réalisé par les RNA, l'adaptation se résume à un simple réglage d'un très petit nombre de coefficients de la boucle de commande sans capacité de mémorisation. L'utilisation de systèmes ayant des possibilités d'apprentissage plus importantes, tels que les RNA, permet de gagner des fonctionnalités de plus haut niveau, comme la prise de décision ou la reconnaissance de situation [27]. On parle dans ce cas *d'apprentissage en commande* plutôt que de *commande adaptative*.

1. La notion de stabilité sera précisée plus loin dans cette section.

Dans le cas d'un processus confronté à deux reprises aux mêmes conditions d'utilisation après un certain intervalle, un système de commande adaptative sera incapable de se remémorer les paramètres précédemment choisis et devra recommencer son adaptation. Un contrôleur réalisant un apprentissage de plus haut niveau pourra lui reconnaître une situation déjà rencontrée et réutiliser les paramètres mémorisés.
Un système d'apprentissage en commande mémorise les paramètres associés aux différents points de fonctionnement du processus, alors qu'un système de commande adaptative doit les redécouvrir chaque fois que les conditions de fonctionnement changent.
Les méthodes de commande adaptative nécessitent de plus la connaissance *a priori* de la structure du contrôleur utilisé alors que les RNA peuvent fonctionner, moyennant un certain nombre d'hypothèses sur la complexité du problème et sur les modalités de l'apprentissage (voir section 2.1.3), sans informations *a priori*.
Le but recherché lors de la réalisation d'un contrôleur par apprentissage n'est généralement pas de stocker les données nécessaires à la commande dans une mémoire, la plus grande possible, dans laquelle on piocherait par similitude de situations. Il est plus souvent souhaitable de pouvoir inférer à partir de ces données, la structure ou le modèle sous-jacent.
Les différents modèles de réseaux neuronaux permettent justement de couvrir l'étendue des solutions séparant la constitution d'une simple table de mémoire, de l'apprentissage des paramètres d'un modèle fortement contraint [11]. Cette opposition entre mémoire et modèle est un des thèmes centraux de nombreux problèmes d'apprentissage.

2.3.2 Quelques propriétés

Les RNA possèdent plusieurs caractéristiques intéressantes pour la réalisation de systèmes de commande :

- la parallélisation du traitement leur confère une grande rapidité de calcul et les rend très adaptés aux applications temps réel. Ceci est d'autant plus vrai lorsque l'on s'intéresse à des réalisations réellement parallèles sur des machines multiprocesseurs. Pour un bon aperçu de ce type d'implémentations le lecteur pourra se rapporter par exemple à l'article de H. Paugam-Moisy [125] ;

- le caractère distribué et fortement redondant du traitement réalisé leur donne une bonne résistance aux pannes internes. Certains auteurs se sont intéressés à l'étude de l'évolution de leurs performances lorsque certains neurones ou certaines connexions sont supprimés arbitrairement. R. Velazco étudie par exemple les possibilités d'utilisation des RNA dans des applications spatiales embarquées pour lesquelles les circuits électroniques sont dégradés par les rayonnements cosmiques [9] ;

- leur capacité de généralisation leur confèrent une bonne résistance aux bruits. Ceci est particulièrement important lorsque les capteurs permettant de mesurer l'état ou la sortie du processus commandé sont sujets à des perturbations.

C'est notamment le cas en robotique, où l'une des grandes difficultés concerne l'obtention de données sensorielles de bonne qualité.

2.3.3 RNA et stabilité

Les RNA souffrent par contre, dans leur comparaison avec d'autres méthodes de commande (notamment celles issues de l'automatique), d'un manque de résultats théoriques concernant la stabilité du système commandé. Il nous semble nécessaire, avant d'aller plus loin, de faire le point sur le terme de stabilité qui peut prendre plusieurs sens dans le cadre d'un système de commande réalisé par apprentissage.
La première signification possible concerne l'évolution du réseau et de ses sorties au cours de l'apprentissage. On cherche dans ce cas à garantir que l'erreur reste bornée et décroissante durant la phase d'entraînement. Ceci est particulièrement important lorsque l'on réalise un apprentissage en-ligne.
Une définition plus classique de la stabilité concerne l'ensemble de la boucle de commande. On cherche dans ce cas à garantir de manière théorique, que quelles que soient les perturbations que subit le processus, le système de commande est apte à le faire converger vers un état ou une suite d'états donné. Peu d'auteurs ont cherché à proposer une preuve de stabilité pour des systèmes de commande basés sur les RNA. Les quelques résultats disponibles [138, 94] ne concernent que des problèmes très spécifiques et sont difficilement généralisables. Nous pouvons cependant espérer que ceci n'est dû qu'à la jeunesse de cette discipline et que des travaux futurs viendront apporter des solutions à ce problème.
Il est également possible de relativiser cette limitation, car si la stabilité ne peut être prouvée de manière théorique, la reproduction des résultats expérimentaux peut suffire dans certains cas. La notion de stabilité peut de plus perdre de son importance dans un grand nombre d'applications où elle est plus difficile à définir. Que devient elle en effet lorsqu'on s'intéresse à la commande d'un robot cherchant à éviter des obstacles dans un environnement inconnu?

2.4 Le problème de l'apprentissage avec maître distant

Nous présentons dans cette section le problème principal qui se pose généralement lors de l'utilisation des RNA en commande. Les notations employées sont définies dans le tableau 2.1 et la figure 2.17. Nous les utiliserons jusqu'à la fin de ce mémoire.
Les données généralement disponibles pour la réalisation d'un système de commande par apprentissage sont les couples associant la sortie (ou l'état) $y(t)$ du processus à la sortie (ou l'état) désiré $y^*(t)$. On peut ainsi mesurer l'erreur ε_y (différence entre la valeur de sortie désirée et la valeur mesurée) obtenue en sortie du système après application de la commande. Il n'est par contre pas possible d'obtenir directement

l'erreur ε_u en sortie du contrôleur c'est-à-dire la différence entre la commande idéale, qui n'est pas connue, et la commande $\hat{u}(t)$ préconisée par le contrôleur.

C'est pourtant cette erreur qui est nécessaire pour réaliser l'adaptation du contrôleur par une méthode d'apprentissage supervisé classique (e.g. la rétropropagation du gradient).

Ce problème (voir figure 2.16) est décrit par M. Jordan [74] sous le nom de problème de l'apprentissage avec maître distant.

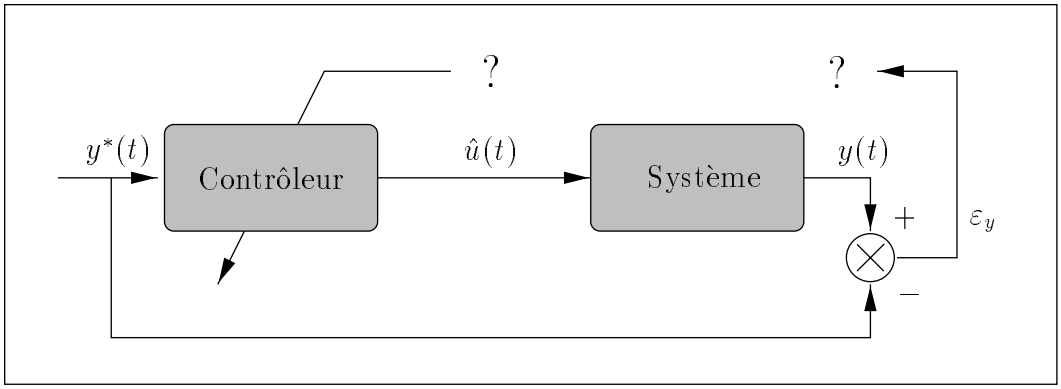

FIG. 2.16 – *Le problème de l'apprentissage distant. L'erreur en sortie du processus n'est pas directement exploitable, et l'erreur en sortie du contrôleur est inconnue.*

Jordan développe une analogie avec un joueur de basket cherchant à améliorer ses capacités dans le tir au panier. La difficulté consiste à traduire une erreur, mesurée en termes de distance entre le ballon et le panier, en termes de gestes effectués.

Jordan distingue les variables «proches» sur lesquelles agit directement le contrôleur, et les variables «distantes» sur lesquelles le contrôleur agit au travers du système commandé. Le problème vient du fait que la mesure d'erreur n'existe que sur des variables distantes.

Dans le cas d'un robot suivant une trajectoire pré-calculée, on peut mesurer l'erreur entre la position désirée et la position réelle mais il est difficile d'en déduire une erreur sur la commande ayant conduit à cette position.

Ce problème peut également posséder une composante temporelle, dans le cas où l'effet des actions de commande n'est pas ponctuel et instantané. Il est alors difficile de savoir quelle a été l'influence exacte des actions passées sur la sortie actuelle du processus.

Cette difficulté peut être rapprochée de celle rencontrée pour la modification des poids entre les unités cachées d'un réseau neuronal lorsque l'erreur n'est mesurée que sur les unités de sortie. On ne dispose malheureusement pas de technique équivalente à la rétropropagation du gradient pour résoudre le problème.

Les architectures de commande présentées dans la section suivante se distinguent principalement par la stratégie mise en œuvre pour résoudre le problème de l'apprentissage avec maître distant.

$u(t)$	commande appliquée au procédé commandé à l'instant t
$y(t)$	sortie du procédé commandé
$\hat{u}(t)$	commande estimée par le contrôleur
$\hat{y}(t)$	sortie estimée du procédé commandé
$u^*(t)$	consigne désirée en sortie du contrôleur
$y^*(t)$	sortie désirée pour le procédé commandé
ε_u	erreur sur la commande
ε_y	erreur en sortie du procédé

TAB. 2.1 – *Conventions de notation issues de [49].*

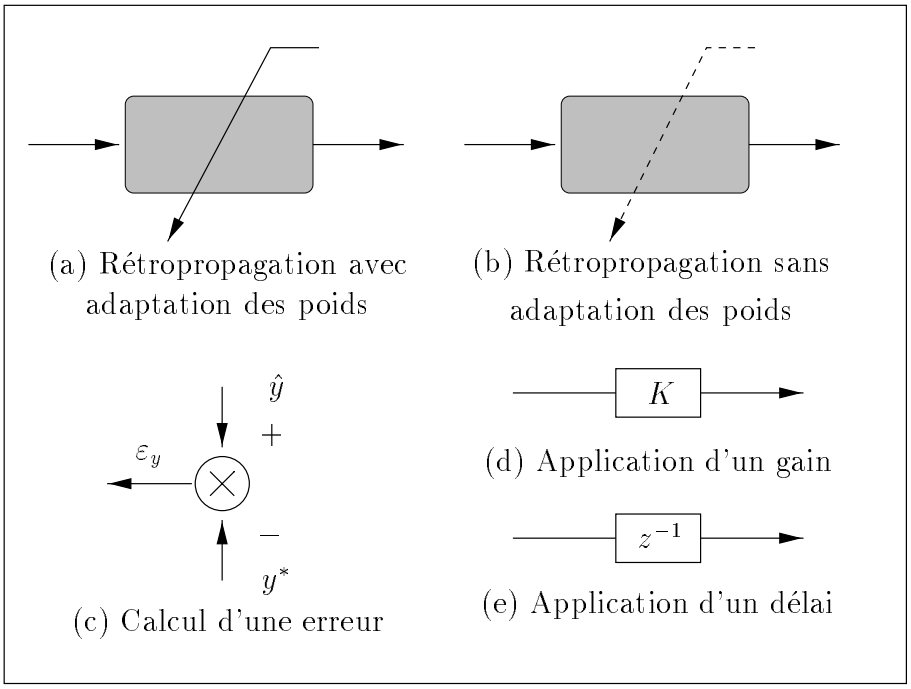

FIG. 2.17 – *Conventions graphiques. La figure (a) représente la rétropropagation d'une erreur à travers un réseau (au sens de l'algorithme de rétropropagation du gradient) avec adaptation des paramètres de ce dernier, la figure (b) représente également une rétropropagation mais sans modification des paramètres. La figure (c) montre le calcul d'une erreur à partir de la sortie constatée et de la sortie désirée. La figure (d) montre l'application d'un gain K sur une valeur et la figure (e) représente la mémorisation et le retardement d'une valeur durant un pas de temps.*

2.5 Une classification des méthodes de commande

Dans cette section nous présentons une classification des principales méthodes de commande neuronales. La plupart d'entre elles sont indépendantes du type de réseau utilisé et du type de processus commandé. Les modèles seront présentés pour des processus ne

comportant qu'une seule entrée et une seule sortie, mais peuvent très bien s'appliquer dans le cas général. On parlera dans le premier cas de machine SISO (single input-single output) et dans le deuxième de machine MIMO (multi input - multi output). Dans un souci de simplicité nous confondrons dans les modèles présentés, les états internes des processus commandés et leurs sorties observables et utilisables par les systèmes de commande. De même, afin de ne pas surcharger les schémas, nous supposerons généralement que la commande ne se fait qu'à partir de la sortie désirée pour le processus commandé, et ne nécessite pas la connaissance de la sortie précédente de ce processus (voir figure 2.18).

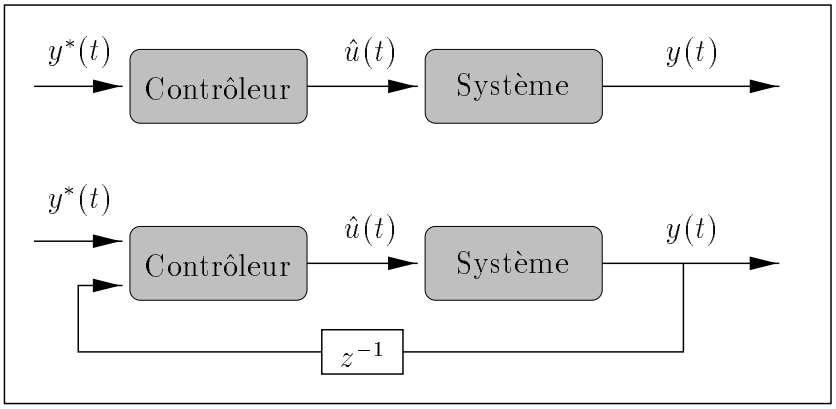

FIG. 2.18 – *Simplification des schémas. Nous présenterons les contrôleurs comme dans le cas de la figure du haut, en supposant qu'il n'est pas nécessaire de connaître l'état précedent du processus commandé pour choisir la commande.*

La grande majorité des architectures rencontrées utilise des réseaux multi-couches entraînés par rétropropagation du gradient, mais d'autres modèles sont possibles. K. Narendra [110] note en particulier que les réseaux récurrents, en raison de leur aspect dynamique, sont plus à même de modéliser des systèmes ayant eux-mêmes un caractère dynamique.
Nous séparons ces approches en deux classes principales suivant qu'elles nécessitent ou non l'identification préalable d'un modèle du processus commandé. Nous parlerons de *méthode de commande indirecte* lorsqu'un tel modèle est nécessaire, par opposition aux *méthodes directes* qui n'en nécessitent pas.

2.5.1 Méthodes directes

2.5.1.1 Reproduction d'un contrôleur existant

La première méthode utilisée pour la réalisation d'un système de commande neuronal consiste à reproduire le fonctionnement d'un contrôleur existant. Même si cette approche semble au premier abord peu intéressante puisqu'elle nécessite l'existence d'un autre contrôleur, elle peut s'avérer utile si ce dernier est trop complexe ou trop lent

pour être utilisé en temps réel, ou encore s'il utilise des données qui ne sont pas disponibles en permanence. Il est également possible de classer dans cette catégorie les cas où le système de commande reproduit est un opérateur humain.

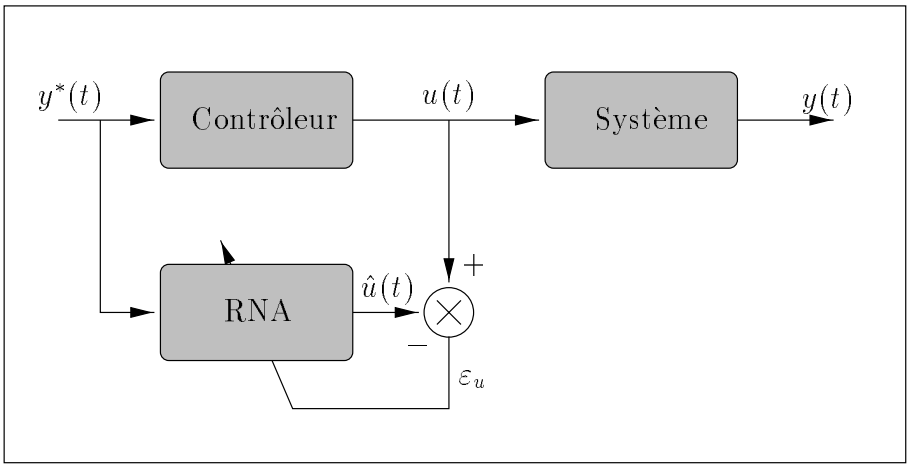

FIG. 2.19 – *Apprentissage d'un système de commande neuronal par reproduction d'un contrôleur existant.*

L'architecture générale est représentée sur la figure 2.19. Elle consiste à apprendre au réseau à reproduire la commande $\hat{u}(t)$ préconisée par le premier contrôleur à partir de la sortie désirée $y^*(t)$ et éventuellement de la sortie précédente.
Un des premiers exemples de système de commande neuronal, proposé par B. Widrow et F. W. Smith [167] en 1964, utilise cette technique pour résoudre le problème de la commande d'un pendule inversé.
D. A. Pomerleau [131] utilise également une architecture similaire pour apprendre à un véhicule à suivre une voie, en observant les lignes blanches en bordure de la route, sur une entrée vidéo. Après une courte période d'observation d'un conducteur humain le réseau est capable de conduire dans une grande variété de situations. Le système a été utilisé sur un véhicule, qu'il a réussi à conduire près de 90 % du temps lors d'une traversée Est-Ouest des États-Unis. Ce système est présenté plus en détail dans la section 4.2.3.2.
On peut noter que cette approche demande de parcourir, lors de la phase d'apprentissage, tous les modes de fonctionnement du système commandé. Il est donc nécessaire de posséder une bonne connaissance *a priori* des conditions d'utilisation du contrôleur.

2.5.1.2 Amélioration d'un système de commande linéaire

Cette approche consiste à utiliser conjointement un contrôleur linéaire classique et un contrôleur neuronal. L'idée principale est de réaliser une somme des commandes issues des deux contrôleurs, en augmentant progressivement l'importance donnée à la commande $\hat{u}(t)$ préconisée par le réseau, au fur et à mesure de l'apprentissage de ce dernier.

M. Kawato [78] propose d'utiliser un contrôleur de type «feedback» classique (CFC). Dans sa version la plus simple un tel contrôleur se contente d'appliquer un gain sur l'erreur ε_y mesurée en sortie du processus. La commande est donc proportionnelle à cette erreur. On obtient dans ce cas l'architecture décrite par la figure 2.20. Le réseau reçoit en entrée la sortie désirée $y^*(t)$ pour le processus commandé (et éventuellement sa sortie précédente) et on utilise comme signal d'erreur la sortie du CFC. Lorsque l'apprentissage avance, le réseau apprend donc à minimiser par ses commandes la sortie du CFC et par conséquent l'erreur en sortie du processus. La sortie du réseau prend par la même occasion une part de plus en plus importante dans la commande.

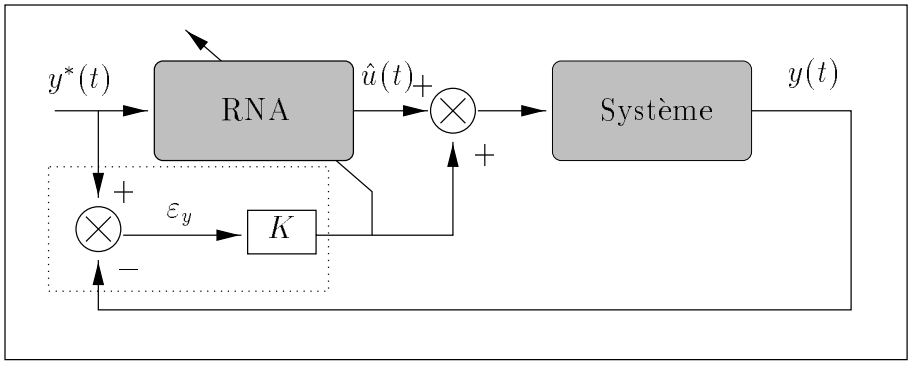

FIG. 2.20 – *Amélioration d'un contrôleur feedback classique. La zone entourée en pointillés correspond au CFC.*

On trouve parfois pour cette architecture le nom de «feedback error learning». H. Myamoto et al. [108] utilisent cette approche pour la commande d'un bras robotique. H. Ohno et al. [117] l'utilisent eux pour la commande du système de freinage sur une automobile.

2.5.1.3 Utilisation directe de l'erreur en sortie du procédé

Une approche simple consiste à tenter d'utiliser directement l'erreur ε_y mesurée en sortie du processus pour adapter le contrôleur. Plusieurs stratégies sont possibles. La première consiste à utiliser cette erreur comme s'il s'agissait de l'erreur ε_u en sortie du contrôleur. Cette approche ne peut fonctionner que si ces deux erreurs sont fortement corrélées ce qui est rarement le cas.
La méthode la plus utilisée consiste à considérer le processus comme une couche supplémentaire du réseau à travers laquelle on rétropropage l'erreur (voir figure 2.21). Pour pouvoir utiliser cette stratégie, parfois appelée apprentissage spécialisé, il est donc nécessaire de connaître le Jacobien du processus, c'est à dire la quantité $\partial y/\partial u$. Il est alors possible d'utiliser l'algorithme de rétropropagation du gradient en considérant que le processus est une extension fixe du réseau. Le calcul du gradient de l'erreur par rapport aux poids W du réseau se fait en appliquant la règle :

$$\frac{\partial \varepsilon_y}{\partial W} = \frac{\partial \varepsilon_y}{\partial y}\frac{\partial y}{\partial u}\frac{\partial u}{\partial W} \tag{2.15}$$

Le terme $\frac{\partial \varepsilon_y}{\partial y}$ dépend de la fonction d'erreur choisie, $\frac{\partial y}{\partial u}$ est le Jacobien et $\frac{\partial u}{\partial W}$ est obtenu par l'algorithme de rétropropagation du gradient.

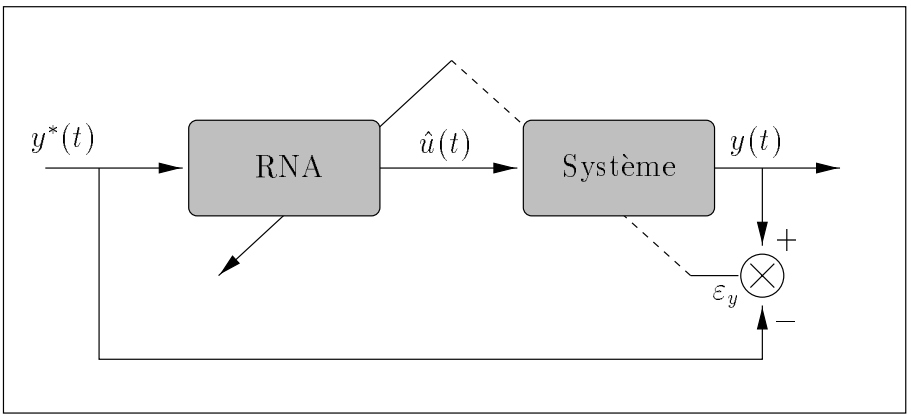

FIG. 2.21 – *Apprentissage spécialisé.*

Dans le cas où la valeur du Jacobien n'est pas connue, D. Psaltis et al. [134] proposent de l'identifier, en perturbant légèrement les entrées du processus, d'une valeur δu et en mesurant les perturbations δy obtenues en sortie.

Il est également possible de n'utiliser que le signe du Jacobien qui est souvent plus facile à connaître. La mesure du gradient de l'erreur par rapport aux poids est alors approximative mais son sens reste correct.
Y. Zhang et al. [173] utilisent cette dernière technique pour la commande en direction d'un navire.
Nous pouvons également classer dans cette section les travaux de K. P. Venugopal qui utilise un algorithme d'apprentissage particulier dont le principe est de modifier les poids du réseau en se basant sur l'observation de la corrélation entre la dernière modification effectuée et la variation de l'erreur (cette approche ne doit pas être confondue avec l'apprentissage par renforcement présenté dans la section 2.5.3). Cet algorithme appelé Alopex ne nécessite pas la mesure des dérivées de l'erreur par rapport aux poids du réseau et permet donc d'utiliser directement l'erreur mesurée en sortie du processus. Il est appliqué notamment à la commande d'un robot sous-marin [162].

2.5.1.4 Identification directe du modèle inverse

Cette approche nécessite deux phases séparées pour l'apprentissage puis pour l'utilisation du réseau. Durant l'apprentissage, le réseau et le processus sont placés en parallèle. Un échantillon de commandes u est fourni au processus. On utilise alors les sorties y de ce dernier comme entrées du réseau qui est entraîné à retrouver en sortie les commandes u. Le réseau apprend ainsi un modèle inverse du processus, c'est-à-dire une fonction donnant la commande appliquée $u(t)$ à partir de la sortie actuelle du processus $y(t)$ et éventuellement de sa sortie passée $y(t-1)$.

Après cette phase d'apprentissage le réseau est donc théoriquement capable de fournir la commande $u(t)$ nécessaire pour obtenir une sortie $y^*(t)$ qui lui est donnée en entrée. Il est donc placé directement en série avec le système commandé.
La figure 2.22 résume ces deux phases d'utilisation.

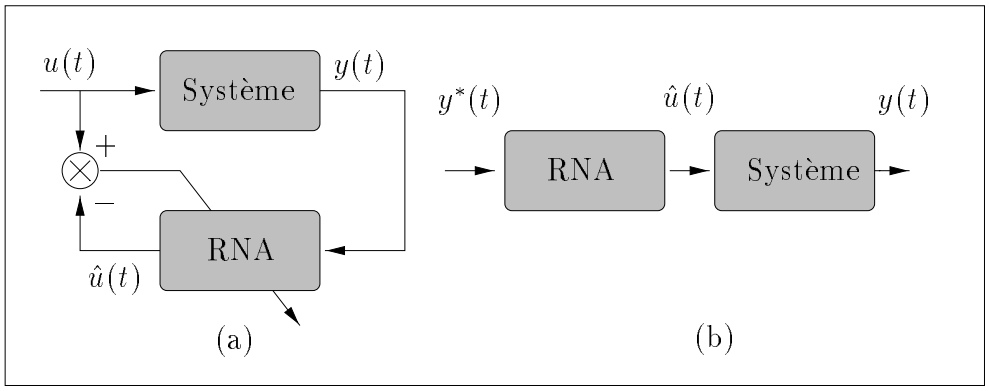

FIG. 2.22 – *Identification du modèle inverse : (a) phase d'apprentissage (b) phase d'utilisation.*

Lors de la phase d'apprentissage, il est nécessaire de faire parcourir au processus l'ensemble de ses états possibles, ou tout au moins l'ensemble des états qui seront utilisés durant la phase de commande. Si dans de nombreux cas un simple échantillonnage de l'espace des commandes suffit, il peut parfois être nécessaire d'utiliser un autre système de commande pour guider le processus durant cette phase.
La limitation majeure de cette approche vient du fait que le processus doit évidement être inversible pour que le système de commande fonctionne. Un problème se pose notamment lorsque plusieurs commandes différentes permettent d'amener le processus dans le même état. Le modèle inverse est alors mal défini puisqu'il associe plusieurs valeurs de consignes différentes à une sortie désirée donnée. Les algorithmes d'apprentissage supervisés réagissent généralement à ce problème en réalisant une moyenne des différentes valeurs possibles. Cette moyenne n'est pas forcement elle-même une consigne valide.
Cette approche a cependant été utilisée avec succès par de nombreux auteurs. On peut citer par exemple les travaux de Y. Kuroe et al. [84] qui l'utilisent pour la commande d'un bras robotique.
D. Psaltis et al. [133] proposent une variante de cette architecture, appelée «Indirect Learning Architecture[1]», décrite par la figure 2.23. Dans cette architecture les deux réseaux partagent les mêmes poids (i.e. le deuxième réseau est une copie du premier). La valeur de sortie désirée y^* est propagée à travers le premier réseau pour produire une commande $\hat{u}_1$ qui est appliquée au système. La valeur de sortie y mesurée est alors utilisée comme entrée du deuxième réseau pour obtenir une deuxième consigne

1. Le terme indirect n'a ici aucun rapport avec la notion de méthode de commande directe ou indirecte.

$\hat{u}_2$. C'est la différence entre $\hat{u}_1$ et $\hat{u}_2$ qui est utilisée pour corriger les réseaux. L'idée exploitée par cette architecture est que si l'erreur tend vers zéro, la différence $y - y^*$ tend également vers zéro. On constate cependant que le système converge souvent vers une solution triviale donnant toujours la même commande, ce qui rend l'architecture peut utilisable.

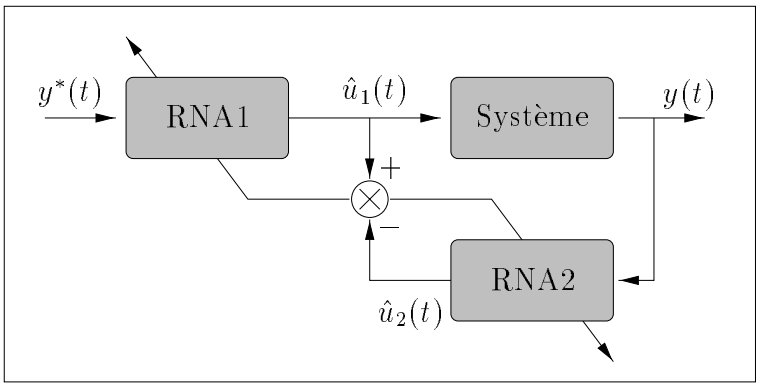

FIG. 2.23 – *Indirect Learning Architecture.*

2.5.2 Méthodes indirectes

2.5.2.1 Identification de processus

Nous entamons cette classification par une présentation des méthodes d'identification de processus utilisant des RNA. Il ne s'agit pas à proprement parler de méthodes de commande, mais plutôt d'une première phase nécessaire dans les approches indirectes. Par identification d'un processus nous entendons l'entraînement d'un réseau, à reproduire une fonction donnant les sorties ou l'état du processus à partir des entrées qui lui sont appliquées .

Le principe général de l'identification est simple et consiste à placer en parallèle le réseau et le processus à identifier, comme indiqué sur la figure 2.24. Le réseau reçoit en entrée la commande $u(t)$ appliquée et eventuellement la sortie $y(t-1)$ précédente du processus. Il est entraîné à produire la nouvelle sortie (ou le nouvel état) $y(t)$ du processus.

Cette méthode d'identification est souvent appelée méthode série-parallèle, par opposition à la méthode parallèle présentée figure 2.25. Dans cette dernière méthode le réseau ne reçoit pas en entrée la sortie réelle $y(t-1)$ du processus mais la sortie $\hat{y}(t-1)$ qu'il a lui-même prédite au pas de temps précédent. L'approche parallèle prend tout son intérêt si la boucle donnant l'état du système est remplacée par une connexion récurrente du réseau. Ce dernier est alors libre de développer sa propre représentation de l'espace des états du processus.

L'approche série-parallèle est cependant souvent considérée comme plus stable [138] car le réseau est régulièrement «recalé» en utilisant l'état réel du processus.

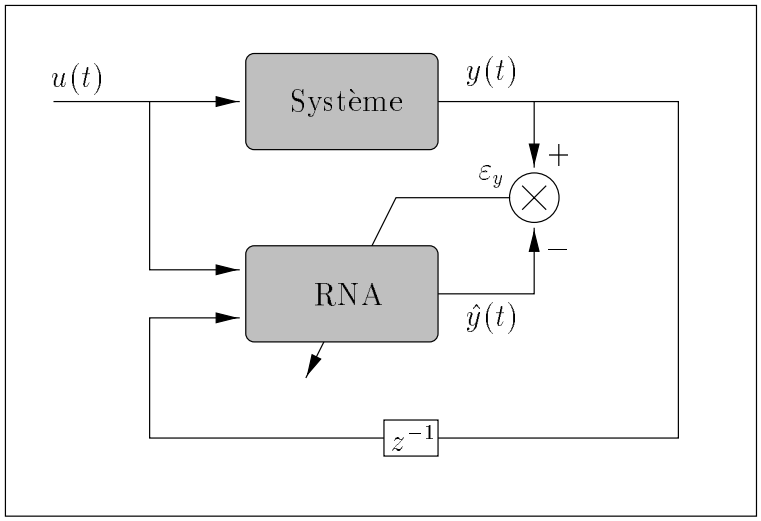

FIG. 2.24 – *Identification de processus par la méthode série-parallèle.*

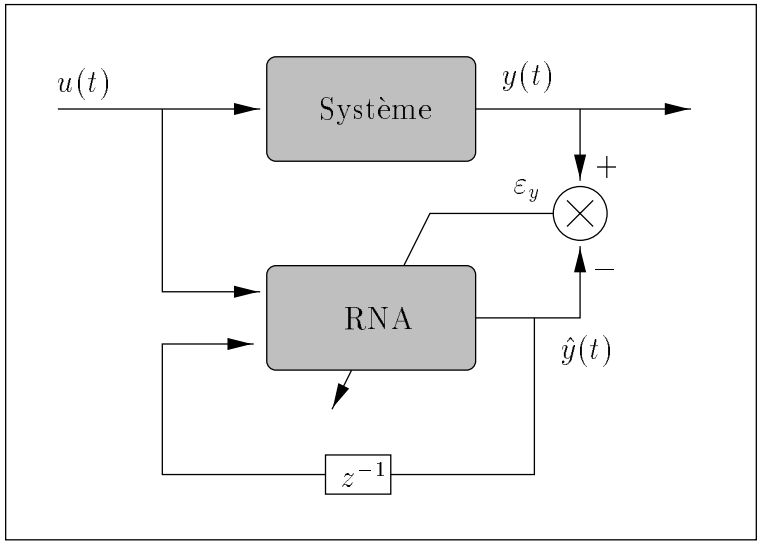

FIG. 2.25 – *Identification de processus par la méthode parallèle.*

L'utilisation en-ligne de ces stratégies d'identification peut poser un certain nombre de problèmes liés à la disponibilité graduelle des données. On ne maîtrise en effet ni l'ordre dans lequel les exemples sont présentés au réseau ni leur distribution, qui sont liés à la dynamique du processus et au contrôleur utilisé pour piloter ce dernier durant l'apprentissage.

Une méthodologie plus complète pour l'identification de processus dynamiques peut être trouvée dans l'article de K. S. Narendra et K. Parthasarathy [110].

2.5.2.2 Apprentissage indirect du système de commande

Cette méthode décrite par la figure 2.26 utilise un modèle du processus commandé réalisé par un premier RNA. Ce modèle peut être obtenu comme indiqué dans la section précédente. On cherche ainsi à obtenir par un réseau une bonne approximation $\hat{y}(t)$ de la sortie du processus lorsqu'une commande $u(t)$ lui est appliquée.
Le principe de l'architecture est alors, puisque l'on dispose d'un modèle différentiable du processus, d'utiliser l'algorithme de rétropropagation du gradient en considérant que le système de commande et le modèle ne forment qu'un seul réseau. L'erreur est propagée à travers le modèle sans modifier ses poids, et seul le réseau donnant la commande subit un apprentissage. On peut alors se ramener au cas de l'équation 2.15 en faisant l'approximation :

$$\frac{\partial y}{\partial u} = \frac{\partial \hat{y}}{\partial u}$$

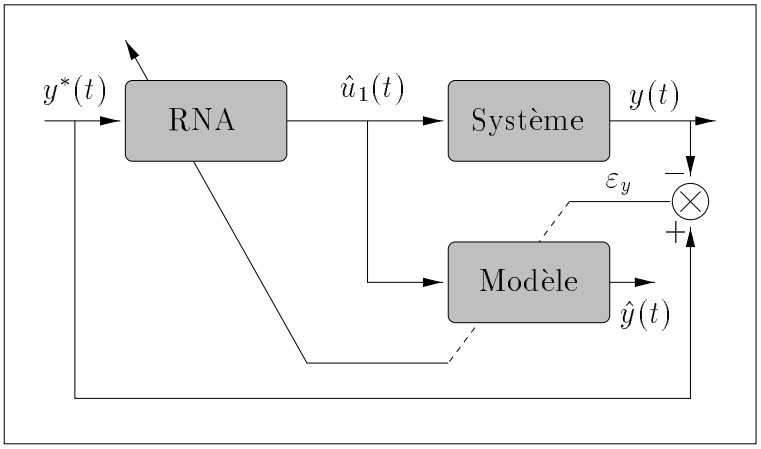

FIG. 2.26 – *Utilisation d'un modèle différentiable du processus.*

Dans le cas probable où le modèle utilisé est imparfait, cette imperfection introduit un biais qui perturbe l'apprentissage. Il est cependant tout de même possible de converger (après un temps plus long) vers un système de commande satisfaisant puisque l'erreur minimisée ne fait pas intervenir la sortie estimée $\hat{y}$ et ne dépend donc pas de la qualité du modèle. Ce dernier n'intervient que dans le calcul de la quantité $\partial y/\partial u$.
M. Jordan [74] montre que dans le cas où plusieurs commandes sont possibles pour amener le processus dans l'état désiré (i.e. l'inverse du processus est mal défini), cette architecture conduit à sélectionner une solution particulière. Le système de commande n'apprend pas une moyenne des différentes solutions possibles.
Cette architecture est notamment utilisée par K. Ishii et al. [67] pour la commande d'un robot sous-marin, ou par Q. H. Wu et al. [168] pour la commande d'un turbogénérateur.

N. Pican et F. Alexandre [128] utilisent cette approche avec un modèle particulier de réseau baptisé OWE (pour Orthogonal Weight Estimator). Dans ce type de réseau, que

les auteurs utilisent pour créer aussi bien le contrôleur que le modèle direct du système commandé, plusieurs jeux de poids sont utilisés en fonction du contexte. Ces ensembles de poids sont fixés par un deuxième réseau qui prend pour entrée les informations permettant de déterminer ce contexte. Cette approche permet de réduire la complexité du problème traité par le réseau principal en le découpant en plusieurs sous-problèmes. L'application présentée est la commande d'un bras robotique, de longueur variable et capable de porter une masse. Le réseau choisissant les poids reçoit en entrée la longueur du bras et la masse transportée.

D. Nguyen et B. Widrow proposent une variante de cette méthode [113] dans le cas où la mesure d'erreur en sortie du processus n'est pas disponible à chaque pas de temps, mais uniquement après un certain nombre d'exécutions du système. Le principe utilisé consiste à rétropropager l'erreur à travers plusieurs copies de l'ensemble contrôleur + modèle correspondant aux différents pas de temps pendant lesquels le système à été utilisé. Le fonctionnement de l'apprentissage est décrit par la figure 2.27. On peut noter que le principe mis en œuvre dans cette architecture présente des similitudes avec l'algorithme de rétropropagation dans le temps [164] utilisé pour l'apprentissage de réseaux récurrents. L'approche est appliquée par Nguyen et Widrow au problème bien connu du parking parallèle d'un camion tractant une remorque. L'erreur n'est mesurable dans cette application qu'une fois que le véhicule est garé, ou qu'il a raté définitivement son approche.

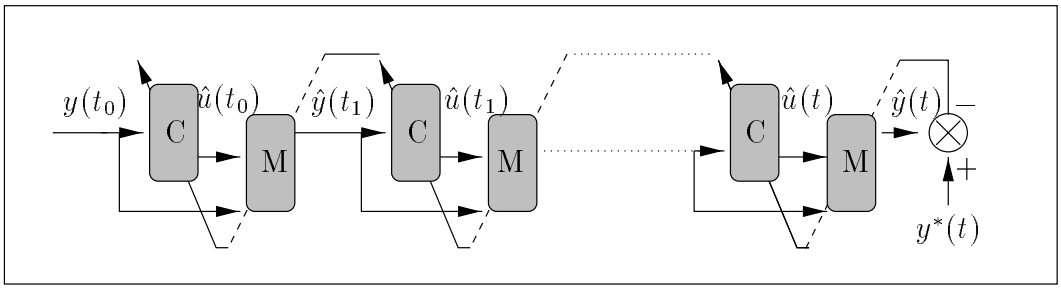

FIG. 2.27 – *Architecture proposée par Nguyen et Widrow. Les blocs marqués C correspondent au système de commande et les blocs marqués M correspondent au modèle du processus.*

R. Biewald [17] propose une architecture similaire pour la commande réactive d'un robot mobile. Nous reviendrons sur ce travail dans la section 4.2.3.3.

2.5.2.3 Réalisation d'un système de commande par modèle prédictif

Les architectures de commande présentées ci-dessus ont toutes pour principe de tenter d'obtenir par apprentissage un modèle neuronal inverse du fonctionnement du processus. Que ce soit par une méthode directe ou par une méthode indirecte, le but est en effet toujours de construire un système donnant la commande à partir de l'état

(ou de la sortie) actuel et de l'état (ou de la sortie) désiré. Le principe mis en oeuvre par la commande par modèle prédictif est totalement différent. Les réseaux sont ici utilisés pour construire un modèle direct du processus, comme décrit dans la section 2.5.2.1, afin de prédire les états futurs de ce dernier. Le système de commande consiste alors à utiliser un algorithme d'optimisation numérique non linéaire afin de minimiser une fonction de coût dépendant des états futurs prédits. Cette fonction de coût est généralement de la forme suivante :

$$J(N_1, N_2, N_u, t) = \sum_{i=t+N_1}^{t+N_2} \mu(i)[y^*(i) - \hat{y}(i)]^2 + \sum_{i=t}^{t+N_u} \lambda(i)[u(i) - u(i-1)]^2, \qquad (2.16)$$

où N_1 et N_2 définissent l'horizon de prédiction, N_u est l'horizon de contrôle et $\lambda(i)$ et $\mu(i)$ sont des constantes permettant de régler l'influence relative des différents termes. La première partie de cette équation est fonction de l'erreur entre la sortie désirée et la sortie prédite pour le processus. La deuxième partie est fonction de l'incrément de commande $u(i) - u(i-1)$ et permet d'obtenir une commande régulière et sans à-coups. La figure 2.28 présente l'architecture obtenue avec un modèle neuronal du processus.

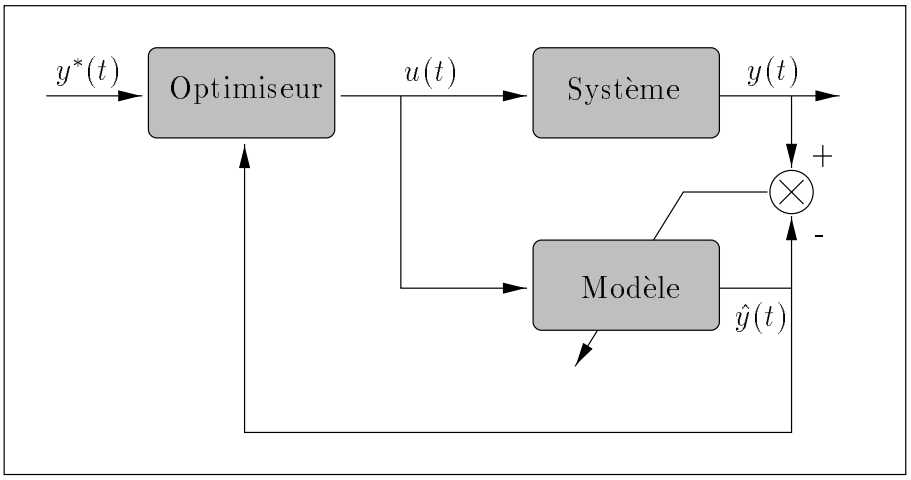

FIG. 2.28 – *Commande par modèle prédictif.*

P. M. Mills et al. [103] présentent une synthèse assez complète des possibilités d'utilisation d'un modèle neuronal dans le cadre de la commande par modèle prédictif (MBPC). J. B. Gomm et al. [47] utilisent un système de ce type pour la commande du niveau d'eau dans un système complexe de plusieurs réservoirs.
L'approche proposée par J. G. Ortega et E. F. Camacho [120] est assez différente puisqu'ils proposent d'utiliser un RNA non pas pour modéliser le système, mais pour apprendre le résultat de l'algorithme d'optimisation, ce dernier étant trop complexe pour être utilisé en-ligne. Cette architecture, appliquée à la commande d'un robot mobile sera détaillée dans la section 3.2.3.
H. Fritz [40] utilise une méthode similaire, également pour la commande de véhicules

autonomes. L'objectif recherché est le maintient de la distance de sécurité dans une file de voitures.

2.5.3 Utilisation d'un apprentissage par renforcement

Nous présentons cette approche dans une section spéciale car elle se distingue fortement des précédentes dans le principe mis en œuvre. Nous aurions pu toutefois la classer dans les méthodes directes.
Les architectures que nous avons présentées sont toutes basées sur l'utilisation d'un apprentissage supervisé. Elles requièrent donc une information quantitative sur l'erreur commise par le système de commande. Si nous avons vu que cette information est rarement disponible directement, il est souvent beaucoup plus facile de disposer d'une appréciation sur la réalisation ou non des objectifs de commande. Si nous prenons le cas d'un robot mobile cherchant à éviter des obstacles il est difficile de connaître à tout moment la différence entre la commande choisie et la commande idéale, mais il est possible de dire si les commandes choisies ont permis ou non de remplir l'objectif (i.e. si aucun obstacle n'a été percuté). Plusieurs algorithmes ont été proposés pour permettre l'apprentissage des RNA à partir d'une information uniquement qualitative sur le succès ou l'échec de la commande. Le principe est de «récompenser» le réseau si l'action choisie a conduit à une amélioration de l'état du processus et de le «punir» dans le cas contraire. La récompense (resp. la punition) consiste à renforcer (resp. à diminuer) la tendance du réseau à choisir, dans l'état courant, la commande qu'il vient de sélectionner.
La seule information qui doit être fournie par l'utilisateur durant la phase d'apprentissage est la fonction donnant le signal (appelé signal de renforcement) indiquant l'effet positif ou négatif de l'action choisie. On parle pour ce type d'algorithmes, *d'apprentissage par renforcement.*
De manière générale l'apprentissage par renforcement se distingue des méthodes d'apprentissage supervisé par les caractéristiques suivantes [37] :

- la stratégie de commande est adaptée sur la base du signal de renforcement qui indique uniquement si l'action choisie a été bonne ou mauvaise. Ce signal n'indique pas quelle action aurait été plus appropriée ;

- puisque les commandes ne sont pas proposées par un superviseur, le système doit lui-même utiliser une stratégie d'exploration des différentes actions possibles. Cette stratégie doit présenter un compromis entre complétude de l'exploration et rapidité d'apprentissage ;

- la valeur du signal de renforcement peut représenter les conséquences de commandes choisies plusieurs pas de temps auparavant. Le système doit pouvoir utiliser le renforcement obtenu sur un intervalle de temps important.

Nous décrivons ci-dessous les deux principales familles d'approches rencontrées dans la littérature pour la réalisation de systèmes d'apprentissage par renforcement.

2.5.3.1 Le critique heuristique adaptatif

Dans de nombreux cas, l'utilisation du renforcement immédiat $r(t)$ peut ne pas être suffisante pour estimer le bien fondé de l'action entreprise.
Dans le cadre de la robotique mobile par exemple, même si un mouvement ne provoque pas une collision immédiate, il peut placer le robot dans une position telle que la collision est très probable dans un futur proche.
On cherche donc généralement à maximiser les performances du système non pas sur un pas de temps mais sur un intervalle plus long. Il est donc nécessaire de construire un critique capable d'évaluer ces performances. Ce critique est réalisé par un premier RNA dans cette approche.
Supposons que le renforcement immédiat du système à l'instant t, ne dépende que de l'état courant $x(t)$ et de la commande sélectionnée $u(t)$, et soit noté $r(x(t), u(t))$. On peut alors définir la mesure de performance du système comme la somme des renforcements futurs :

$$J(x(t), u(t)) = \sum_{i=t}^{\infty} \gamma^{i-t} r(x(i), u(i)), \tag{2.17}$$

où γ est une constante inférieure à 1. Le terme γ^{i-t} permet donc de diminuer l'influence des renforcements les plus éloignés dans le temps.
La valeur exacte de J ne peut être connue qu'après l'exécution de l'ensemble des commandes. On cherche donc à en réaliser une prédiction $\hat{J}$ à l'aide du réseau critique. La relation entre deux valeurs successives de J est donnée par l'expression suivante :

$$J(x(t), u(t)) = r(x(t), u(t)) + \gamma J(x(t+1), u(t+1))$$

Lorsque le réseau réalise une bonne approximation, on doit donc retrouver cette relation entre ses prédictions :

$$\hat{J}(x(t), u(t)) = r(x(t), u(t)) + \gamma \hat{J}(x(t+1), u(t+1))$$

Dans le cas contraire on peut utiliser la valeur $\delta(k)$ suivante, comme signal d'erreur pour l'apprentissage du réseau :

$$\delta(k) = r(x(t), u(t)) + \gamma \hat{J}(x(t+1), u(t+1)) - \hat{J}(x(t), u(t)) \tag{2.18}$$

Cette approche porte le nom de *méthode des différences temporelles* [155].
La valeur J est généralement appelée renforcement interne par opposition au renforcement externe r.
Le système de commande est lui aussi réalisé par un RNA. Son but est de produire à tout moment une action maximisant J ou son approximation $\hat{J}$.
Plusieurs stratégies sont possibles pour son apprentissage qui se déroule en parallèle de celui du critique comme décrit par la figure 2.29 :

- puisque le critique est réalisé par un réseau différentiable, le gradient du renforcement interne par rapport à la commande peut être calculé. Les poids W du

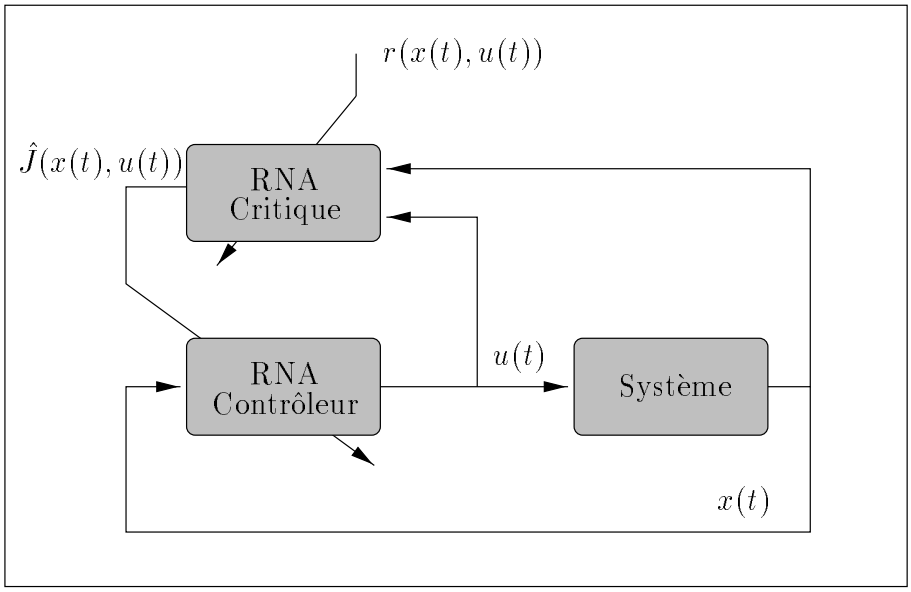

FIG. 2.29 – *Apprentissage par renforcement. La méthode du critique heuristique adaptatif.*

réseau sont alors modifiés en suivant la règle :

$$\Delta W(t) = \eta \frac{\partial J(x(t), u(t))}{\partial u(t)} \frac{\partial u(t)}{\partial W(t)}$$

P. J. Werbos [165] propose une revue de ces méthodes ;

– une approche plus simple consiste à utiliser directement la mesure de performance $\hat{J}$. Lorsque cette valeur est positive ou lorsque cette valeur augmente, l'action choisie a eu un effet positif et l'on doit renforcer la tendance du réseau à la choisir dans l'état courant. L'adaptation opposée doit être réalisée dans le cas contraire. C'est ce principe qui est à la base de l'approche pionnière de A. G. Barto et al. [12].

2.5.3.2 Le Q-Learning

La méthode du Q-Learning [163] n'utilise qu'une seule structure pour coder à la fois la fonction d'évaluation de l'état courant et la loi de commande. Si l'on suppose que le nombre d'états du processus commandé et le nombre d'actions possibles sont finis, il est possible de stocker les valeurs d'évaluation des performances associées à une action et un état dans un tableau construit au fur et à mesure. Cette mesure de performance notée $Q(x(t), u(t))$ est similaire à celle réalisé par le critique dans la méthode précédente. Le principe de l'implantation neuronale du Q-Learning est d'approcher cette mesure par un réseau entraîné par la méthode des différences temporelles.

Les actions de commandes sont alors sélectionnées en choisissant celle pour laquelle le réseau donne la plus forte valeur d'évaluation Q. De manière à permettre au système de commande d'explorer l'espace des actions possibles, une perturbation aléatoire dont l'intensité diminue avec le temps est introduite dans le choix de la commande. Cette perturbation doit en effet diminuer au fur et à mesure que la qualité de l'estimation de la fonction $Q(x(t), u(t))$ par le réseau augmente.
Cette méthode a été particulièrement étudiée dans le cadre de la robotique mobile pour l'apprentissage de comportements simples comme l'évitement d'obstacles ou le suivi de contour [71, 160].
On peut noter que le Q-Learning est un cas particulier de la méthode précédente où le critique est aussi utilisé comme contrôleur.

De nombreuses applications de ces approches existent dans la littérature. On peut citer les travaux de V. Gullapalli et al. [51] qui utilisent un apprentissage par renforcement pour apprendre à un bras robotique à insérer des objets dans un orifice. D. A. White et D. A. Sofge [166] proposent une architecture générale basée sur l'utilisation d'un critique adaptatif pour la commande de processus manufacturiers.
Les approches basées sur l'apprentissage par renforcement sont particulièrement séduisantes puisqu'elles peuvent être appliquées à une très large classe de problèmes. Le principe de l'apprentissage par essai/erreur ainsi réalisé est lui aussi très intéressant, notamment dans le cadre de la robotique autonome, de par sa plausibilité biologique.
La réalisation pratique pose cependant plusieurs problèmes. Le principal est la lenteur de la convergence de l'apprentissage. Ce phénomène s'explique par la pauvreté des informations exploitées, mais aussi par le conflit entre exploration de l'espace des commandes et sélection de la commande optimale.
Cette exploration des commandes, en partie aléatoire, peut également être problématique dans le cas où le processus commandé possède des états critiques qui ne doivent pas être atteints.
Ces différents aspects seront développés dans le cadre des applications à la robotique mobile dans la section 4.2.3.1.

2.6 Conclusion

Au cours de ce chapitre nous avons présenté une synthèse des différentes approches de commande utilisant des réseaux de neurones artificiels. Le choix de l'un ou l'autre de ces schémas d'utilisation sera discuté pour chaque problème abordé dans la deuxième partie de ce mémoire.
Si la littérature regorge d'exemples d'emploi des RNA sur des problèmes jouets ou des problèmes tests bien connus, les réalisations pratiques sont beaucoup plus rares. Une grande partie des travaux présentés se limitent en effet à l'étude de problèmes déjà résolus par des méthodes classiques, auxquels ils n'apportent qu'une contribution limitée.

Il nous semble donc essentiel avant d'envisager l'utilisation de RNA pour la résolution d'un problème de commande, de nous poser la question de l'apport possible par rapport aux méthodes existantes. J-M. Renders [138] avance même qu'il est sans doute nécessaire d'étendre les objectifs actuels des systèmes de commande issus de l'automatique pour tirer pleinement partie des méthodes neuronales.
Dans la suite de ce mémoire nous nous efforcerons donc de comparer les propriétés des approches utilisées avec celles des autres méthodes proposées dans la littérature.

Conclusion de la première partie

Dans le premier chapitre de ce mémoire nous avons passé en revue les principales architectures de commande pour robots mobiles, rencontrées dans la littérature. Ceci nous a permis de définir trois niveaux fonctionnels nécessaires pour conférer une véritable autonomie au véhicule. Le niveau supérieur (i.e. planification de missions) fait intervenir des traitements symboliques pour lesquels les techniques classiques de l'IA sont plus adaptées, à notre avis, que les réseaux de neurones artificiels. Ces derniers peuvent par contre par leur rapidité de traitement, leurs capacités de généralisation, d'adaptation, mais aussi par la possibilité qu'ils offrent de spécifier précisément la tâche à accomplir par apprentissage, avoir un rôle à jouer dans les deux autres niveaux qui demandent une forte réactivité. C'est aussi le point de vue de C. Torras [159] qui soutient notamment que chez l'homme : *la planification et le contrôle des mouvements impliquent deux types différents de traitement. D'une part la planification globale implique un traitement symbolique séquentiel, et d'autre part les méthodes visant à éviter localement des obstacles impliquent un traitement distribué de manière fortement parallèle (i.e. du type de celui réalisé par les RNA).*
Dans la suite de ce travail nous nous sommes donc tout naturellement tournés vers l'utilisation des techniques connexionnistes (i.e. utilisant les réseaux de neurones artificiels) pour la réalisation de systèmes correspondant aux deux premiers niveaux décrits dans la section 1.4 (i.e. systèmes de commande réactifs et contrôle d'exécution de missions). La deuxième partie de ce mémoire présente un exemple de réalisation d'une manœuvre complexe mettant en œuvre plusieurs systèmes de commande basés sur les RNA.

Deuxième partie

Modèles neuronaux pour la commande d'un véhicule mobile

Introduction

Dans cette partie, nous exposons les approches explorées durant cette thèse pour la commande du véhicule.
Ces travaux sont présentés à travers l'exemple d'une manœuvre complexe qui consiste pour le robot, à suivre une trajectoire de référence tout en dépassant d'éventuels autres véhicules, se trouvant sur la même voie de circulation et roulant moins vite que lui.
La réalisation de n'importe quelle manœuvre de ce type implique plusieurs formes de traitement distinctes. Ceci nous a amené à étudier plusieurs systèmes de commande différents, permettant d'augmenter progressivement la complexité des tâches réalisées par le robot.

Le premier d'entre eux est présenté dans le chapitre 3 où nous nous intéressons au simple suivi d'un chemin de référence issu d'un planificateur, sans tenir compte des données provenant des capteurs extéroceptifs. Ce type d'objectifs constitue en quelque sorte le degré minimum d'autonomie du robot, et on parle généralement de *commande par retour d'état* pour le décrire. Le but y est en effet uniquement de fixer les valeurs de certaines variables d'état du véhicule (la position dans notre cas) à partir d'une suite de consignes pré-calculées.
L'objectif de l'approche proposée est de tirer parti des possibilités offertes par les RNA de modéliser précisement, par apprentissage, les réponses du robot aux commandes qui lui sont appliquées. Nous cherchons plus particulièrement à réaliser un système capable de s'adapter à des conditions d'utilisation changeantes.

Afin de pouvoir réagir à la présence d'autres véhicules ou d'obstacles divers, le robot doit également être capable de choisir ses mouvements, non pas à partir de consignes extérieures mais en se basant sur sa perception de l'environnement. Dans le chapitre 4 nous nous intéressons donc aux possibilités d'approcher par apprentissage une fonction associant les commandes aux données sensorielles du véhicule. On parle généralement de *commande référencée capteur* pour désigner ce type de traitement, qui permet d'effectuer des tâches plus complexes et demandant plus d'autonomie de la part du robot que celle évoquée dans le chapitre 3. Nous présentons de manière plus générale une méthodologie pour la réalisation de manœuvres simples, basées sur les capteurs du robot, utilisant un modèle de réseau neuro-flou particulier.

Les deux types de contrôleurs présentés dans les chapitres 3 et 4 correspondent au premier niveau d'une architecture de commande, défini dans la section 1.4.
Nous reviendrons dans ces deux cas sur les possibilités d'application des différents schémas d'utilisation des RNA que nous avons évoqués dans le chapitre 2. Nous nous efforcerons également, de comparer les approches proposées avec celles rencontrées dans la littérature et ayant pour but la résolution de problèmes similaires.

La réalisation d'une manœuvre complexe implique enfin, comme nous l'avons vu dans la section 1.4, de pouvoir enchaîner plusieurs systèmes de commande différents en fonction d'un plan pré-établi mais aussi en fonction des événements extérieurs. Nous abordons donc dans le chapitre 5 l'utilisation des RNA pour la réalisation de ce niveau d'une architecture de commande.

Chapitre 3

Réseaux de neurones artificiels et suivi de chemin

La première partie de notre travail a porté sur l'application des réseaux de neurones artificiels à la résolution d'une tâche simple ne demandant que peu d'autonomie de la part du robot. Nous proposons ici un système de commande ayant pour but le suivi d'un chemin de référence planifié à l'avance. Ce chapitre débute par un exposé des problèmes particuliers posés par cette tâche, et par le type de véhicule expérimental dont nous disposons. Nous proposons également une revue des principales approches rencontrées dans la littérature, pour la résolution de ce problème, afin de pouvoir établir une comparaison avec notre méthode. Celle-ci est validée, dans un premier temps, sur simulateur.

3.1 Problématique et objectifs

Lorsque l'environnement dans lequel évolue un véhicule autonome est relativement bien connu, il est possible de planifier hors-ligne le chemin que celui-ci doit suivre pour atteindre son but. Cet itinéraire peut se résumer à une série de points du plan, que le véhicule doit parcourir. On peut également lui associer un profil de vitesse. On parle dans ce dernier cas de trajectoire plutôt que de chemin.
Plusieurs lois de commande analytiques ont été proposées pour permettre au robot de suivre un tel itinéraire (voir §3.2). De manière générale, ces approches obtiennent de très bonnes performances lorsque tous les paramètres du véhicule sont parfaitement connus. Ces résultats se dégradent cependant assez rapidement en cas de perturbations ou de changements de certains paramètres du robot ou de son environnement (nous présenterons des exemples de ce comportement plus loin dans ce chapitre). De telles perturbations sont malheureusement relativement fréquentes sur un véhicule de la complexité de celui que nous utilisons (véhicule de type voiture). Leurs causes peuvent être par exemple :

- une différence dans le diamètre des roues, qui peut provenir tout simplement d'un

mauvais gonflage, ou de la charge du robot. Une telle différence, entraîne si la vitesse angulaire reste la même pour les deux roues, une dérive du robot dans une direction constante ;

- un mauvais alignement des roues qui provoque le même type de comportement ;
- un frottement plus important au niveau de l'une des roues ;
- un mauvais étalonnage ou un dysfonctionnement des capteurs internes fournissant l'angle de braquage des roues ou la vitesse du robot.

En plus de ces sources d'erreurs le véhicule peut également être sujet, à une vitesse importante, à des glissements qui sont difficiles à modéliser.
Si l'ensemble de ces erreurs peut être corrigé de manière inconsciente par un conducteur humain, il n'en est pas de même pour un véhicule évoluant en mode automatique.
Nous avons donc cherché à réaliser un système de commande capable de s'adapter à des changements de certains paramètres du véhicule ou de son environnement. Le but de notre approche est d'utiliser les techniques des réseaux de neurones artificiels pour «apprendre» les réponses exactes du véhicule aux commandes qui lui sont appliquées. Cet apprentissage doit pouvoir être réalisé de manière continue pour permettre *une adaptation du contrôleur à des conditions d'utilisation changeantes.*

3.2 Description des approches courantes

Nous présentons dans cette section deux lois de commande très largement utilisées dans la littérature. L'une d'entre elles concerne le suivi de chemin, et l'autre le suivi de trajectoire. Ces algorithmes seront utilisés par la suite comme référence pour juger les performances de notre approche. Dans la dernière partie de cette section nous présentons quelques exemples d'utilisation de réseaux de neurones artificiels pour le suivi de chemin. Nous utilisons, dans cette section et dans le reste du mémoire, les notations et la modélisation du véhicule introduits dans l'annexe A.

3.2.1 Méthode proposée par Y. Kanayama et al.

La méthode proposée par Kanayama et al. [76] s'applique au suivi de trajectoire (i.e. un profil de vitesse doit être spécifié pour le véhicule).
Dans cette méthode la trajectoire est représentée par une suite de configurations de référence notées $q_r = (x_r, y_r, \theta_r)^T$ où x_r et y_r sont les coordonnées de la position désirée pour le centre de l'essieu arrière du véhicule dans le repère principal et θ_r est l'orientation désirée pour le véhicule dans ce même repère.
Le but de la méthode est de réduire la distance entre cette configuration et la configuration courante du véhicule $q_c = (x_c, y_c, \theta_c)^T$

Les auteurs définissent une erreur de configuration q_e :

$$q_e = \begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = \begin{bmatrix} \cos(\theta_c) & \sin(\theta_c) & 0 \\ -\sin(\theta_c) & \cos(\theta_c) & 0 \\ 0 & 0 & 1 \end{bmatrix} (q_r - q_c) \tag{3.1}$$

Cette erreur est la valeur de la configuration de référence q_r exprimée dans un repère associé au véhicule.
Les auteurs proposent une loi de commande, agissant sur la vélocité v ainsi que la vitesse angulaire $w = \dot{\theta}$ du véhicule, dont le but est de minimiser q_e. Cette loi utilise en plus de l'erreur de configuration q_e, la vélocité v_r ainsi que la vitesse angulaire w_r associées à la configuration désirée. Elle est donnée par :

$$\begin{bmatrix} v_{cons} \\ w_{cons} \end{bmatrix} = \begin{bmatrix} v(q_e, v_r, w_r) \\ w(q_e, v_r, w_r) \end{bmatrix} = \begin{bmatrix} v_r \cos(\theta_e) + K_x x_e \\ w_r + v_r(K_y y_e + K_\theta \sin(\theta_e)) \end{bmatrix}, \tag{3.2}$$

où v_{cons} et w_{cons} sont les consignes, et K_x, K_y et K_θ sont des paramètres constants dont le choix et le réglage sont laissés à l'utilisateur.
Kanayama et al. ont prouvé, au moyen d'une fonction de Liapunov, la stabilité de cette loi de commande à la condition que la vitesse de la configuration de référence ne s'annule jamais. Ils proposent de plus un réglage des paramètres garantissant un minimum d'oscillations. Ils considèrent pour cela le comportement du système dans le cas d'une trajectoire rectiligne. Celui-ci est équivalent à un second ordre en y_e (erreur suivant l'axe des y dans le repère de la position de référence) :

$$\ddot{y_e} + 2\zeta\xi\dot{y_e} + \xi^2 = 0,$$

où ζ est le coefficient d'amortissement $\frac{K_\theta}{2\sqrt{K_y}}$, et ξ est la pulsation propre du système $v_r\sqrt{K_y}$. Afin d'obtenir un compromis optimal entre l'absence d'oscillation et une convergence rapide vers $y_e = 0$ on choisit :

$${K_\theta}^2 = 4K_y$$

Ce réglage des paramètres est généralement admis même lorsque la trajectoire n'est plus rectiligne.
L'équation A.7 nous permet d'obtenir aisément une commande en angle de braquage des roues avant, plus facile à appliquer que celle en vitesse de rotation du véhicule :

$$\phi_{cons} = arctan(\frac{l_w w_{cons}}{v_r}) \tag{3.3}$$

3.2.2 Méthode proposée par C. Samson

C. Samson propose une loi de commande [144, 101] permettant de suivre un chemin de manière géométrique indépendamment de la vitesse du véhicule. La seule variable de commande utilisée est la vitesse angulaire $w = \dot{\theta}$ de la voiture.

Soit $\mathcal{C}$ le chemin suivi et $\rho(s)$ sa courbure au point d'abscisse curviligne s, on note R' le point de $\mathcal{C}$ le plus proche de R (point de référence du véhicule) et d la distance$||RR'||$ (voir figure 3.1). De même on note θ_{des} l'angle de la tangente à la courbe en R' dans le repère principal. Cet angle représente l'orientation désirée pour le véhicule.

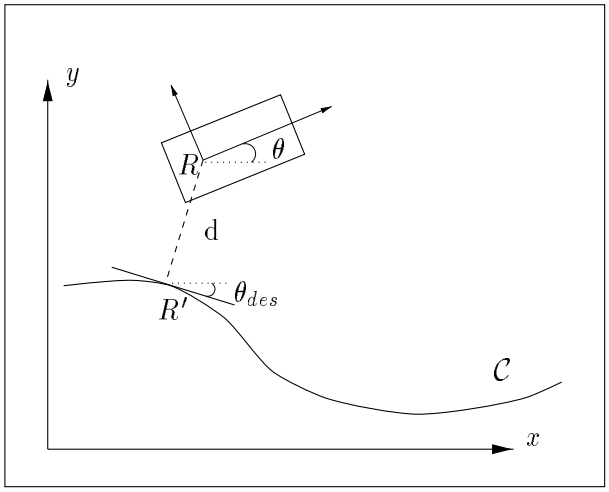

FIG. 3.1 – *Suivi de chemin par la méthode de C. Samson.*

Le but de la loi de commande proposée est de réduire à la fois l'erreur en distance d et l'erreur en orientation $\theta_e = \theta - \theta_{des}$.
Elle est donnée dans sa forme la plus simple par[1] :

$$w = v\rho(s_{R'}) - k_1 v d - k_2 |v| \theta_e, \tag{3.4}$$

où $s_{R'}$ est l'abscisse curviligne du point R' et k_1 et k_2 sont deux constantes positives. La vitesse de translation v du véhicule qui intervient comme un paramètre de la loi de commande est donc laissée libre et n'influe pas sur la convergence géométrique vers $\mathcal{C}$ (à condition de rester non nulle).
Tout comme pour la loi de commande de Kanayama, des valeurs des constantes permettant de limiter les oscillations peuvent être obtenues dans le cas d'un chemin rectiligne :

$$k_1 = \xi^2 \text{ et } k_2 = 2\zeta\xi,$$

où ξ est la pulsation propre du système et ζ son amortissement. Ces valeurs sont également admises dans le cas général.

3.2.3 Approches basées sur l'utilisation des réseaux de neurones artificiels

Les exemples de systèmes de commande basés sur l'utilisation de RNA, pour le suivi de chemin ou de trajectoire sont assez rares à notre connaissance. Nous pouvons citer

1. L'équation que nous donnons est en fait une approximation au premier ordre de la loi énoncée à l'origine par C. Samson.

quelques-uns d'entre eux.
Le but recherché dans ces travaux est l'obtention par apprentissage de la commande permettant de réduire la distance entre la configuration courante et la configuration de référence :

- P. Henaff [59] propose pour cela une approche qu'il baptise rétropropagation indirecte. Il définit un critère dépendant de l'erreur en position et en orientation. Le modèle cinématique du véhicule est utilisé pour calculer la dérivée de ce critère par rapport aux consignes. Ceci permet la réalisation d'un système de commande utilisant un RNA entraîné par l'algorithme de rétropropagation à fournir les consignes minimisant le critère d'erreur. La technique employée est ainsi très proche de celle décrite dans la section 2.5.1.3 (apprentissage spécialisé).

  Si cette approche semble fournir de bons résultats (sur simulateur), elle n'apporte à notre avis aucun avantage par rapport à une loi de commande analytique telle que celles que nous venons de présenter. Elle reste en effet basée sur l'utilisation d'un modèle simple du robot plutôt que sur son comportement réel, et l'on ne possède ici aucune preuve de stabilité ;

- I. Rivals et al. [140] utilisent la technique de l'apprentissage indirect présentée dans la section 2.5.2.2. Le système peut, dans ce cas, être entraîné directement sur le véhicule sans utilisation d'un modèle simplifié et donc prendre en compte les paramètres réels du robot. Les résultats présentés semblent cependant être moins bons que ceux obtenus par une loi de commande classique. L'apprentissage doit de plus être réalisé sur un très grand nombre de situations et rien ne peut garantir que les résultats pourront être généralisés à toutes les positions du véhicule ;

- J. G. Ortega et E. F. Camacho [120] utilisent un système de commande par modèle prédictif (voir §2.5.2.3). Cette approche permet de traiter à la fois le problème du suivi de chemin et celui de l'évitement d'obstacle. Les auteurs utilisent un modèle analytique du véhicule et de ses capteurs pour prédire l'état du système lors des prochains pas de temps. Ils utilisent hors-ligne un algorithme d'optimisation numérique non linéaire, permettant de trouver les consignes maximisant un critère tenant compte à la fois de la proximité des obstacles et de la position du véhicule par rapport au chemin de référence. Cet algorithme étant trop lourd pour être utilisé en temps réel, les auteurs proposent d'utiliser un réseau multi-couches entraîné à reproduire les consignes choisies.

  Cette approche ne présente pas à notre avis un grand intérêt si l'on ne se préoccupe que du suivi de chemin puisque encore une fois, elle se base sur un modèle cinématique simple du robot. Elle présente par contre l'avantage de combiner un comportement d'évitement d'obstacle avec celui de suivi.

Nous pouvons enfin signaler que nous avons nous-même tenté d'utiliser une technique basée sur un apprentissage par renforcement. Si des résultats prometteurs ont été obtenus sur simulateur, il nous semble très difficile, de réaliser l'apprentissage directement

sur le robot. Ceci limite très fortement l'intérêt de cette approche puisque, encore une fois, l'apprentissage se fait à partir d'un modèle simple du véhicule.

3.3 Présentation de notre approche

3.3.1 Principe général

Comme nous l'avons vu précédemment nous cherchons à utiliser les capacités des réseaux de neurones artificiels pour «apprendre» la relation entre les déplacements réellement effectués par le robot et les commandes qui lui sont appliquées. Notre objectif est de nous affranchir de l'utilisation d'un modèle analytique, forcement imparfait, du robot. Afin de pouvoir commander le véhicule, nous cherchons plus précisément à obtenir à l'aide d'un réseau une approximation du modèle inverse de son fonctionnement, c'est-à-dire de la fonction donnant les consignes appliquées à partir du déplacement provoqué par ces consignes. Un tel modèle peut être utilisé comme contrôleur pour obtenir les consignes permettant d'effectuer un déplacement désiré.
Parmi les approches présentées dans le chapitre 2, nous pouvons retenir deux méthodes permettant d'approcher ce modèle par apprentissage :

- l'apprentissage indirect (voir §2.5.2.2) qui demanderait dans un premier temps l'apprentissage d'un modèle direct du véhicule par un premier RNA. Dans un deuxième temps ce réseau devrait être placé en série avec celui chargé de l'apprentissage du modèle inverse pour permettre son entraînement ;

- l'apprentissage direct du modèle inverse. Comme nous l'avons vu dans la section 2.5.1.4, cette approche consisterait dans notre cas à réaliser un apprentissage supervisé en présentant en entrée du réseau un déplacement mesuré sur le robot, et en l'entraînant à donner en sortie les consignes ayant provoqué ce déplacement.

L'utilisation de la première de ces deux solutions rendrait très difficile une modification en ligne[1] du contrôleur, puisque cela impliquerait également la modification simultanée du modèle direct du véhicule. Nous avons donc opté pour la deuxième de ces approches.
Il nous est bien sûr impossible de réaliser un apprentissage en utilisant tous les déplacements possibles du véhicule. Nous nous limitons donc aux mouvements effectués pendant un intervalle de temps Δt fixe. Une fois l'apprentissage du modèle inverse réalisé, celui-ci peut être utilisé pour guider le véhicule le long de la trajectoire désirée. Pour cela il suffit de donner au réseau le point de la trajectoire que le véhicule doit occuper au bout du prochain intervalle de temps Δt et d'utiliser les consignes qu'il fournit en sortie.

1. Modification pendant l'utilisation du réseau comme système de commande, au fur et à mesure de la disponibilité des exemples.

Nous proposons de réaliser dans un premier temps (cf. §3.4.1) un apprentissage hors-ligne[1] à partir de données recueillies en envoyant un échantillon de consignes au véhicule. Dans un deuxième temps (cf. §3.4.4) le système de commande pourra être modifié en-ligne en fonction du comportement du véhicule.

3.3.2 Mise en œuvre

3.3.2.1 Choix des entrées/sorties du système

Comme dans le cas des lois de commande que nous avons présentées dans la section précédente, nous choisissons de commander le véhicule à partir de sa vitesse v et de son angle de braquage ϕ.

L'une des difficultés du suivi de chemin vient de la nécessité de contrôler les trois paramètres de configuration du véhicule (position dans le plan et orientation), à partir de seulement deux variables de commande. Notre contrôleur se base sur l'idée que lors du suivi d'un chemin, il est possible de ne considérer que la seule position du robot. L'orientation de celui-ci est en effet contrôlée de manière implicite lorsqu'il enchaîne ses déplacements successifs.

Les seules entrées nécessaires pour notre réseau sont donc les coordonnées du point à atteindre[2] lors du prochain pas de temps. Nous choisissons d'exprimer la position de ce point en coordonnées polaires dans un repère lié au véhicule. Ce repère a pour centre le point de référence du véhicule, pour lequel nous choisissons le centre de l'essieu avant F. Son axe des abscisses est fixé parallèle à l'axe longitudinal du véhicule (voir figure 3.2).

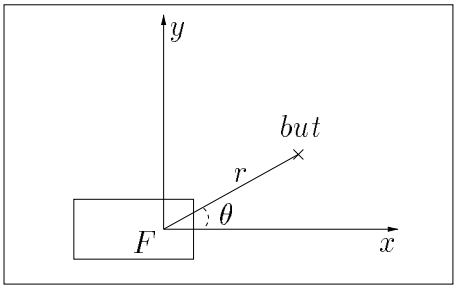

FIG. 3.2 – *Point à atteindre dans le repère lié au véhicule.*

Nous avons donc besoin d'un RNA possédant deux entrées (coordonnées polaires du point à atteindre) et deux sorties (vitesse et angle de braquage du véhicule). Nous préciserons plus loin le codage utilisé pour ces différentes valeurs.

1. Apprentissage réalisé en dehors des périodes d'utilisation du réseau comme système de commande.

2. Point sur lequel nous désirons placer le point de référence du véhicule.

3.3.2.2 Choix du réseau

Le choix du modèle de réseau de neurones artificiels utilisé est un critère essentiel pour la réalisation de notre système de commande. Il doit satisfaire plusieurs contraintes parmi lesquelles nous pouvons retenir principalement [138] :

- la capacité d'approcher n'importe quelle fonction (approximation universelle) ;
- une contrainte de couverture qui exprime le fait que pour chaque sortie y du réseau et pour toute entrée e, il doit exister au moins un paramètre (poids) w telle que la valeur de $\frac{\partial y}{\partial w}$ soit suffisamment grande dans le voisinage de e ;
- une contrainte de généralisation locale qui exprime le fait que si la valeur $\frac{\partial y}{\partial w}$ est importante dans le voisinage de e, elle doit décroître régulièrement lorsque l'on s'éloigne de ce voisinage.

Cette dernière propriété est particulièrement importante dans le cas d'un apprentissage en ligne. Son non-respect implique que la modification d'un seul paramètre peut changer le traitement effectué par le réseau sur l'ensemble de son domaine d'entrée. Puisque les exemples d'apprentissage utilisés dépendent de l'état actuel du système commandé (le véhicule dans notre cas) et des objectifs de la commande, ils ne peuvent être choisis librement. Il existe donc un risque de spécialisation excessive du réseau dans une partie seulement du domaine d'entrée si ces exemples ne sont pas uniformément répartis. Une telle spécialisation va alors de pair avec un «désapprentissage» dans les autres zones de l'espace d'entrée.
Nous avons vu dans la section 2.1.2.1 que les réseaux multi-couches entraînés par rétropropagation du gradient ne respectent pas cette contrainte de généralisation locale, ce qui les disqualifient pour notre application.
D. A. Pomerleau [131] apporte dans son système (voir §4.2.3.2) une solution à ce problème. Il propose en effet de conserver en permanence un ensemble d'exemples d'apprentissage représentatif de toutes les situations rencontrées. Le choix de ces exemples n'est cependant pas un problème simple, et le temps de calcul nécessaire à un pas d'apprentissage peut devenir non négligeable, si leur nombre est important.
Parmi les autres choix possibles nous avons retenu principalement deux types de réseaux : les réseaux à fonctions de base radiales et les réseaux de type CMAC que nous avons présentés respectivement dans les sections 2.1.2.2 et 2.1.2.3.
Ce deuxième modèle a été utilisé sur plusieurs types de problèmes en robotique [82]. Ces principaux atouts sont sa simplicité et surtout la rapidité de son apprentissage. Sa principale limitation provient de l'impossibilité de modéliser une fonction continue. Les valeurs de sortie sont en effet discrètes en raison de l'échantillonnage de l'espace d'entrée. Il est bien sûr possible de «choisir» la précision de l'approximation obtenue en jouant sur la taille et le nombre des champs récepteurs. Le nombre d'exemples d'apprentissage nécessaires risque toutefois de croître avec le nombre de ces cellules.
C'est principalement à cause de ce problème que nous avons opté, après avoir expérimenté les réseaux de type CMAC, pour l'utilisation d'un réseau de type RBF. Nous ne présentons donc dans la suite que les résultats obtenus avec ce modèle.

3.4 Résultats en simulation

3.4.1 Apprentissage

Nous présentons dans un premier temps les résultats obtenus, à partir d'un échantillon de données recueillies sur un véhicule simulé[1] supposé parfaitement conforme à notre modèle, en réalisant un apprentissage hors-ligne. Nous utilisons 1100 exemples correspondant aux mouvements associés à des vitesses[2] entre 0 et 5 $m.s^{-1}$ et des angles de braquage entre $-0,45$ et $0,45$ radians. Pour chacun de ces exemples, le déplacement du véhicule (différence entre la position finale et la position initiale) est enregistré[3] au bout d'un pas de temps $\Delta t = 0,5s$. Le mouvement est exprimé sous la forme des coordonnées polaires r et θ du centre de l'essieu avant du robot, dans le repère lié à la position de ce point avant le déplacement (voir figure 3.3).

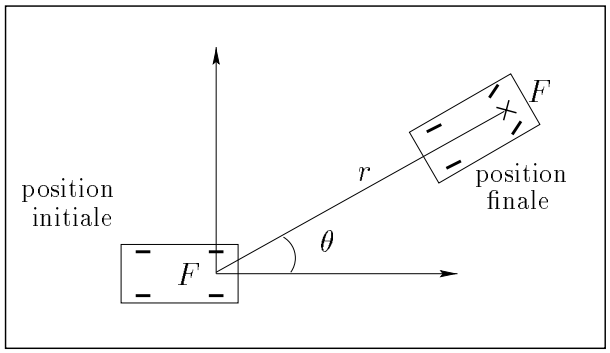

FIG. 3.3 – *Mesure du mouvement.*

Ces exemples sont séparés en deux ensembles que nous appellerons l'ensemble d'apprentissage (60 %) et l'ensemble de test (40 %). Seuls les exemples du premier ensemble sont utilisés durant l'apprentissage. Les autres servent à tester les capacités de généralisation du réseau.
Nous utilisons un réseau de type RBF normalisé, entraîné en utilisant l'algorithme présenté dans la section 2.1.2.2. Seuls les poids des liens entre les unités cachées et les unités de sortie sont soumis à un apprentissage[4]. Les centres des gaussiennes sont disposés suivant un treillis régulier, et demeurent fixes tout comme les variances.
Les deux entrées de notre réseau (coordonnées polaires r et θ correspondant au mouvement mesuré) sont ramenées respectivement aux intervalles [0;1] et [-1;1].

1. Le simulateur utilisé ainsi que le niveau de détail de la simulation sont décrits en annexe B.
2. La vitesse mesurée est celle du point de référence F du véhicule.
3. La mesure de ce déplacement constitue une difficulté supplémentaire sur le robot réel. Le dispositif utilisé sur notre robot (voir annexe A) est en effet assez peu précis pour des mouvements importants. Les résultats sont cependant très satisfaisants tant que l'on s'intéresse à des déplacements courts de l'ordre de ceux que nous mesurons ici.
4. Des tests réalisés en étendant l'apprentissage aux paramètres des fonctions gaussiennes ont fourni de moins bons résultats.

L'un des paramètres les plus importants à choisir est le nombre d'unités cachées utilisées dans le réseau. Nous avons réalisé plusieurs expériences en utilisant respectivement 16, 25, 36 puis 49 neurones cachés répartis de manière régulière sur le domaine d'entrée. La variance des gaussiennes est également une valeur essentielle pour le fonctionnement du système. Le réglage de ce paramètre fait généralement intervenir le calcul d'une valeur appelée taux de recouvrement, définie par :

$$\tau = \exp(-\frac{1}{2}\frac{(\frac{\delta}{2})^2}{\sigma^2}), \tag{3.5}$$

où δ est la distance séparant les centres des gaussiennes et σ^2 leurs variances. Cette valeur caractérise l'activation des unités pour une entrée située à une distance $\frac{\delta}{2}$ de leur centre (voir figure 3.4).

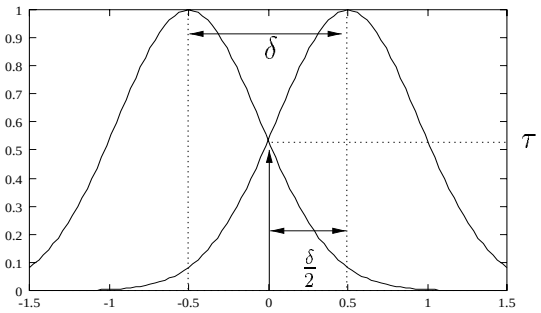

FIG. 3.4 – *Taux de recouvrement* τ.

Il est admis , qu'il existe un optimum de τ, situé entre 60 et 90 %, dépendant du nombre d'unités du réseau et de la fonction que l'on cherche à approcher [138]. Nous avons donc testé plusieurs valeurs de σ donnant des taux de recouvrement respectivement égaux à 60, 75 et 90 %.
Durant les expériences réalisées, la valeur du pas du gradient η a été fixée à 0,1. Les résultats obtenus avec les différents réseaux utilisés se sont avérés très proches, tant en termes d'erreur sur la base d'apprentissage que sur la base de test.
Nous présentons dans le tableau 3.1 les erreurs obtenues après 20 cycles complets d'entraînement. La mesure retenue est la moyenne de l'erreur quadratique (voir équation 2.1) sur l'ensemble des exemples.
Dans ce tableau n désigne le nombre d'unités du réseau, τ son taux de recouvrement, E_a l'erreur sur la base d'apprentissage et E_g celle mesurée sur la base de test. Les valeurs données sont des moyennes de mesures obtenues lors de 3 expériences différentes réalisées en initialisant de manière aléatoire les poids du réseau et en présentant les exemples dans un ordre également aléatoire. L'ensemble des réseaux utilisés a permis d'obtenir de très bons résultats et les différences observées semblent peu significatives (l'erreur initiale varie entre 2 et 3 soit près de 300 fois plus que les plus mauvais résultats). La convergence a cependant été plus ou moins rapide selon les cas. Le tableau 3.2 montre pour chaque réseau le nombre de présentations p de la base d'apprentissage

n	16	16	16	25	25	25
τ	60 %	75 %	90 %	60 %	75 %	90 %
E_a	0,00661	0,00283	0,00545	0,00372	0,00203	0,00577
E_g	0,00638	0,00282	0,00595	0,00361	0,00199	0,00622
n	36	36	36	49	49	49
τ	60 %	75 %	90 %	60 %	75 %	90 %
E_a	0,00208	0,00143	0,00401	0,00244	0,00196	0,00282
E_g	0,00226	0,00150	0,00432	0,00277	0,00198	0,00296

TAB. 3.1 – *Moyenne de l'erreur quadratique sur les bases de test et d'apprentissage après 20 pas d'entraînement.*

n	16	16	16	25	25	25	36	36	36	49	49	49
τ	60 %	75 %	90 %	60 %	75 %	90 %	60 %	75 %	90 %	60 %	75 %	90 %
p	5	4	10	4	4	16	6	6	14	6	10	10

TAB. 3.2 – *Nombre de pas d'entraînement nécessaires pour obtenir une erreur quadratique moyenne sur la base de généralisation inférieure à 0,007.*

nécessaires pour obtenir une erreur de généralisation inférieure à 0,007 (résultat obtenu par le plus mauvais des réseaux).
Cette vitesse de convergence est relativement importante dans le cas d'un apprentissage en ligne. Les meilleurs résultats ont été obtenus dans le cas des réseaux possédant 16 ou 25 neurones cachés, et pour un taux de recouvrement de 60 ou 75 %.
La figure 3.5 présente l'évolution de l'erreur du réseau, possédant 25 unités cachées et un taux de recouvrement de 75 %, que nous utiliserons dans la suite de ce chapitre.

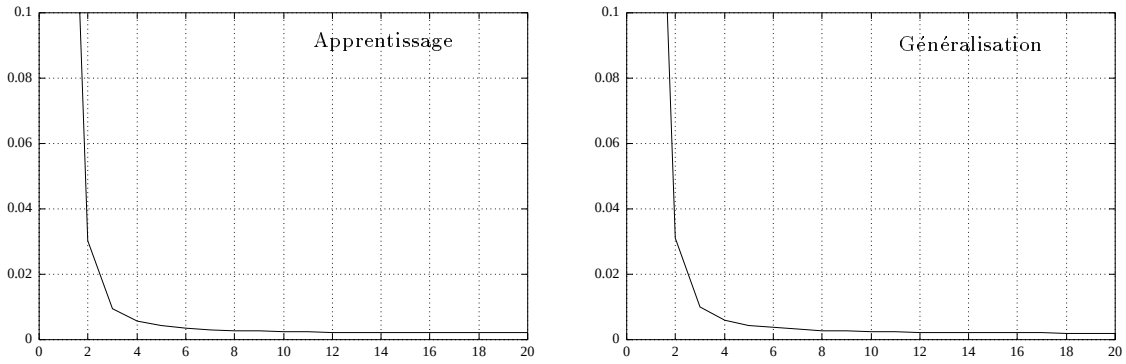

FIG. 3.5 – *Courbes d'erreur sur la base d'apprentissage et sur la base de test. La quantité portée en abscisse correspond au nombre de pas d'apprentissage.*

Les courbes d'erreur d'apprentissage et de test sont quasiment confondues. Dans les deux cas un pas d'apprentissage suffit à diviser l'erreur initiale par 10, deux pas suffisent à la diviser par 90.

3.4.2 Suivi de chemin

Notre système de commande est réalisé simplement en fournissant en entrée du réseau la différence entre la position désirée pour le point de référence du robot à l'instant $t_{actuel} + \Delta_t$[1] et la position actuelle de ce point.
Les consignes fournies par le réseau doivent théoriquement permettre d'atteindre le point désiré.
L'intervalle durant lequel ces consignes sont maintenues, avant le calcul de nouvelles valeurs est théoriquement indépendant de Δ_t. Nous avons toutefois choisi la même valeur $0,5\ s$ lors de nos simulations.
La vitesse curviligne (vitesse le long du chemin) du véhicule est contrôlée en choisissant la distance du point à atteindre sur la courbe. Elle peut dépendre de plusieurs paramètres comme la courbure du chemin suivi ou la proximité d'obstacles, ou être planifiée à l'avance et associée au chemin. Celui-ci peut alors être exprimé sous la forme d'une suite de points de passage correspondant aux différents intervalles de temps. Il s'agit plutôt de suivi de trajectoire dans ce dernier cas.
La figure 3.6 montre un exemple de suivi à la vitesse de $3\ m.s^{-1}$ sur une trajectoire d'une longueur proche de $200\ m$ présentant de nombreux changements de direction. Cette dernière a été obtenue à partir d'un planificateur, développé au sein du projet SHARP, respectant les contraintes cinématiques du véhicule. La principale caractéristique de ce planificateur est de produire des chemins dont la courbure est continue et compatible avec le rayon de braquage du véhicule (voir [145] pour plus de détails). Cette courbe peut donc être théoriquement suivie par le robot, sans manœuvres ni changements de vitesse.
La figure 3.7 présente l'erreur obtenue durant cette expérience. Cette erreur représente la distance entre le point de référence du robot et le point le plus proche du chemin suivi. Elle n'excède à aucun moment $8\ cm$.

3.4.3 Calcul d'un chemin de rattrapage local

Le réseau que nous utilisons n'est entraîné qu'à partir de mouvements d'une durée de $0,5\ s$. Ces mouvements ne permettent d'atteindre que des points relativement proches. Si le véhicule se trouve à un moment donné à une distance importante du chemin de référence, les mouvements nécessaires pour rejoindre celui-ci risquent de sortir du domaine sur lequel s'est déroulé l'apprentissage. Cette situation peut se produire par exemple si le véhicule effectue un évitement d'obstacle puis doit rattraper son itinéraire ou plus simplement si le chemin planifié ne respecte pas les caractéristiques du véhicule (e.g. changements de courbure trop brutaux).
Nous proposons pour résoudre ce problème d'utiliser, lorsque cela s'avère nécessaire des itinéraires de rattrapage pour le véhicule.

1. $\Delta_t = 0,5\ s$ est l'intervalle utilisé pour mesurer les mouvement lors de la collecte des données d'apprentissage.

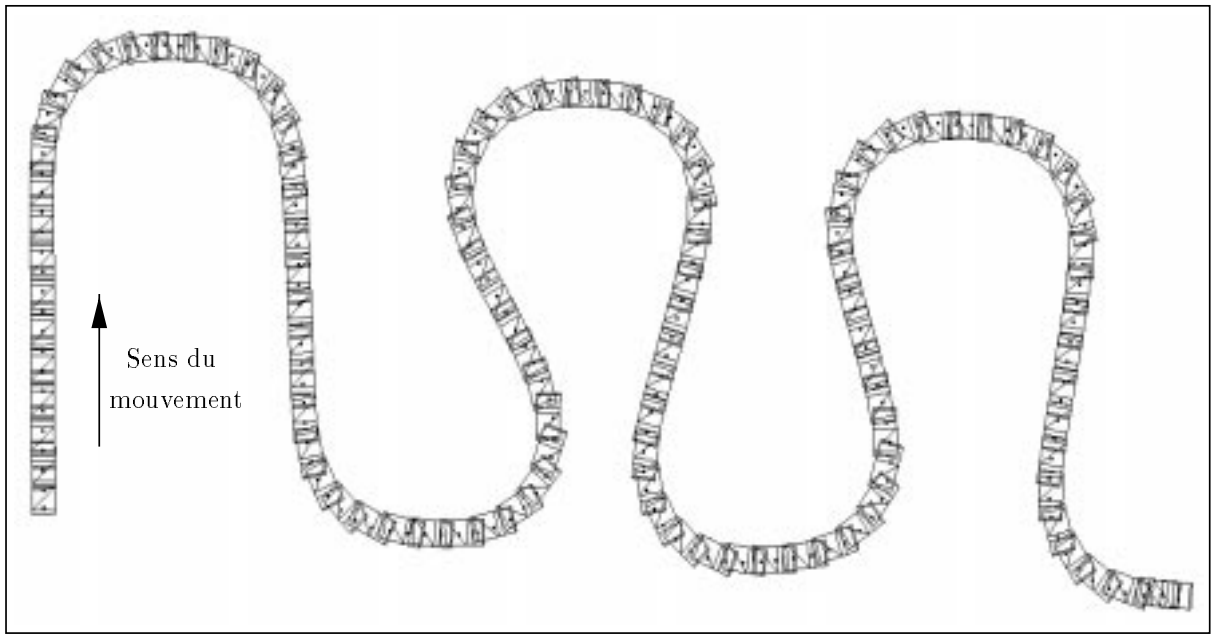

FIG. 3.6 – *Positions successives du véhicule lors d'un suivi de chemin.*

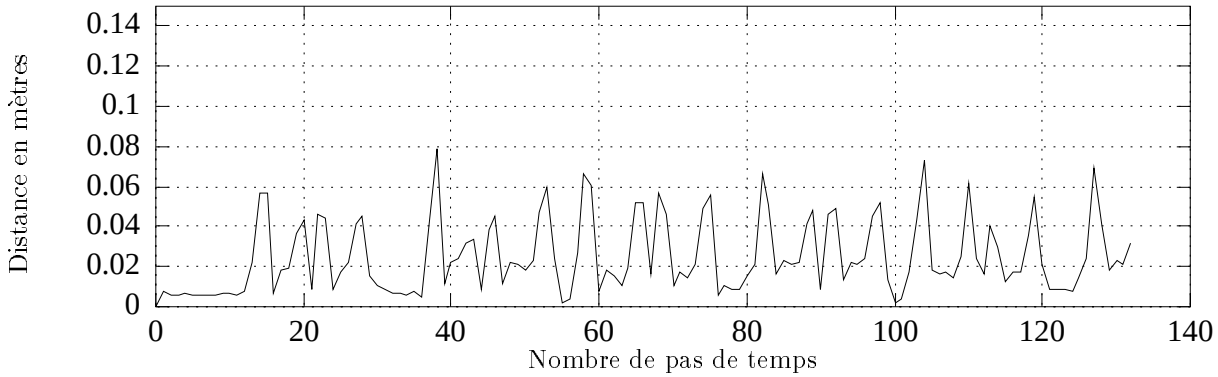

FIG. 3.7 – *Distance par rapport au chemin de référence durant le suivi.*

Notre méthode se base sur le calcul, à chaque pas de temps ou cela est nécessaire, d'un chemin joignant la position actuelle du véhicule (position du point de référence F) à un point de la trajectoire de référence. Cet itinéraire est alors substitué à celui de référence.

Afin de pouvoir garantir que le chemin de rattrapage peut être suivi par le robot il est nécessaire que ce chemin remplisse un certain nombre de conditions. La courbe employée doit respecter plusieurs contraintes, notamment au niveau de la position actuelle du robot et du point ou le chemin de référence est rattrapé. Le respect de l'ensemble de ces contraintes, dans le cas général, pose cependant un grand nombre de problèmes particulièrement complexes.

Nous avons donc préféré opter pour une approche permettant de «contourner» une partie de ces contraintes. Parmi les familles de courbes disponibles, nous avons en effet choisi pour des motifs de rapidité de calcul, un type de courbes simples, déjà employées par K. Yoshizawa et al. [169] sur un problème similaire, mais sur un type de robot mobile différent. Nous nous limitons en effet aux polynômes de degré 2 : $y = Ax^2 + Bx$. Le choix des paramètres A et B est relativement simple. Il doit permettre de respecter plusieurs contraintes imposées respectivement par :

- la continuité de la tangente à la courbe décrite par le robot (i.e. par son point de référence F), qui découle de la contrainte de non-glissement des roues énoncée en annexe A et exprimée par l'équation A.12 ;
- le rayon de giration minimum du véhicule qui impose une courbure maximum à notre chemin ;
- le rattrapage du chemin de référence lorsque ceci ne contrarie pas les deux points précédents.

Le nombre de paramètres de notre courbe ne permet pas toutefois de garantir la première contrainte (continuité de la tangente) au niveau du point de rattrapage (i.e point ou le chemin de référence est rejoint).
Nous choisissons donc pour ce point une position «fuyante» devant le véhicule. Notre technique consiste en effet à recalculer un nouvel itinéraire de rattrapage à chaque pas de temps. Le véhicule poursuit ainsi un point du chemin de référence qu'il ne peut jamais atteindre puisque ce point se déplace devant lui. Il vient par contre naturellement s'aligner sur le chemin de référence, par cette méthode.

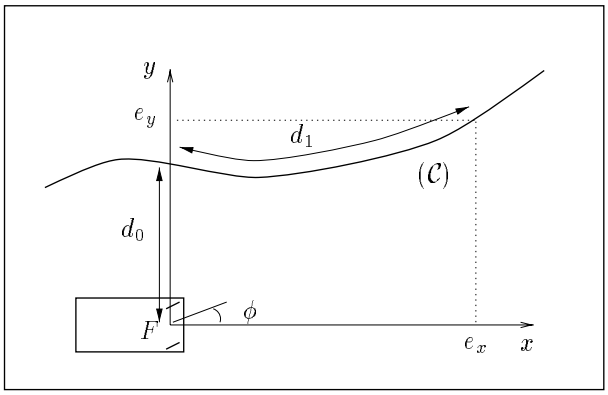

FIG. 3.8 – *Calcul d'une trajectoire de rattrapage.*

Le réglage des paramètres A et B s'effectue de la manière décrite ci-dessous. Si nous exprimons la courbe de rattrapage $y = f(x)$ dans le repère lié au point de référence du véhicule, et que l'on note e_x et e_y les coordonnées du point à rejoindre dans ce même repère (voir figure 3.8), les contraintes à respecter se traduisent de la manière suivante :

continuité de la tangente La tangente à la courbe suivie au moment du calcul par le point de référence F du robot, exprimée dans le repère lié à ce point est donnée par les équations A.8 et A.9. Cela nous donne :

$$\frac{dy}{dx} = tan(\phi),$$

où ϕ est l'angle de braquage du véhicule. La tangente à la courbe de rattrapage $y = Ax^2 + Bx$, exprimée dans le même repère, au point d'origine $x = 0$ (position actuelle de F) est donnée par :

$$\frac{dy}{dx} = 2Ax^2 + B = B$$

Nous obtenons donc :

$$B = tan(\phi) \tag{3.6}$$

borne du rayon de giration Notre chemin de rattrapage peut être caractérisé par sa courbure $\kappa = \frac{d\theta}{ds}$, où s est l'absice curviligne. La courbure maximum que peut suivre le point de référence F est donnée par l'équation A.4 :

$$\kappa_{max} = \frac{1}{\rho_{Fmin}} = \frac{sin(\phi_{max})}{l_w}$$

La courbure d'une courbe $y = f(x)$ est donnée par l'équation suivante issue de [112] :

$$\kappa = \frac{\frac{d^2y}{dx^2}}{[1 + (\frac{dy}{dx})^2]^{3/2}},$$

soit pour notre courbe :

$$\kappa = \frac{2A}{[1 + (2Ax + B)^2]^{3/2}} \tag{3.7}$$

Le maximum de la courbure est donc atteint au point $x = 0$. Puisque nous désirons que cette valeur soit inférieure à κ_{max}, il suffit de vérifier :

$$\left| \frac{2A}{(1 + B^2)^{3/2}} \right| \leq \kappa_{max},$$

soit :

$$|A| \leq A_{max} = \frac{\kappa_{max}(1 + B^2)^{3/2}}{2} \tag{3.8}$$

rattrapage du chemin de référence La courbe de rattrapage devant passer par le point (e_x, e_y) visé sur le chemin de référence, nous obtenons :

$$e_y = Ae_x^2 + Be_x,$$

et donc :

$$A = \frac{e_y - Be_x}{e_x^2} \tag{3.9}$$

Si cette valeur est supérieure au paramètre A_{max} calculé précédemment nous fixons $A = A_{max}$.

Nous pouvons noter que ces contraintes ne garantissent pas à elles seules que le chemin de rattrapage pourra être suivi. Elles ne font qu'assurer qu'il existe une vitesse et un angle de braquage du robot le permettant. Notre véhicule ne peut cependant pas changer instantanément l'orientation ou la vitesse de ses roues, et rien n'assure donc que les consignes calculées par le système de commande pourront être réellement appliquées. Toutefois même si le braquage et la vitesse du véhicule ne sont pas exactement ceux qui étaient spécifiés, ils tendent vers ces valeurs. La nouvelle courbe calculée au pas de temps suivant pourra donc être suivie dans de meilleures conditions.
Comme nous l'avons dit plus haut, nous ne nous occupons pas des caractéristiques de notre courbe au point où elle rattrape le chemin de référence. La position de ce point est choisie sur l'itinéraire de référence à une distance d_1 du point de ce chemin le plus proche du robot (voir figure 3.8).
Le choix d'une valeur de d_1 trop faible risquerait de provoquer des oscillations du robot autour du chemin. Ceci est d'autant plus vrai que la vitesse du robot est grande. Une distance trop importante ralentirait par contre le rattrapage.
Lors de nos simulations, la valeur de d_1 a donc été fixée arbitrairement de la manière suivante :

$$d_1 = \begin{cases} \beta d_0 v_F & \text{si } \beta d_0 v_F \leq 1 \\ \beta & \text{si } \beta d_0 v_F > 1 \end{cases} \tag{3.10}$$

où d_0 est la distance séparant le robot du point le plus proche sur le chemin de référence (voir figure 3.8), et β est une constante positive. Le point de rattrapage est ainsi choisi d'autant plus loin du robot que la distance entre ce dernier et le chemin de référence est grande et que la vitesse est importante.
Les expériences réalisées à des vitesses variant entre 0 et 5 $m.s^{-1}$ ont montrées que les meilleurs résultats étaient obtenus pour des valeurs de β autour de 10.
La dernière valeur à choisir est l'erreur limite à partir de laquelle il est nécessaire de calculer un chemin de rattrapage. Nous avons choisi pour cela de fixer une borne sur la distance d_0. Durant nos simulations cette borne a été fixée à $0,1v$, où v est la vitesse du véhicule. Si l'orientation désirée est connue, il est également possible de fixer une erreur maximum en orientation admissible.

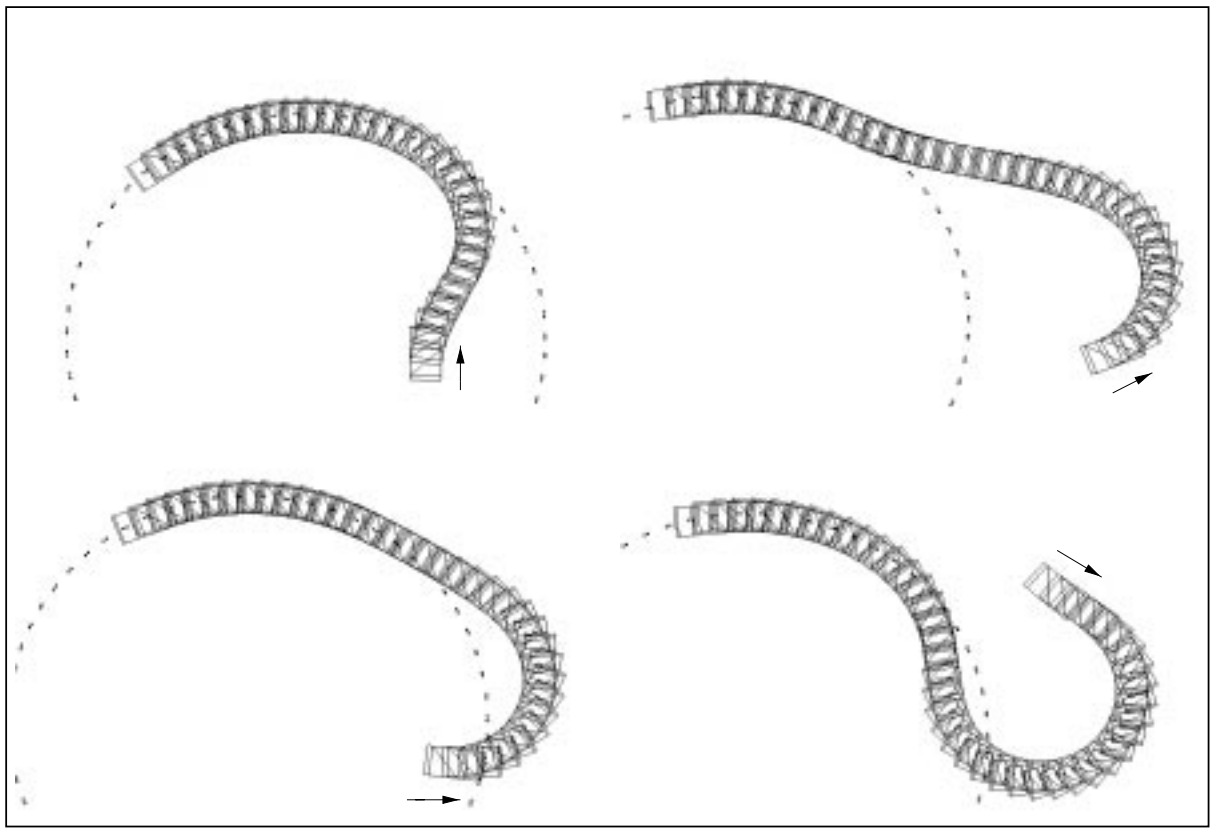

FIG. 3.9 – *Rattrapage d'un chemin circulaire à partir de plusieurs positions initiales différentes. Le chemin de référence est représenté par les pointillés. Les flèches montrent le sens du mouvement.*

La figure 3.9 montre les positions successives du véhicule lors de plusieurs exemples de rattrapage d'un chemin circulaire. La vitesse du véhicule est fixée à $3m.s^{-1}$ durant ces expériences.
La figure 3.10 présente les résultats obtenus lors du suivi d'un chemin composé uniquement d'arcs de cercles et de lignes droites. Il a été démontré par Dubins [34], que ce type de courbes représentaient le plus court chemin faisable pour un véhicule de type voiture. Un suivi de ces courbes impose cependant un arrêt lors de chaque transition entre une droite et un arc de cercle ou entre deux arcs de cercle afin de réorienter les roues, puisque la courbure change brutalement en ces points. Dans notre simulation, qui se déroule à la vitesse constante de $3m.s^{-1}$, le véhicule est donc dans l'impossibilité de suivre la trajectoire de référence de manière parfaite, et doit ensuite la rattraper.

3.4.4 Apprentissage en cours d'utilisation

L'objectif principal de notre système de commande est de permettre une adaptation à certains changements des paramètres du véhicule ou de l'environnement en cours

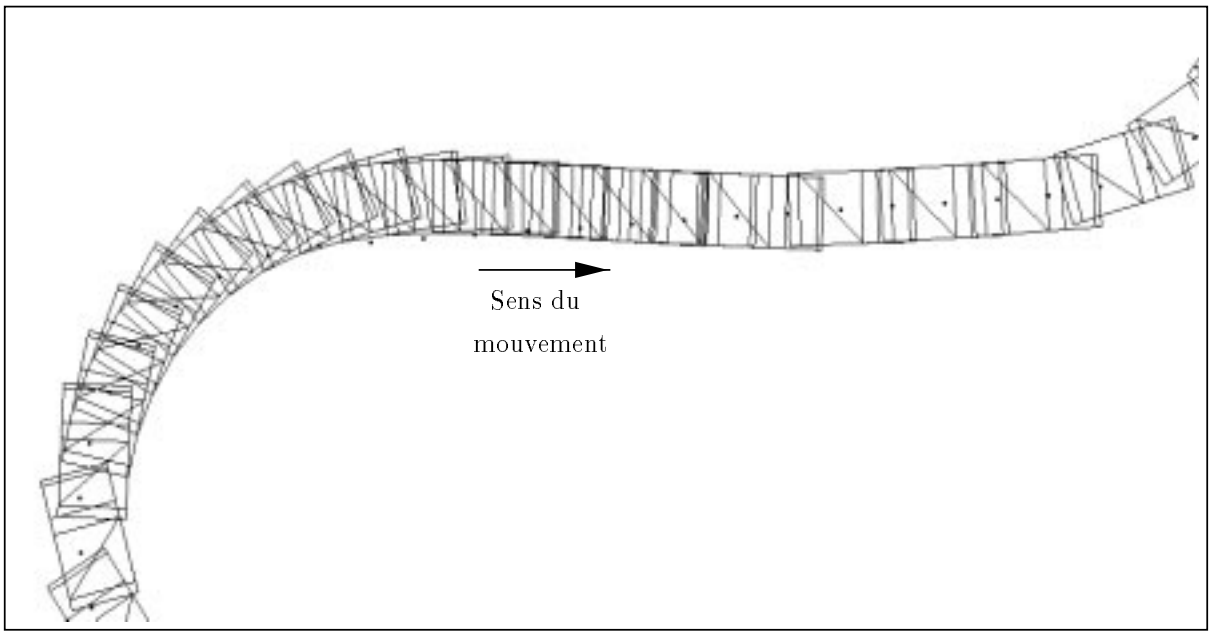

FIG. 3.10 – *Suivi d'une courbe de Dubins à vitesse constante. Le chemin de référence est représenté en pointillés. Le changement de courbure est trop brutal pour permettre un suivi parfait.*

d'utilisation. Nous nous sommes donc intéressés à l'application d'un apprentissage en-ligne à notre contrôleur. Notre méthode consiste à présenter au réseau, sans arrêter le véhicule, l'exemple composé du mouvement effectué lors du dernier pas de temps et des consignes l'ayant provoqué.

La principale difficulté rencontrée n'est pas paradoxalement l'apprentissage lui-même, mais plutôt la détection des périodes où cet apprentissage est nécessaire (i.e. les périodes où les résultats du suivi sont anormaux). Il est en effet difficile de savoir si le système de commande fonctionne correctement ou non. Lorsque le mouvement effectué par le robot ne correspond pas à celui qui est désiré, cela peut provenir d'une défaillance du véhicule mais aussi tout simplement du fait que les consignes n'ont pu être suivies immédiatement. Puisque l'accélération et la vitesse de braquage du robot ne sont pas infinies, rien ne garantit que la vitesse et/ou le braquage désirés sont atteints dès le début du pas de temps.

Nous avons recensé deux possibilités pour détecter un éventuel dysfonctionnement du contrôleur :

- ne s'intéresser qu'aux pas de temps où les consignes sont réellement appliquées dès le début du mouvement. Cette méthode fonctionne relativement mal car lorsque le système de commande n'est pas adapté au véhicule, cela se traduit par des mouvements changeant sans cesse, et le robot ne se stabilise quasiment jamais sur une consigne particulière ;

- s'intéresser au sens des erreurs obtenues lors du mouvement. Si un problème particulier comme ceux que nous avons évoqués au début de ce chapitre (voir §3.1) se produit, la différence entre le mouvement mesuré et le mouvement désiré possède toujours la même direction. C'est cette approche que nous avons retenue dans nos expérimentations. Nous nous sommes intéressés aux pourcentages de pas de temps où les positions obtenues se trouvent à gauche, à droite, en avant ou en arrière de celles désirées. L'apprentissage peut être déclenché lorsqu'un de ces pourcentages devient trop important.

Il est également possible, au prix d'une très légère dégradation des performances de réaliser un apprentissage continu, que le système fonctionne correctement ou non.

Un problème similaire se pose au niveau de l'apprentissage. Les consignes données n'étant pas suivies instantanément [1], le mouvement mesuré ne correspond pas forcement à leur application. L'utilisation des couples {mouvement mesuré ; consigne} comme exemples pour entraîner le réseau n'est donc pas forcement pertinente. Nous avons testé plusieurs heuristiques pour résoudre ce problème :

- sélectionner les pas de temps où les consignes sont réellement suivies des le début. Comme nous venons de le voir, ceci est relativement rare et cette solution donne d'assez mauvais résultats ;

- utiliser tous les exemples disponibles sans se soucier de l'état réel du véhicule. Cette solution donne également d'assez mauvaises performances ;

- utiliser à la place des consignes appliquées au robot, la vitesse et l'angle de braquage moyen de ce dernier au cours du pas de temps. Cette méthode a donné de loin les meilleurs résultats.

Nous avons opté pour la dernière de ces trois solutions. Il s'agit toutefois d'une heuristique et ceci explique la très légère baisse de performance constatée lorsque l'apprentissage est réalisé en continu sur un réseau ayant déjà été entraîné hors-ligne sur le même véhicule. La figure 3.11 présente avec les notations du chapitre 2 le diagramme du système obtenu.

Nous exposons ci dessous les résultats obtenus pour un véhicule souffrant d'une erreur sur la mesure de son angle de braquage ϕ. Une telle erreur pourrait correspondre sur le robot dont nous disposons à un mauvais étalonnage ou à un dysfonctionnement du dispositif de mesure. Elle entraîne également une erreur de commande puisque l'asservissement du braquage se fait en réduisant la distance entre l'angle désiré et l'angle mesuré. Nous pouvons noter qu'une roue dégonflée provoquerait un comportement analogue du point de vue qualitatif. Nous avons choisi pour ce test une erreur relativement

1. Le véhicule simulé ainsi que le robot dont nous disposons mettent environ 2 secondes pour faire évoluer leur braquage d'une butée à l'autre.

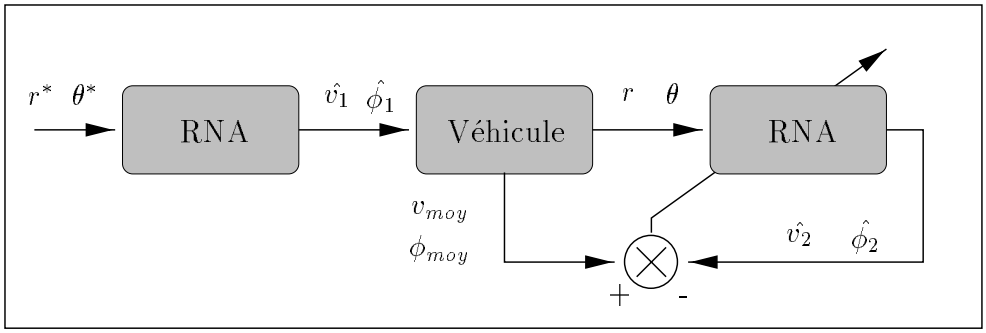

FIG. 3.11 – *Apprentissage en-ligne. Les deux éléments notés RNA correspondent au même réseau. Les consignes* $\{\hat{v_1}\ ;\ \hat{\phi_1}\}$ *sont choisies par le réseau afin d'obtenir le mouvement désiré* $\{r^*\ ;\ \theta^*\}$*. Les consignes moyennes réellement appliquées au robot sont notées* v_{moy} *et* ϕ_{moy}*. Celles-ci provoquent un mouvement* $\{r\ ;\ \theta\}$*. Ce mouvement est à nouveau présenté au réseau qui estime qu'il correspond à l'application des consignes* $\{\hat{v_2}\ ;\ \hat{\phi_2}\}$*. La différence entre* $\{v_{moy}\ ;\ \phi_{moy}\}$ *et* $\{\hat{v_2}\ ;\ \hat{\phi_2}\}$ *peut donc être utilisée comme mesure de l'erreur du réseau.*

importante de 0,15 radians. Le chemin utilisé est le même que celui présenté sur la figure 3.6. La vitesse reste elle aussi de 3 $m.s^{-1}$.
La figure 3.12 montre les erreurs en distance[1] obtenues durant ce test en utilisant la loi de commande de Kanayama (voir section 3.2.1) ainsi que notre système de commande, sans réaliser d'apprentissage en-ligne. La loi de Kanayama a été testée avec des pas de temps de 0, 1 s puis 0, 5 s. L'erreur obtenue est très importante dans tous les cas. Elle varie entre 40 cm et 2 m.

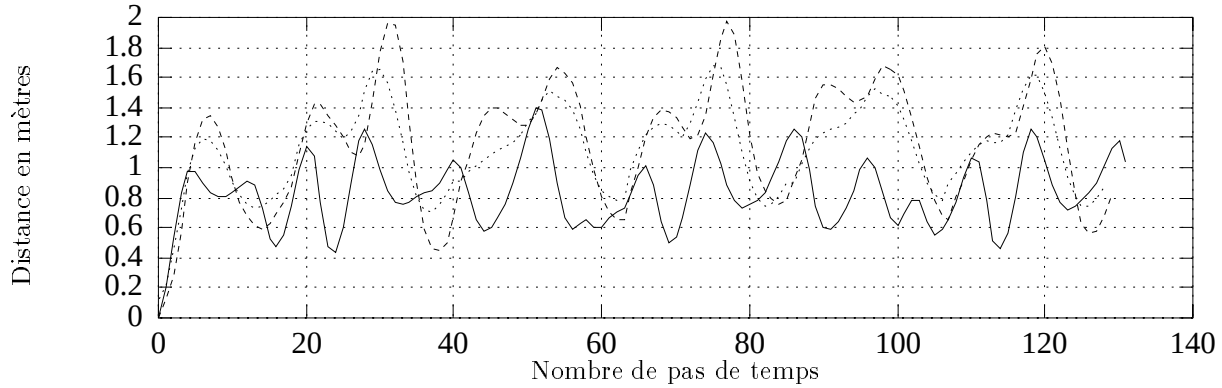

FIG. 3.12 – *Distance par rapport au chemin de référence durant le suivi. La courbe continue correspond à notre système de commande utilisé sans apprentissage. Les courbes constituées de pointillés et de tirets correspondent à la loi de commande de Kanayama utilisée respectivement avec un pas de temps de 0,1 puis 0,5 secondes.*

La figure 3.13 présente l'évolution de l'erreur en distance lorsqu'un apprentissage en-

1. Distance entre le point de référence du véhicule et le point le plus proche sur le chemin.

ligne est effectué. La courbe continue correspond à un premier passage du véhicule sur le chemin. Celle en pointillés montre l'erreur obtenue lors d'un deuxième passage du même véhicule sur le chemin sans réinitialisation du réseau. Le pas du gradient η a été fixé durant cette expérience à 0,2.
Nous pouvons constater que la distance entre le véhicule et le chemin suivi décroît rapidement jusqu'à devenir négligeable lors du deuxième passage.

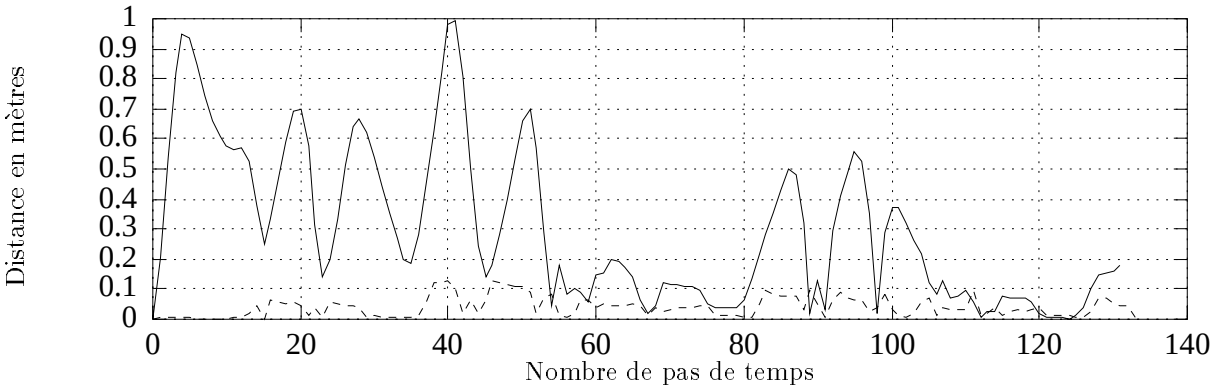

FIG. 3.13 – *Distance par rapport au chemin de référence durant le suivi. La courbe continue correspond à notre système de commande subissant un apprentissage en-ligne. La courbe constituée de tirets correspond à un second passage sur le même chemin.*

Enfin la figure 3.14 présente les résultats obtenus par notre système de commande avec un réseau entraîné hors ligne sur ce véhicule. Le suivi est aussi bon que dans le cas du véhicule normal (i.e. ne souffrant d'aucune erreur de mesure).

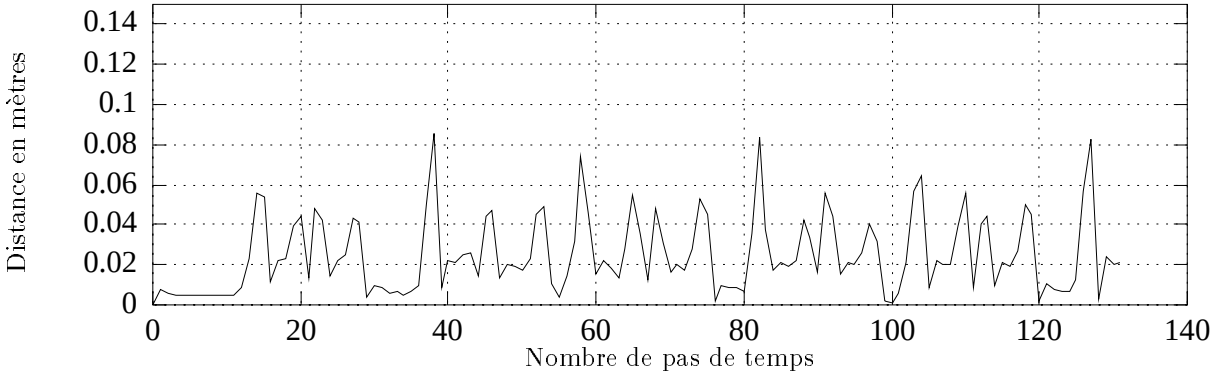

FIG. 3.14 – *Distance par rapport au chemin de référence durant le suivi dans le cas d'un véhicule entraîné hors-ligne.*

Durant l'apprentissage en-ligne, le réseau ne modifie son traitement que pour la fraction du domaine d'entrée, correspondant aux exemples qui lui sont présentés. La figure 3.15 montre la consigne de braquage donnée par le réseau en fonction de l'orientation θ et de la distance r du point à atteindre[1]. La surface en pointillés correspond au réseau

1. Ces valeurs sont ramenées aux intervalles $[-1;1]$ et $[0;1]$.

initial (avant l'apprentissage en-ligne). Celle dessinée en continu montre la sortie après apprentissage pendant deux suivis du chemin. Puisque l'expérience s'est déroulée à la vitesse constante de 3 $m.s^{-1}$, la sortie du réseau n'est modifiée que pour des points atteints avec cette vitesse ($r \approx 0,6$).

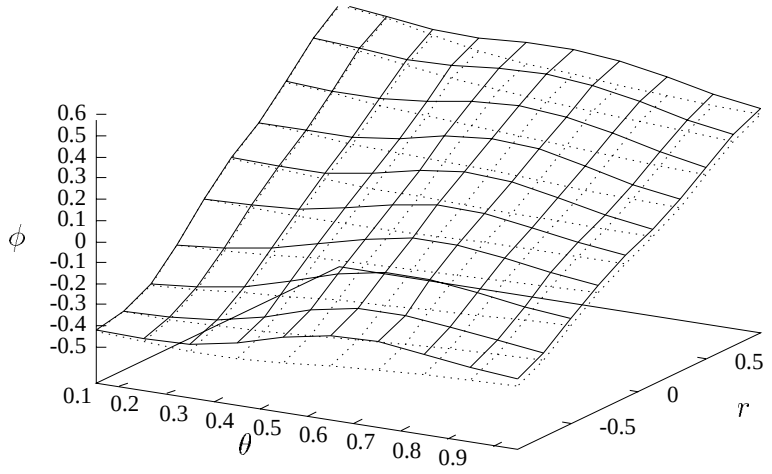

FIG. 3.15 – *Consigne en angle de braquage en fonction des coordonnées du point à atteindre.*

La deuxième expérience que nous présentons se rapporte à un autre type de dysfonctionnement de la mesure de l'angle de braquage. Cette mesure correspond pour le véhicule que nous considérons ci-dessous à la moitié de la valeur réelle de ϕ[1]. La figure 3.16 présente l'erreur en distance obtenue par notre contrôleur sans apprentissage (courbe continue) puis avec apprentissage en-ligne (courbe en pointillés).

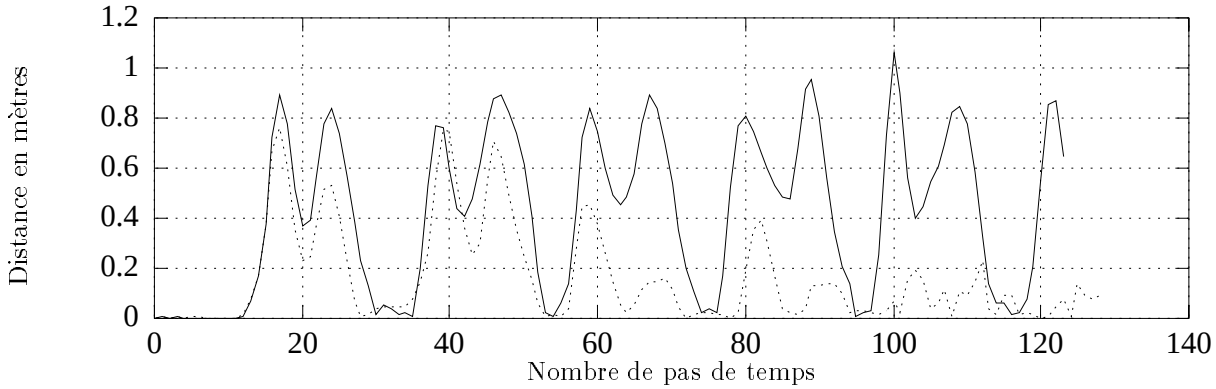

FIG. 3.16 – *Distance par rapport au chemin de référence durant le suivi. La courbe continue correspond à notre système de commande sans apprentissage. La courbe en pointillés correspond à notre système de commande subissant un apprentissage en ligne.*

1. Ceci correspond à une panne réellement observée sur notre robot.

La distance obtenue est ici cyclique car l'erreur de mesure s'annule lorsque le braquage désiré vaut zéro et le suivi redevient donc correct chaque fois que le chemin est rectiligne. Les tests réalisés en utilisant la loi de commande de Kanayma on donné un comportement erratique du véhicule qui oscille de manière violente.
Le dernier test que nous présentons correspond à une défaillance du dispositif de mesure de la vitesse. Nous avons supposé une vitesse mesurée inférieure d'un tiers à la vitesse réelle. Cela se traduit par un véhicule se déplaçant plus rapidement que désiré. La figure 3.17 présente dans ce cas l'évolution de la vitesse du véhicule lorsqu'un apprentissage est réalisé. La courbe continue correspond à un premier parcours du chemin. Celle constituée de pointillés montre la vitesse obtenue lors d'un deuxième passage sur le même chemin sans réinitialisation du réseau.

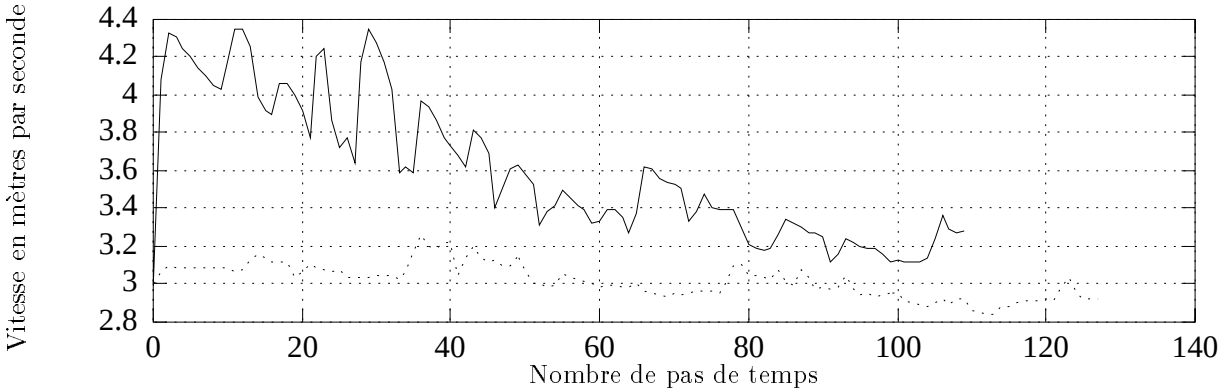

FIG. 3.17 – *Vitesse durant le suivi. La courbe continue correspond à notre système de commande subissant un apprentissage en-ligne. La courbe en pointillés correspond à un second passage sur le même chemin.*

Nous pouvons constater que la vitesse initialement supérieure d'un tiers à celle désirée ($4\ m.s^{-1}$) décroît assez rapidement. Elle ne se stabilise cependant pas[1] et oscille entre 2,9 et 3,1 $m.s^{-1}$.

Il est intéressant de noter que lors d'un apprentissage en-ligne, si le mouvement du véhicule ne correspond pas à celui qui était attendu, il n'existe aucun moyen de connaître la commande qui aurait permis d'obtenir celui-ci. Nous ne disposons donc pas des patrons {mouvement désiré ; consigne nécessaire}. Les seuls exemples disponibles sont ceux constitués des couples {mouvement mesuré ; consigne appliquée}. Les résultats montrent toutefois que cette information est suffisante pour obtenir de bonnes performances. Ceci est dû aux capacités de généralisation du réseau qui permettent d'étendre le gain de performance, provoqué par l'apprentissage d'un exemple correspondant à un mouvement particulier, aux mouvements proches de celui-ci.

1. La vitesse se stabilise uniquement après un temps d'apprentissage beaucoup plus long.

Les expériences que nous avons exposées montrent de bons résultats dans le cas de perturbations très importantes du fonctionnement du véhicule. Nous avons toutefois obtenu de moins bonnes performances dans deux cas particuliers :

- lorsque la longueur des mouvements réalisés par le véhicule dépasse la valeur maximum utilisée pour normaliser l'entrée r du réseau (i.e. la ramener à l'intervalle $[0;1]$). Notre contrôleur est bien sûr incapable d'adapter correctement sa consigne en vitesse dans ce cas, puisque la valeur de r qu'il reçoit reste fixée à 1 quel que soit le mouvement ;
- lorsque l'erreur de mesure de l'angle de braquage du véhicule devient trop importante. Le calcul des chemins de rattrapage est alors faussé et le véhicule peine à rejoindre rapidement le chemin de référence.

Ces deux cas correspondent cependant à des dysfonctionnements très importants pour lesquels les autres méthodes de suivi que nous avons testées sont également inopérantes.

3.5 Conclusion

Au cours de ce chapitre nous avons présenté une approche originale pour le suivi de trajectoire, utilisant les techniques des réseaux de neurones artificiels. Cette méthode de commande permet de modifier en permanence le fonctionnement du contrôleur, lorsque les conditions d'utilisation le demandent. Elle a été testée avec succès sur un simulateur reproduisant le comportement d'un véhicule réel. Les tests sur le robot expérimental dont nous disposons sont en cours et nous espérons obtenir des résultats très prochainement.

Notre système de commande souffre cependant de l'absence d'une preuve de stabilité. L'obtention d'une telle preuve supposerait dans un premier temps, l'emploi d'un autre type de courbe de rattrapage. Il nous faudrait en effet pouvoir garantir que le robot est capable de suivre à tout moment la trajectoire qui lui est proposée, ce qui n'est pas le cas actuellement. Il nous semble donc intéressant de poursuivre le travail réalisé, en examinant les possibilités offertes par d'autres familles de courbes. Parmi les candidats possibles, nous pouvons notamment retenir les polynômes de degré supérieur à ceux que nous utilisons [112], les courbes de Bézier [147] ou encore les clothoides [75].

Le système de commande que nous venons de présenter ne confère qu'une autonomie minimum au robot qui reste dépendent d'un planificateur de trajectoires. Nous nous intéressons donc dans le chapitre suivant à un autre type de contrôleur permettant au véhicule d'évoluer en se basant sur ses données perceptives.

Chapitre 4

Réseaux de neurones artificiels et commande référencée capteurs

Au cours de ce chapitre nous présentons une méthodologie pour la réalisation de systèmes de commande réactifs, spécialisés dans l'accomplissement de manœuvres ou de mouvements simples et basés sur les capteurs extéroceptifs du robot. Cet exposé débute par une présentation des différents travaux appliqués à la résolution de ce problème rencontrés dans la littérature, et particulièrement de ceux utilisant les réseaux de neurones artificiels.
Notre approche, basée sur l'utilisation des RNA et de la logique floue, permet à la fois une spécification rapide de la tâche à accomplir, et un réglage précis du comportement attendu pour le véhicule. Elle est testée sur simulateur à travers la réalisation de plusieurs manœuvres.

4.1 Objectifs

Afin d'acquérir une réelle autonomie de mouvement notre véhicule doit être capable de réaliser un certain nombre de manœuvres en se basant uniquement sur ses données perceptives. On parle généralement de *commande référencée capteur* pour qualifier le traitement, qui se résume à l'utilisation d'une transformation de l'espace des données capteurs vers l'espace des actionneurs du robot, nécessaire pour ce type de tâches.
Comme nous l'avons vu dans la section 1.4, nous souhaitons disposer pour cela d'une bibliothèque de systèmes de commande, spécialisés chacun dans un type de mouvement particulier. Les manœuvres considérées peuvent faire partie de la mission du véhicule, comme par exemple dans le cas d'une manœuvre de parking ou intervenir lorsqu'un événement perturbe le suivi du plan, comme dans le cas de l'évitement d'un obstacle imprévu. Au cours de ces déplacements nous désirons pouvoir spécifier précisément l'itinéraire du véhicule. Notre but n'est pas uniquement la réalisation d'un mouvement sans collisions. La valeur d'un certain nombre de paramètres comme la distance par rapport à des types d'obstacles particuliers ou aux bordures des voies de circulation doivent pouvoir être garanties. Nous désirons également obtenir autant que possible des

mouvements «lisses» et réguliers présentant un minimum d'oscillations et d'à-coups. Ces critères sont cruciaux dans le cas d'un véhicule de grande taille évoluant à des vitesses importantes, sur des voies où il ne se trouve pas seul.
Les difficultés inhérentes à la réalisation de ces manœuvres sont multiples. La principale d'entre elles est liée à la pauvreté des informations délivrées par les capteurs extéroceptifs de notre véhicule. Les seules données disponibles proviennent de capteurs télémétriques ultrasonores. Le principe de fonctionnement de ces derniers est basé sur la mesure du temps de vol d'une onde acoustique émise par le capteur et réfléchie par les obstacles. La fiabilité des informations ainsi obtenues dépend d'un certain nombre de facteurs [1] :

- la surface des obstacles qui influe sur la réflexion des ondes ultrasonores. Un objet ayant une surface régulière est plus facilement détecté qu'un objet irrégulier. Certaines matières peuvent également absorber les ondes ;

- l'angle d'incidence de l'onde sur les objets de l'environnement. La détection est aléatoire sinon impossible si l'axe de cette onde est trop éloigné de la perpendiculaire à la surface de l'obstacle ;

- les réflexions multiples qui peuvent créer de faux échos ;

- les interférences crées par l'utilisation de plusieurs capteurs qui peuvent fausser la mesure.

De même lorsqu'un écho est obtenu il est impossible de connaître la position précise de l'obstacle à l'intérieur du cône d'émission du capteur (seule la distance est connue).

Les propriétés cinématiques du véhicule compliquent également la tâche. Il n'est pas possible, comme cela l'est sur un grand nombre de robots expérimentaux de laboratoire, de diriger le véhicule dans n'importe quelle direction à n'importe quel moment. Les contraintes non-holonomes (voir annexe A.2.2) liées à un robot de type voiture restreignent en effet fortement les déplacements possibles à un moment donné. Les différentes manœuvres réalisées doivent donc être anticipées suffisamment tôt pour que le mouvement puisse être mené à terme. Ceci justifie d'autant plus la spécialisation des contrôleurs réalisés.

4.2 Description des approches courantes

Le lien entre la perception et l'action pour un robot mobile, est un thème qui a été abordé en utilisant de nombreuses démarches différentes. Un certain nombre de celles-ci ont été testées au sein du projet SHARP. Dans cette section nous présentons les familles d'approches les plus courantes.

4.2.1 Champs de potentiels fictifs

Les premiers travaux sur les champs de potentiels fictifs ont été proposés par O. Khatib [79] pour la commande d'un bras manipulateur. Le principe de cette approche est relativement simple. Il consiste à faire évoluer le robot dans un champ de forces (ou champ de potentiels) guidant ses mouvements. Aux obstacles sont associés des potentiels répulsifs (positifs) alors que le ou les buts génèrent des potentiels attractifs (négatifs). Le calcul des consignes se fait de manière à diriger le robot vers une zone où la fonction de potentiel obtenue est minimum, généralement par une méthode de type descente de gradient. Le robot est donc naturellement repoussé par les obstacles et attiré par le but.

Si cette approche est plutôt destinée à des applications de planification, elle a également été utilisée pour la réalisation de systèmes de commande réactifs. Les obstacles pris en compte sont alors les représentations sur une carte locale de l'environnement, des données fournies par les capteurs du robot [20, 62]. Les schémas moteurs proposés par R. C. Arkin [8] et présentés dans le premier chapitre de ce mémoire (§1.3.2.1) fonctionnent également suivant un principe très proche de celui-ci.

B. H. Krogh et C. E. Thorpe [83] introduisent la notion de potentiels généralisés. Ces derniers ne dépendent plus uniquement de la position des obstacles, mais également du vecteur vitesse actuel du robot. Il est ainsi possible de prendre en compte les contraintes cinématiques du véhicule. Cette technique a également été utilisée par M. Hassoun [55] pour un problème d'assistance à la conduite sur un véhicule de type voiture.

Ces approches souffrent cependant de plusieurs limitations. La plus fréquemment citée concerne l'existence de minima locaux de la fonction de potentiel, dans lesquels le robot peut rester bloqué. Plusieurs voies ont été explorées pour résoudre ce problème. Pour plus de précisions sur ces travaux le lecteur pourra se reporter à la thèse de P. Reignier [135]. Toutefois ce point n'est pas crucial pour le type de systèmes que nous cherchons à réaliser puisque nous désirons spécialiser nos contrôleurs dans des mouvements bien particuliers pour lesquels il est sans doute possible d'identifier les minima locaux.

Reignier observe par contre des problèmes d'oscillations et souligne que ceux-ci sont inhérents à ces approches. Il est de plus très difficile de spécifier précisément à l'avance le type de mouvements attendus. Cela impliquerait un réglage méticuleux des champs de forces associés à chaque obstacle ou à chaque écho capteur.

4.2.2 Logique et commande floue

Les systèmes de commande flous que nous avons présentés dans la section 2.2.1.3 ont largement été utilisés dans le cadre de la robotique mobile.

Ces travaux sont motivés par la capacité qu'ont les systèmes flous de travailler à partir de données imprécises ou incertaines, telles que celles que délivrent les capteurs extéroceptifs utilisés en robotique, mais aussi par la possibilité qu'ils offrent de spécifier dans des termes proches du langage courant le comportement attendu pour le véhicule.

Tout comme dans le cas des approches basées sur les RNA, une des caractéristiques

de ce type de contrôleur est en effet de ne pas nécessiter une modélisation du système commandé. Plutôt que le processus lui-même, c'est la manière de le commander qui est décrite par les règles floues [41].
Les règles régissant le comportement désiré pour un véhicule peuvent donc être dans un grand nombre de cas relativement simples à édicter. Il peut s'agir par exemple de règles du type :

Si des obstacles sont proches alors consigne de vitesse est faible,

ou encore :

Si des obstacles sont proches devant alors consigne de direction est tourner droite.

L'un des premiers exemples de travaux utilisant la logique floue en robotique mobile est celui de la voiture floue de M. Sugeno [150]. Le but qui y est recherché est de permettre à un véhicule de type voiture de se garer sur une place de parking. Le système utilise trois informations (i.e. variables d'entrée) qui sont les distances par rapport aux murs latéraux et frontaux, ainsi que l'angle de l'axe longitudinal du véhicule par rapport à l'axe de la place de parking.
Plusieurs auteurs ont cherché à utiliser des contrôleurs flous prenant pour entrée les données fournies par des capteurs proximétriques. Parmi ceux-ci nous pouvons citer S. Ishikawa [68] qui propose un ensemble de règles implémentant en simulation à la fois des comportements de suivi de chemin et d'évitement d'obstacle pour un robot holonome se déplaçant à la vitesse maximale de 30 $cm.s^{-1}$. Les entrées utilisées sont les valeurs délivrées par des capteurs simulés supposés parfaits, ainsi que les variations de ces mêmes valeurs. Un deuxième contrôleur flou permet de donner des poids différents aux règles codant les deux comportements suivant la situation. F. Pin [129] propose une réalisation «hardware» sur un robot réel d'un système similaire utilisant très peu de règles. K. T. Song et J. C. Tai [149] présentent un contrôleur fonctionnant également à partir des données capteurs, pour un véhicule non-holonome et constatent la présence de nombreuses oscillations qu'ils tentent de supprimer en introduisant des obstacles et des sous-buts virtuels. P. Garnier [41] et P. Reignier [135] constatent eux aussi sur des applications assez proches des problèmes d'oscillations et concluent tous les deux à la nécessité d'un réglage précis du système par apprentissage supervisé.
De nombreux auteurs se sont également intéressés à l'utilisation de la logique floue pour la fusion de comportements réalisés de manière indépendante les uns des autres. Ces travaux sont passés en revue par A. Saffiotti [143].

Il apparaît donc que l'utilisation de la logique floue pourrait être un moyen de coder aisément le type de comportement attendu pour notre véhicule durant les manœuvres que nous cherchons à réaliser. Le réglage précis des mouvements obtenus est cependant d'une toute autre complexité. Un problème d'oscillations semble également lié à ces difficultés de réglage, et ce d'autant plus que le robot évolue à des vitesses importantes. Cette limitation rend l'emploi d'un contrôleur flou classique du type de ceux que nous venons de présenter assez mal adapté à l'application que nous visons. Nous reviendrons cependant plus loin sur les possibilités offertes par le réglage précis d'un tel système de

commande par un apprentissage supervisé.

4.2.3 Approches basées sur l'utilisation des RNA

Les capacités qu'ont les réseaux de neurones artificiels de traiter des données bruitées, de généraliser leurs connaissances à des situations nouvelles et surtout d'améliorer leurs performances à partir d'expériences passées ont motivées de nombreux travaux en robotique mobile. Nous présentons dans cette section quelques uns d'entre eux parmi les plus représentatifs.

4.2.3.1 Utilisation d'un apprentissage par renforcement

Le paradigme de l'apprentissage par renforcement que nous avons présenté dans la section 2.5.3 est particulièrement séduisant dans le cadre d'une application à la robotique mobile. Dans ce type d'approche, le robot est laissé libre d'explorer l'espace des actions possibles afin de sélectionner celles qui maximisent un critère donné, dépendant de la tâche à accomplir. Il n'est nullement nécessaire de lui spécifier les actions à entreprendre lors d'une phase préalable d'apprentissage supervisé.
Le véhicule peut lui-même choisir la nature des informations perceptives pertinentes pour déclencher telle ou telle action, et construire sa propre représentation interne de l'environnement. Ceci est très important puisque la perception qu'il a du monde extérieur est forcement différente de la notre, étant donné la nature des capteurs utilisés. Lors d'un apprentissage supervisé, réalisé par exemple en imitant un pilote humain, les informations utilisées par ce dernier pour conduire ne sont pas forcement accessibles au robot. L'apprentissage par renforcement est également beaucoup plus plausible d'un point de vue biologique que l'apprentissage supervisé, puisqu'il ne nécessite pas l'intervention d'un hypothétique professeur.
Les exemples de travaux de ce type sont très nombreux. La plupart d'entre eux concernent de petits robots évoluant en environnement intérieur (i.e. laboratoire) et des comportements simples tels que le suivi de mur ou l'évitement d'obstacle. Dans ce dernier cas la fonction de renforcement utilisée peut dépendre simplement de la distance entre le robot et les obstacles, ou encore d'une éventuelle collision. Un bon aperçu de ces réalisations peut être trouvé dans les travaux de L. J. Lin [95], J. R. Millan et C. Torras [102] ou C. Touzet [160]. Les RNA sélectionnés sont généralement de type réseaux multi-couches, mais d'autres modèles sont également utilisés. M. Ortiz et P. J. Zufiria [121] comparent par exemple les performances obtenues avec des réseaux de type RBF ou CMAC. D. F. Hougen et al. [65] utilisent un modèle de réseau très simple appelé ROLNNET permettant un apprentissage rapide sur le problème du parking d'un camion avec remorque.
Les travaux de R. Sun autour des architectures CONSYDERR et CLARION [154, 152] sont plus originaux puisque l'auteur rajoute à un niveau réactif neuronal, entraîné par renforcement, un algorithme capable d'extraire des règles symboliques à partir du réseau. L'auteur avance que la connaissance représentée par ces règles est de plus haut

niveau et plus facile à généraliser que celle acquise par le réseau.

L'ensemble de ces travaux présente cependant un inconvénient majeur lié au temps très important nécessaire pour l'apprentissage. Ceci peut être imputé à deux raisons principales :

- la pauvreté de l'information disponible durant l'apprentissage. Le robot sait si les actions qu'il a effectuées ont été bénéfiques ou non, mais dans le deuxième cas il ignore quelle action aurait été meilleure ;

- le conflit existant lors de l'apprentissage entre exploration et sélection de la commande optimale (ou exploitation). L'exploration a pour but de permettre au robot de «tester» l'ensemble des commandes possibles dans une situation donnée afin d'en découvrir l'effet. Cela implique parfois le choix d'actions amenant un renforcement négatif et est donc souvent contradictoire avec le but recherché par le robot (e.g. évitement d'obstacle, ...). La recherche exclusive de ce but amènerait le robot à sélectionner à tout moment les actions donnant le renforcement attendu maximum et donc à passer à coté de solutions non encore explorées mais donnant des résultats supérieurs.

Plusieurs auteurs ont tenté d'apporter des solutions à ce problème de lenteur de l'apprentissage. S. Thrun [157, 158] passe en revue différentes méthodes pour accélérer l'exploration. Il propose notamment de maintenir une carte de compétence indiquant les zones de l'espace (états $\times$ actions) pour lesquelles un apprentissage a déjà été effectué.

R. S. Sutton propose dans son système Dyna [156] de maintenir un modèle donnant à partir d'un état et d'une action donnée le nouvel état et le renforcement attendu. Ce modèle est construit en utilisant un deuxième RNA entraîné en même temps que celui fournissant la loi de commande. Il permet de simuler des suites d'actions virtuelles donnant des renforcements virtuels, utilisés comme si ces actions étaient réellement effectuées, pour accélérer l'apprentissage.

Si ces approches sont très intéressantes dans de nombreux cas étant donné le peu d'informations extérieures nécessaires pour l'apprentissage, elles restent à notre avis et dans l'état actuel des connaissances, limitées à de petits robots évoluant à faible vitesse dans des environnements simples. S'il est aisé de faire évoluer pendant plusieurs heures un robot de 20 cm de haut dans une pièce de quelques mètres carrés pour lui permettre de réaliser un apprentissage, il semble beaucoup moins réaliste d'envisager la même expérience avec un véhicule de la taille du nôtre. Les besoins d'exploration liés à l'apprentissage par renforcement risquent également de poser des problèmes de sécurité si la vitesse d'évolution du robot est importante.

Ces différentes raisons rendent à notre avis l'apprentissage par renforcement assez peu adapté à notre application.

4.2.3.2 Utilisation d'un apprentissage supervisé

L'utilisation d'un apprentissage supervisé pour l'entraînement du robot peut permettre à celui-ci de reproduire le comportement d'un autre système de commande ou encore, ce qui est plus intéressant, d'un pilote humain, en se basant sur les données perceptives disponibles.
C'est cette dernière solution qui est à la base des Travaux de D. A. Pomerleau [131] autour du système ALVINN[1]. Le but recherché est la commande en braquage d'un véhicule de type voiture lors de la conduite sur route, et ce afin de suivre une voie de circulation. L'auteur utilise un réseau multi-couches dont l'entrée est composé de (30×32) neurones se comportant comme une rétine et recevant une image pré-traitée provenant d'une caméra. Le prétraitement réalisé consiste à mettre en évidence sur ces images les bordures des voies de circulation. La sortie du réseau se compose de 30 neurones correspondant chacun à une position angulaire des roues du véhicule (voir figure 4.1).

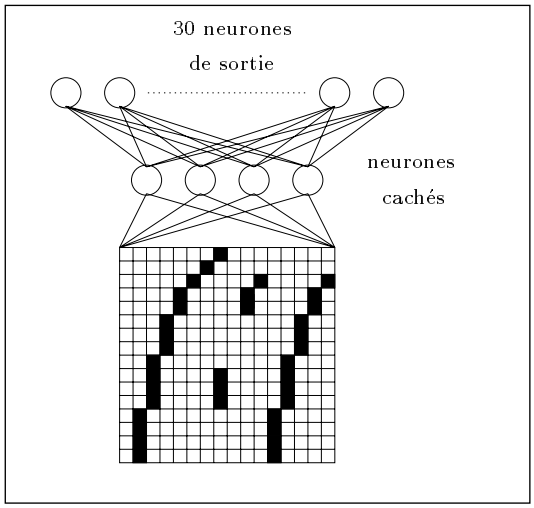

FIG. 4.1 – *Le système ALVINN de Pomerleau [131].*

Le réseau est entraîné à partir de l'observation d'un pilote humain et apprend à associer la direction que celui-ci choisit, aux images de la caméra. Quelques minutes d'apprentissage suffisent à donner une véritable autonomie au système. Pour éviter une trop grande spécialisation du réseau si les exemples disponibles ne sont pas suffisamment variés (e.g. si la route reste droite), l'auteur propose de conserver dans une mémoire un ensemble de cas représentatifs de toutes les situations rencontrées et de les ré-utiliser à chaque pas d'apprentissage. De nombreuses extensions à ce travail ont été proposées. Parmi celles-ci nous pouvons citer le changement de voies [72] ou l'intégration de plusieurs réseaux entraînés sur des types de route différents [73]. Dans [132] les auteurs proposent également d'utiliser plusieurs réseaux spécialisés, choisis à partir d'informations symboliques issues d'une carte annotée de l'environnement.

1. Acronyme de Autonomous Land Vehicle In a Neural Network.

Il est intéressant de noter à propos de ce dernier exemple que *la notion de système hybride neuro-symbolique en intelligence artificielle y rejoint celle de système hybride en robotique.*
D. L. Marruedo et J. G. Ortega [98] proposent un système de commande basé sur l'utilisation d'un capteur proximétrique laser, balayant la zone située à l'avant du robot. L'entraînement se fait en simulation, à partir d'un modèle du robot et de ses interactions avec l'environnement. Ceci permet d'obtenir une très grande quantité d'exemples pour l'apprentissage. Nous pouvons noter que le capteur proximétrique utilisé fournit des résultats bien plus fiables que les capteurs ultrasonores équipant notre robot et que comme nous le verrons plus loin une simulation réaliste de ces derniers est très difficile. Une telle approche n'est donc pas envisageable dans notre cas.
Les travaux correspondant à l'imitation d'un pilote humain à partir de données aussi pauvres que celles délivrées par des capteurs ultrasonores sont relativement rares. Un exemple simple pourra être trouvé dans [137] mais le robot y reste simulé et les capteurs y sont considérés comme parfaits (i.e. ne souffrant pas des problèmes présentés en début de ce chapitre).
La raison principale du faible nombre de réalisations suivant cette approche qui semble pourtant simple et intuitive est la difficulté rencontrée pour obtenir des exemples suffisamment nombreux et variés pour garantir une généralisation correcte du réseau. Ceci est d'autant plus vrai pour un véhicule de la taille du notre, pour lequel toute expérience demande une préparation relativement lourde.

4.2.3.3 Autres approches

En plus des deux grandes familles que nous venons de présenter, il existe de nombreuses autres approches pour la commande réactive de robots mobiles à partir de RNA [107]. La plupart d'entre elles suivent un des schémas présentés dans le chapitre 2. Nous pouvons en citer deux parmi les plus intéressantes pour le type d'application que nous visons.

- J. G. Ortega et E. F. Camacho [120], tout d'abord, proposent d'utiliser une approche basée sur la commande par modèle prédictif (voir §2.5.2.3). Ce travail que nous avons déjà abordé à propos du suivi de chemin (§3.2.3), devrait permettre à travers la spécification de la fonction de coût utilisée de contrôler très précisément les mouvements attendus pour le robot. L'utilisation d'un modèle du véhicule et surtout de ses capteurs, que celui-ci soit réalisé ou non par un RNA, inhérente à cette approche, est cependant comme nous l'avons déjà dit peut réaliste dans le cas de capteurs ultrasonores ;

- R. Biewald emploie pour sa part une approche de commande indirecte proche de celle proposée par D. Nguyen et B. Widrow [113] et présentée dans la section 2.5.2.2. L'auteur utilise trois RNA lors de l'apprentissage, respectivement pour modéliser la cinématique du véhicule, pour modéliser les données fournies par les capteurs en fonction de la position du robot et des obstacles et pour apprendre la

loi de commande. Le but recherché est la réalisation de manœuvres comme le suivi de mur ou le passage de bifurcations dans un couloir. Les deux réseaux chargés de la modélisation sont entraînés séparément sur simulateur. Deux techniques différentes sont proposées pour l'entraînement du troisième :

- utiliser l'erreur en position. Dans ce cas il est nécessaire de spécifier la trajectoire idéale à suivre pour le robot lors de la manœuvre. La distance entre le véhicule et les points de cette trajectoire est rétropropagée à travers le réseau modélisant la cinématique du robot, jusqu'au réseau chargé de l'apprentissage de la loi de commande ;
- utiliser une erreur sur la mesure des capteurs. Dans ce type d'apprentissage l'auteur défini une valeur minimum et une valeur maximum pour chaque capteur du véhicule, et une fonction d'erreur dépendant des dépassements de ces valeurs. Cette erreur est rétropropagée à travers le réseau modélisant le comportement des capteurs, puis à travers celui modélisant les déplacements du véhicule. Comme dans le cas précédent il est alors possible de réaliser l'apprentissage de la loi de commande par le troisième réseau à partir de l'erreur ainsi rétropropagée.

La principale limitation de cette approche concerne encore une fois la modélisation de capteurs réels. Le travail présenté ne dépasse pas en effet le cadre d'un simulateur dans lequel les capteurs sont considérés comme parfaits.

4.2.4 Autres travaux

Il est bien sûr impossible de citer l'ensemble des approches ayant été utilisées pour aborder la commande réactive de robots mobiles.
L'approche proposée par P. Bessiere par exemple, est fortement originale puisqu'elle utilise une modélisation probabiliste de la relation entre les données perceptives et les actions du robot [16]. Ces dernières dépendent en effet de lois de probabilités qui peuvent être obtenues grâce à une méthode d'apprentissage supervisé.

Le problème de l'évitement d'obstacle a également été traité par R. Zapata [172] par la méthode des zones virtuelles déformables (ZVD). Le robot est entouré d'une zone dont la taille et la forme dépendent de la vitesse du véhicule et qui se déforme lorsqu'un obstacle y fait intrusion. La loi de commande utilisée a pour but de générer des mouvements permettant à cette zone de reprendre sa forme originale. Cette approche a permis de faire évoluer des robots jusqu'à une vitesse de 7 $m.s^{-1}$ en environnement dynamique. Nous pouvons noter que cette méthode de commande a été implémentée par P. Déplanque etal. [32] à l'aide d'un RNA entraîné à fournir des consignes minimisant la déformations des ZVD.

C. Novales [116] propose pour sa part une méthode baptisée lignes de fuite. Dans cette technique le système de commande propose à chaque pas de temps un très grand

nombre de trajectoires (les lignes de fuites), correspondant à l'ensemble des lois de commande applicables durant un intervalle de temps fixe à partir de l'état courant du véhicule. La trajectoire permettant de se rapprocher le plus possible de la position désirée (ou du chemin de référence), tout en garantissant un mouvement sans collision est retenue. Pour permettre une exécution en temps réel, l'ensemble des lignes de fuites est pré-calculé et stocké en mémoire.

Enfin, nous pouvons signaler que des manœuvres plus spécifiques comme par exemple le suivi d'une file de véhicules [31] ou le parking en créneau [123] ont également été traitées par des méthodes issues de l'automatique.

4.3 Présentation de notre approche

4.3.1 Principe et motivations

Dans notre travail nous nous sommes intéressés aux possibilités offertes par les RNA, de spécifier précisément par apprentissage le mouvement attendu pour le robot. L'imitation d'un pilote humain peut en effet, comme nous l'avons vu, offrir un moyen très simple, de réaliser la transformation perception $\rightarrow$ action que nous cherchons à obtenir. La motivation de notre approche provient de deux constations :

- il est impossible de simuler de manière fidèle le comportement d'un capteur de type ultrasonore. Ceci impliquerait en effet la prise en compte d'un nombre beaucoup trop important de facteurs (voir annexe B).

  L'apprentissage doit donc être réalisé directement à partir de données issues du véhicule réel. L'utilisation d'un simulateur peut permettre de tester la faisabilité d'une approche mais pas de réaliser le système de commande final ;

- l'utilisation d'un véhicule expérimental de grande taille, demande la mobilisation d'une infrastructure et de moyens qui rendent difficiles la multiplication des expériences de collecte de données.

Notre but a donc été la réalisation d'un système capable de se contenter d'un nombre réduit d'exemples lors de la phase d'apprentissage.
Comme nous l'avons vu dans la section 2.1.3, les modèles de RNA les plus utilisés demandent généralement un nombre d'exemples relativement important pour obtenir un apprentissage et une capacité de généralisation satisfaisants sur un problème complexe. Nous avons également évoqué la possibilité de pallier cette limitation en incorporant la connaissance *a priori*, disponible sur la tâche à accomplir, dans le réseau[1].
La section 2.2.2 de ce mémoire présente une approche permettant d'incorporer dans un RNA les informations disponibles sous forme de règles floues. Puisque des règles de

1. Nous pouvons noter que la solution consistant à combiner la connaissance *a priori* disponible avec un apprentissage supervisé est également à la base des travaux de P. Bessiere, dans notre laboratoire, autour des systèmes d'inférence probabiliste [15].

ce type sont assez faciles à obtenir pour la commande d'un véhicule mobile, c'est cette méthode que nous avons retenue.
La phase d'apprentissage supervisé du réseau devient donc, dans notre approche, un simple moyen d'affiner le comportement du véhicule provoqué par les règles floues. Ceci doit également permettre de résoudre les limitations évoquées à propos de l'utilisation d'un contrôleur flou classique (voir §4.2.2).

4.3.2 Réalisation pratique

4.3.2.1 Carte locale de l'environnement

Comme nous l'avons déjà évoqué, l'application d'un système d'inférence flou (ou neuro-flou) à la commande d'un véhicule suit une démarche assez intuitive. La solution la plus simple consiste à utiliser les valeurs retournées par les différents capteurs du véhicule comme variables d'entrée. Il est possible de définir pour ces variables des sous-ensembles flous correspondant à des distances. Les prémisses des règles utilisées ont dans ce cas la forme suivante :

$$Si\ DISTANCE\_CAPTEUR\_DEVANT\ est\ GRANDE\ alors\ ...$$

$$Si\ DISTANCE\_CAPTEUR\_GAUCHE\ est\ PETITE\ alors\ ...$$

Les sorties des règles peuvent être directement les consignes appliquées au robot.
Cette approche pose cependant un problème important lorsqu'elle est appliquée à un véhicule utilisant des capteurs de type ultrasonore. Comme nous l'avons indiqué au début de ce chapitre la fiabilité des résultats fournis par ces capteurs dépend d'un grand nombre de facteurs. Un obstacle peut être détecté à un instant donné puis ne plus l'être pendant la période suivante si son orientation par rapport au véhicule a changé. La figure 4.2 montre par exemple un type de comportement observé sur le véhicule réel lors du dépassement d'un obstacle. Dans la partie gauche (a) le véhicule détecte l'obstacle et amorce un dépassement, dans la partie droite (b) l'obstacle n'est plus détecté et le véhicule se rabat. Ceci mène à des oscillations lorsque le robot évolue à faible vitesse et peut causer des problèmes plus graves (i.e. échec du dépassement ou même collision) lorsque la vitesse augmente.
La solution que nous proposons consiste à conserver une mémoire des différents échos capteurs obtenus par le véhicule dans une carte locale[1] très simple de l'environnement. Lorsqu'un capteur renvoie une donnée relative à la position d'un obstacle, le système stocke cette information sous la forme d'un segment de droite correspondant à la zone dans laquelle peut se trouver cet obstacle (voir figure 4.3).
Les positions relatives des segments par rapport au véhicule sont modifiées en fonction des déplacements de ce dernier. Les transformations géométriques nécessaires à cette mise à jour sont relativement simples. Elles ont été décrites notamment par J. Crowley [28].

1. Carte de l'environnement proche du robot.

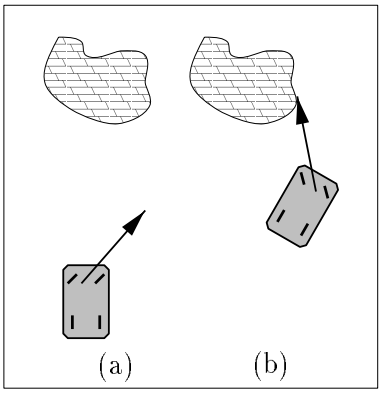

FIG. 4.2 – *Comportement du véhicule en cas d'utilisation directe des données capteurs. La flèche montre la direction suivie. Le commentaire est donné dans le texte.*

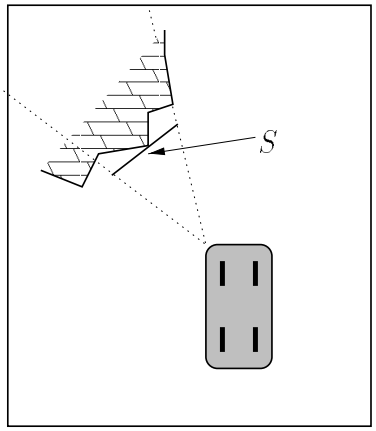

FIG. 4.3 – *Modélisation d'un écho capteur. Le cône de réception du capteur est noté en pointillés. Le segment de droite noté S correspond à l'information conservée en mémoire.*

Lors de la réception d'un nouvel écho capteur, le système tente dans un premier temps de le fusionner avec un des échos stockés en mémoire. La fusion consiste à remplacer l'écho plus ancien par le plus récent. Cette opération réussit si l'un des échos se trouve suffisamment proche de celui qui vient d'être détecté. La distance limite à partir de laquelle la fusion est réalisée dépend de la proximité du véhicule (i.e. deux échos seront plus facilement fusionnés s'ils se trouvent loin du robot).
Cette étape permet d'une part d'éviter la multiplication des informations stockées en mémoire qui rendrait le temps de calcul trop important et d'autre part d'améliorer la précision. Lorsqu'un obstacle est détecté à une grande distance, l'incertitude en position qui lui est associée (i.e. la taille du segment) est importante. Cette incertitude peut être réduite si le même obstacle est détecté de nouveau à une distance plus faible et que le nouvel écho est fusionné avec le précédent.
Chaque information stockée en mémoire se voit également attribuer une durée de vie

qui dépend du nombre de fois où un obstacle a été détecté à cette position. Les segments sont supprimés de la mémoire lorsque leur durée de vie arrive à son terme. Ceci permet encore une fois de réduire le nombre d'échos stockés en mémoire, mais aussi et surtout de traiter les obstacles en mouvement. Ceux-ci laisseraient en effet sur la carte, si tous les échos étaient conservés éternellement, une suite de segments correspondant à leurs positions successives. Cette durée de vie des obstacles est similaire au facteur de confiance proposé par J. Crowley[29].
Durant les expériences présentées dans ce chapitre, nous avons utilisé des constantes telles que la durée de vie d'un obstacle est incrémentée de 2 secondes chaque fois que celui-ci est détecté, mais ne dépasse jamais une valeur maximum de 10 secondes.
La carte de l'environnement local ainsi construite ne nécessite qu'un faible temps de calcul pour sa mise à jour mais reste très sommaire. Le lecteur intéressé par une étude plus poussée des possibilités de modélisation de l'environnement local à l'aide de capteurs ultrasonores pourra se reporter par exemple aux travaux de O. Patrouix [124], de B. Schiele et J. Crowley [146] ou encore de R. Zapata[171] qui propose une approche basée sur la notion de mémoire dynamique développée par J. Droulez [33].

4.3.2.2 Notion de capteur virtuel

Afin de permettre l'utilisation par le système neuro-flou des informations conservées sur la carte locale, nous avons défini un certain nombre de zones autour du robot correspondant en quelque sorte à des capteurs virtuels [1]. Chacune de ces zones possède, soit une forme similaire à celle du cône de réception d'un capteur réel, soit une forme rectangulaire.
Elles sont utilisées comme s'il s'agissait des capteurs réels du robot et concordent donc avec les variables d'entrée du système neuro-flou.
La première approche que nous avons suivie consiste à affecter à chaque capteur virtuel (et donc à chaque entrée du réseau) une valeur correspondant à la distance entre l'origine de ce capteur et le segment le plus proche se trouvant dans la zone qui lui est associée (voir figure 4.5).
La figure 4.4 montre la position et l'orientation des 14 capteurs réels implantés sur notre véhicule expérimental. Afin de perdre un minimum de précision dans les informations, nous avons défini un nombre à peu près équivalent de capteurs virtuels. La figure 4.5 présente leur situation autour du véhicule.
L'utilisation d'une carte locale et de capteurs virtuels présente deux avantages supplémentaires :

- il est possible de fusionner sur la carte les informations provenant de plusieurs capteurs de type différents, comme par exemple une caméra et des capteurs ultrasonores ;

- il est possible de faire figurer sur la carte des données sur l'environnement, connues

1. Ces capteurs virtuels sont assez proches des zones d'intérêt définies par Garnier [41].

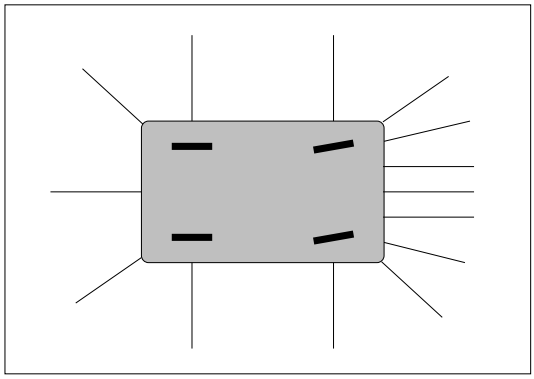

FIG. 4.4 – *Position des capteurs sur le véhicule expérimental.*

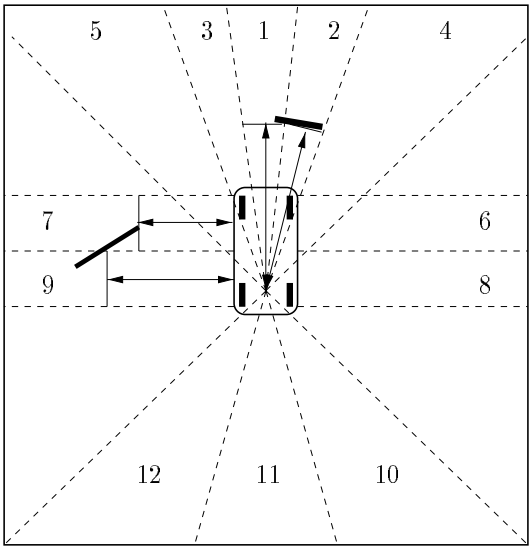

FIG. 4.5 – *Situation et numérotation des capteurs virtuels autour du véhicule. Les lignes en pointillés montrent les limites des zones. Les échos mémorisés sont dessinés en gras. La valeur de distance associée aux capteurs virtuels est indiquée par les flèches.*

a priori et non détectées ou non détectable par le robot. Ces informations peuvent être par exemple la position des limites des voies de circulation.

4.3.2.3 Définition de sous-ensembles flous à deux dimensions d'entrée

Les tests réalisés avec l'approche décrite ci-dessus ont fait apparaître un problème gênant.
Il est souhaitable, afin d'éviter des changements de consignes trop brusques, qu'un faible mouvement du véhicule ne provoque qu'un faible changement des entrées du système (sauf si un nouvel obstacle est brusquement détecté).
De fait le mouvement d'un écho à l'intérieur d'une zone provoque bien une variation

graduelle de l'information de distance associée au capteur virtuel correspondant. La sortie ou l'entrée d'un écho dans la zone de ce capteur, s'accompagne par contre d'une variation brutale de la distance qui est associée à celui-ci (voir figure 4.6).
Ceci provoque des discontinuités importantes dans les consignes de sortie du système et donc des oscillations dans le comportement du véhicule.

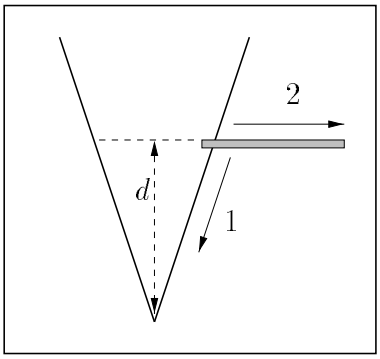

FIG. 4.6 – *Variation brutale de l'information associée à un capteur virtuel. La distance d associée au capteur virtuel varie progressivement si le segment (représenté en gris) se déplace dans la direction 1. Elle change brusquement si le déplacement suit la direction 2.*

Une première solution à ce problème consisterait à multiplier le nombre de capteurs virtuels pour permettre un échantillonnage plus fin de l'espace d'entrée. Ceci s'accompagnerait cependant d'une multiplication du nombre de règles, et donc du nombre de paramètres libres du système de commande, qui n'est pas souhaitable.
Nous nous sommes donc tournés vers une solution consistant à exploiter pour chaque capteur virtuel, en plus de l'information de distance entre l'obstacle et l'origine du capteur (distance longitudinale), une information de distance entre l'obstacle et l'axe central de ce capteur (distance latérale).
Cette deuxième distance est soit une valeur angulaire dans le cas d'un capteur de forme conique, soit une longueur dans le cas d'un capteur de forme rectangulaire (voir figure 4.7).
L'exploitation de ces deux valeurs pour chaque capteur nous amène à définir des sous-ensembles flous possédant deux dimensions d'entrée. La première de ces dimensions correspond à la distance longitudinale et la deuxième à la distance latérale (voir figure 4.8).
Ces sous-ensembles flous sont similaires aux champs récepteurs définis par les unités d'un RNA de type RBF.
Nous en associons 4 portant les labels Proche, Moyen, Loin et Libre, à chaque variable d'entrée du système neuro-flou (i.e. à chaque capteur virtuel). La figure 4.9 montre leur positionnement dans le cas d'un capteur de type rectangulaire.
Nous conservons la position des capteurs virtuels présentée sur la figure 4.5, mais leur surface est doublée de manière à obtenir un recouvrement important des zones

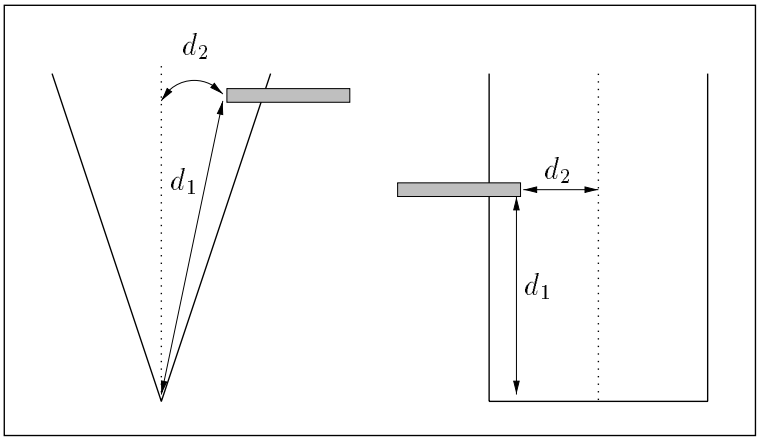

FIG. 4.7 – *Les 2 informations de distance associées à un capteur virtuel de forme conique ou rectangulaire.*

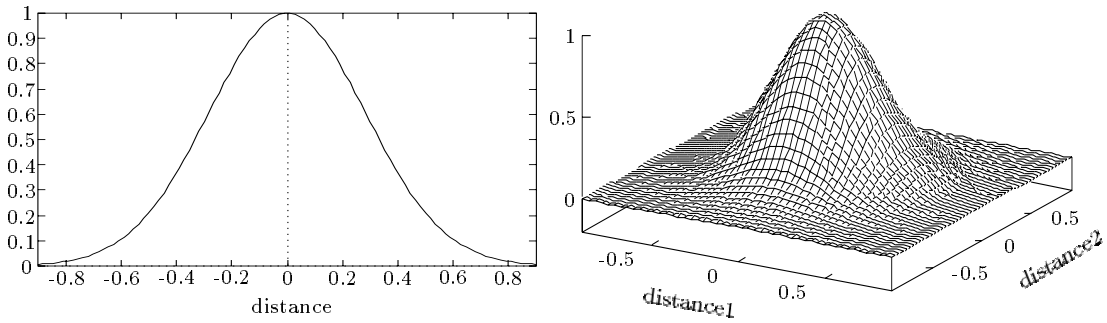

FIG. 4.8 – *Sous-ensemble flou gaussien à une ou deux dimensions.*

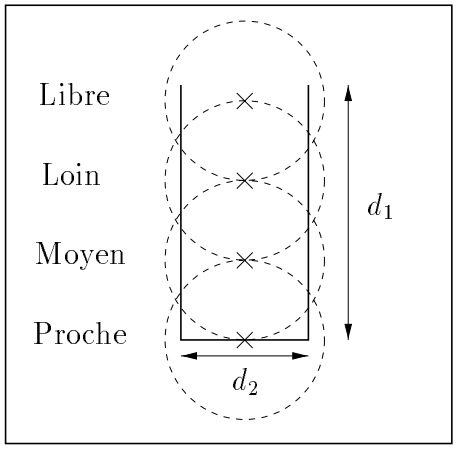

FIG. 4.9 – *Positionnement des sous-ensembles flous associés à un capteur virtuel rectangulaire. Les pointillés correspondent aux limites des zones de l'espace d'entrée en dehors desquelles l'activation des sous-ensembles flous devient négligeable.*

associées. L'information de distance latérale est ramenée à l'intervalle [-1;1] afin de pouvoir utiliser la même définition pour des sous-ensembles flous associés à des capteurs virtuels de formes et de tailles différentes.
Lorsque plusieurs échos sont présents à l'intérieur d'un capteur virtuel, c'est celui pour lequel la somme des deux distances est la plus faible qui est retenu.
Lors des expériences réalisées, nous n'avons pas étendu l'apprentissage aux paramètres des sous-ensembles flous liés à la distance latérale (centre et variance). Ceci permettrait de régler par apprentissage la forme et la position des capteurs virtuels mais compliquerait énormément le choix de l'écho capteur à utiliser pour calculer les distances.
Nous pouvons noter que plutôt que d'introduire cette notion de sous-ensemble flou à deux dimensions, il aurait été possible d'utiliser une deuxième variable d'entrée distincte pour chaque capteur virtuel (la distance latérale) sur laquelle un seul sous-ensemble flou serait défini. En affectant le label MOYEN à ce sous-ensemble, la proposition floue :
DISTANCE_LONGITUDINALE est PROCHE et DISTANCE_LATERALE est MOYEN
donnerait un résultat équivalent [1] à ce que nous obtenons par la proposition :

DISTANCE est PROCHE,

où PROCHE est un sous-ensemble à deux dimensions. Ceci rendrait cependant l'expression et la compréhension des règles beaucoup plus complexes.

4.3.2.4 Autres variables d'entrée/sortie du système

Afin de permettre au véhicule de s'aligner par exemple sur un mur, nous définissons deux variables d'entrée supplémentaires dont les valeurs sont calculées respectivement à partir des capteurs virtuels 6 et 8 ou 7 et 9 (voir figure 4.5). Ces deux entrées correspondent à l'orientation en radians du véhicule par rapport aux obstacles situés sur sa gauche ou sur sa droite. Elles ne sont disponibles que lorsque les deux zones impliquées dans le calcul (7 et 9 ou 6 et 8) contiennent toutes les deux des échos. Dans le cas contraire les règles floues associées à ces entrées ne sont pas activées. Un résultat équivalent aurait pu être obtenu par des règles du type :

Si CAPTEUR_VIRTUEL_6 est PROCHE et CAPTEUR_VIRTUEL_8 est LOIN ....

Le prétraitement réalisé sert uniquement à réduire le nombre de règles nécessaires.
Enfin, une dernière variable d'entrée est utilisée pour indiquer le cap en radians du but du véhicule. Ce but peut être par exemple un point d'une trajectoire de référence dont le véhicule s'est éloigné durant sa manœuvre.
Ces trois variables (orientations latérales et direction du but) utilisent des sous-ensembles flous classiques à une dimension d'entrée.
Les sorties utilisées sont directement les consignes appliquées au robot. Comme dans le chapitre précédent nous utilisons l'angle de braquage et la vitesse pour commander le véhicule. La consigne de braquage est ramenée à l'intervalle [-1;1].

1. Le résultat serait exactement le même à condition d'utiliser la logique probabiliste (opérateur de conjonction produit).

4.4 Notre modèle de réseau neuro-flou

4.4.1 Présentation du réseau

Nous avons opté pour l'utilisation d'un réseau multi-couches et de règles de type Sugeno d'ordre 0. Celles-ci sont en effet beaucoup plus simples à utiliser et permettent dans notre cas d'exprimer l'ensemble des concepts que nous désirons introduire dans le système de commande. Notre réseau présente des similarités avec le système ANFIS proposé par J-S. R. Jang [70]. Nous proposons toutefois plusieurs extensions à ce modèle :

- les sous-ensembles flous utilisés dans les prémisses peuvent être définis par des fonctions d'appartenance gaussiennes asymétriques (voir plus bas) à 1 ou 2 dimensions d'entrée ;
- les règles utilisées acceptent à la fois des opérateurs de conjonction, de disjonction et de négation ;
- chaque règle se voit affecter un poids caractérisant son importance par rapport aux autres règles. L'utilisation de ce paramètre est très importante dans notre application puisque nos règles codent plusieurs comportements du robot ayant des priorités différentes. Un comportement d'évitement d'obstacle doit par exemple être prioritaire sur un comportement d'attraction par un but.

Nous choisissons d'utiliser la logique de Zadeh. Les opérateurs de la logique probabilistes sont en effet plus simples à utiliser dans le cadre d'une descente de gradient, mais sont aussi plus restrictifs [45]. Le degré de vérité des règles est en effet généralement inversement proportionnel au nombre de propositions floues élémentaires composant leurs prémisses, lorsque l'opérateur produit est utilisé. Ceci peut donc poser un problème de cohérence lorsque toutes les règles n'ont pas des prémisses de même taille.
Nous présentons ci-dessous les différents choix réalisés dans la conception du réseau (voir figures 4.13 et 4.14).

Entrée Les unités de la couche d'entrée sont soit des neurones recevant les données extérieures (les entrées du système flou), soit des neurones biais dont l'activation reste fixée à un.

Sous-ensembles flous Les unités de la première couche cachée concordent chacune avec un sous-ensemble flou. Leurs fonctions d'activation correspondent aux fonctions d'appartenance de ces ensembles.
Plusieurs formes sont autorisées pour la définition de ces fonctions :

- dans le cas de sous-ensembles flous à une dimension d'entrée nous utilisons deux fonctions gaussiennes possédant le même centre mais deux variances différentes pour calculer la fonction d'appartenance. La première fonction est utilisée lorsque

la valeur d'entrée est inférieure à la moyenne, et la deuxième dans le cas contraire. Ceci permet de définir une fonction d'appartenance asymétrique (voir figure 4.10), et d'être donc moins restrictif sur la classe des systèmes flous que nous pouvons représenter. Les deux gaussiennes prenant la valeur 1 et une dérivée nulle pour une valeur d'entrée égale à la moyenne, la fonction obtenue reste continue et dérivable, et peut donc être utilisée dans le cadre d'une optimisation par descente de gradient.

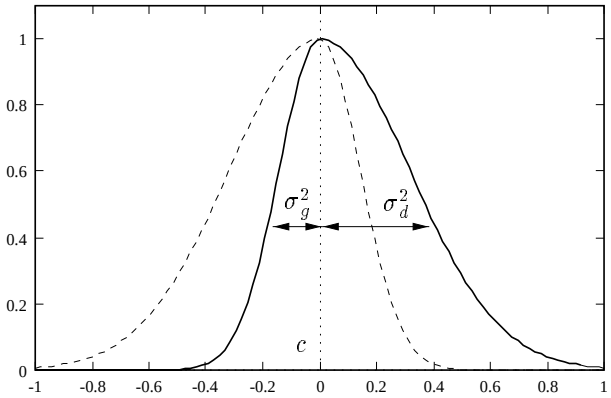

FIG. 4.10 – *Fonction d'appartenance asymétrique (en ligne continue). La moyenne est notée c et les variances σ_g^2 et σ_d^2.*

Chacun des neurones de ce type, que nous appellerons neurones gaussiens, reçoit quatre connexions. La première provient d'un neurone d'entrée et fournit la valeur de la variable concernée par le sous-ensemble flou codé.

Les trois autres proviennent de neurones biais et se voient attribuer des poids correspondant aux paramètres (variances et moyenne) de la fonction d'appartenance. Cette représentation des paramètres libres par des poids permet de conserver le formalisme classique des algorithmes d'apprentissage neuronaux (voir figure 4.11).

L'activation a_i d'un neurone d'indice i de cette couche est donc donnée par l'équation :

$$a_i = \begin{cases} \exp(-\frac{(x-c)^2}{\sigma_g^2}) & \text{si } x < c \\ \exp(-\frac{(x-c)^2}{\sigma_d^2}) & \text{sinon} \end{cases} \tag{4.1}$$

Dans cette équation c, σ_d^2 et σ_g^2 sont respectivement la moyenne, la variance droite et la variance gauche des gaussiennes, et x est l'entrée correspondant à la variable d'entrée du système flou ;

– un deuxième type de sous-ensemble flou peut être obtenu à l'aide de la famille de fonctions suivante :

$$a_i = \begin{cases} 1 & \text{si } x < c \\ \exp(-\frac{(x-c)^2}{\sigma^2}) & \text{sinon} \end{cases} \quad \text{ou } a_i = \begin{cases} \exp(-\frac{(x-c)^2}{\sigma^2}) & \text{si } x < c \\ 1 & \text{sinon} \end{cases} \tag{4.2}$$

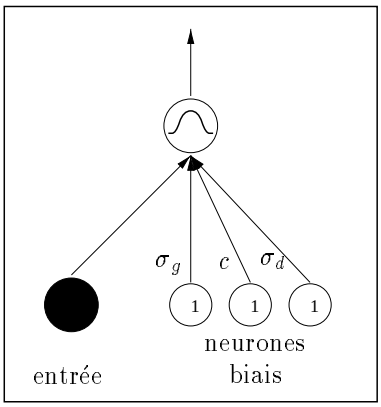

FIG. 4.11 – *Codage des paramètres associés à une fonction d'appartenance.*

La figure 4.12 montre un exemple de fonction de ce type. Nous parlerons de neurones semi-gaussiens pour décrire les unités utilisant cette fonction d'activation.

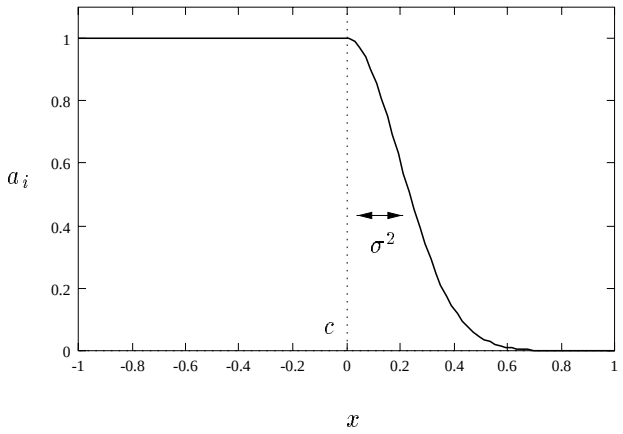

FIG. 4.12 – *Fonction d'appartenance monotone.*

– l'activation des neurones représentant des sous-ensembles flous à deux dimensions d'entrée est également définie par des fonctions de type gaussiennes :

$$a_i = exp\left(-\left(\frac{(x_1 - c_1)^2}{\sigma_1^2} + \frac{(x_2 - c_2)^2}{\sigma_2^2}\right)\right), \tag{4.3}$$

où x_1 et x_2 sont les entrées, c_1 et c_2 les moyennes et σ_1 et σ_2 les variances. Tous comme dans les cas précédents il est possible de définir 2 variances différentes (à gauche et à droite de la moyenne) pour chaque dimension d'entrée. Cela donne un total de 6 paramètres libres (4 variances et deux centres) pour les fonctions d'appartenance de ce type. Toutefois comme nous l'avons dit plus haut nous limitons l'apprentissage aux paramètres associés à la première dimension d'entrée (c_1 et σ_1 dans l'équation 4.3).

La sortie d'un neurone de cette couche codant un sous-ensemble flou A et recevant en entrée la valeur de la variable X, correspond au degré de vérité de la proposition floue X est A.

Négations Les neurones de la deuxième couche cachée sont optionnels. Ils sont utilisés pour représenter d'éventuels opérateurs de négation présents dans les règles. Chaque unité représente donc la négation d'une proposition floue et reçoit une seule connexion provenant du neurone calculant le degré de vérité de cette proposition.
L'activation d'un neurone d'indice i de cette couche est donc donnée par :

$$a_i = 1 - a_j, \tag{4.4}$$

où a_j est l'activation du neurone gaussien j envoyant une connexion vers le neurone i. Nous qualifierons les neurones de ce type de neurones négation.

Calcul du degré de vérité des prémisses La ou les couches suivantes sont utilisées pour calculer le degré de vérité des prémisses de chaque règle. Dans le cas d'une règle n'utilisant qu'un seul type de connecteur (conjonction ou disjonction) un seul neurone que nous appellerons le neurone règle[1] est nécessaire. Celui ci reçoit en entrée les degrés de vérité des différentes propositions floues qui composent la prémisse de la règle, par des connexions provenant de la première ou de la deuxième couche cachée. Le poids d'une connexion provenant d'un neurone biais est également utilisé pour représenter le poids affecté à la règle.
Les opérateurs choisis sont ceux de la logique de Zadeh (minimum et maximum). L'activation d'un neurone règle d'indice i est donc obtenu par une fonction SOFTMAX ou SOFTMIN :

$$a_i = \begin{cases} P\frac{\sum_j (a_j \exp(-Ka_j))}{\sum_j e^{(-Ka_j)}} & \text{dans le cas d'une conjonction (neurone règle ET)} \\ P\frac{\sum_j (a_j \exp(Ka_j))}{\sum_j \exp(Ka_j)} & \text{dans le cas d'une disjonction (neurone règle OU)} \end{cases} \tag{4.5}$$

Dans cette équation j parcourt l'ensemble des neurones représentant les propositions floues présentes dans la prémisse de la règle, a_j correspond à l'activation du neurone j (degré de vérité de la proposition floue correspondante) et P est le poids affecté à la règle (connexion provenant d'un neurone biais).
Les règles plus complexes combinant conjonction et disjonction doivent être décomposées et font intervenir des neurones que nous qualifierons de neurones sous-règles. Chacun d'entre eux correspond à l'utilisation d'un seul opérateur et exploite donc la même fonction d'activation qu'un neurone règle (voir équation 4.5) à la différence près qu'il ne reçoit pas le poids P affecté à la règle. Leur sortie est combinée par d'autres neurones sous-règles ou par un neurone règle.

1. Nous parlerons de neurone règle ET pour une unité codant un opérateur de conjonction et de neurone règle OU pour une unité codant un opérateur de disjonction.

La figure 4.13 montre le résultat obtenu pour la prémisse :

SI (X Est Non Petit Et Y Est Petit) Ou (X Est Moyen Et Y Est Grand)

Ou (X Est Petit Et Y Est Grand) alors ...

Celle ci est décomposée en trois neurones sous-règles ET connectées à un neurone règle OU.

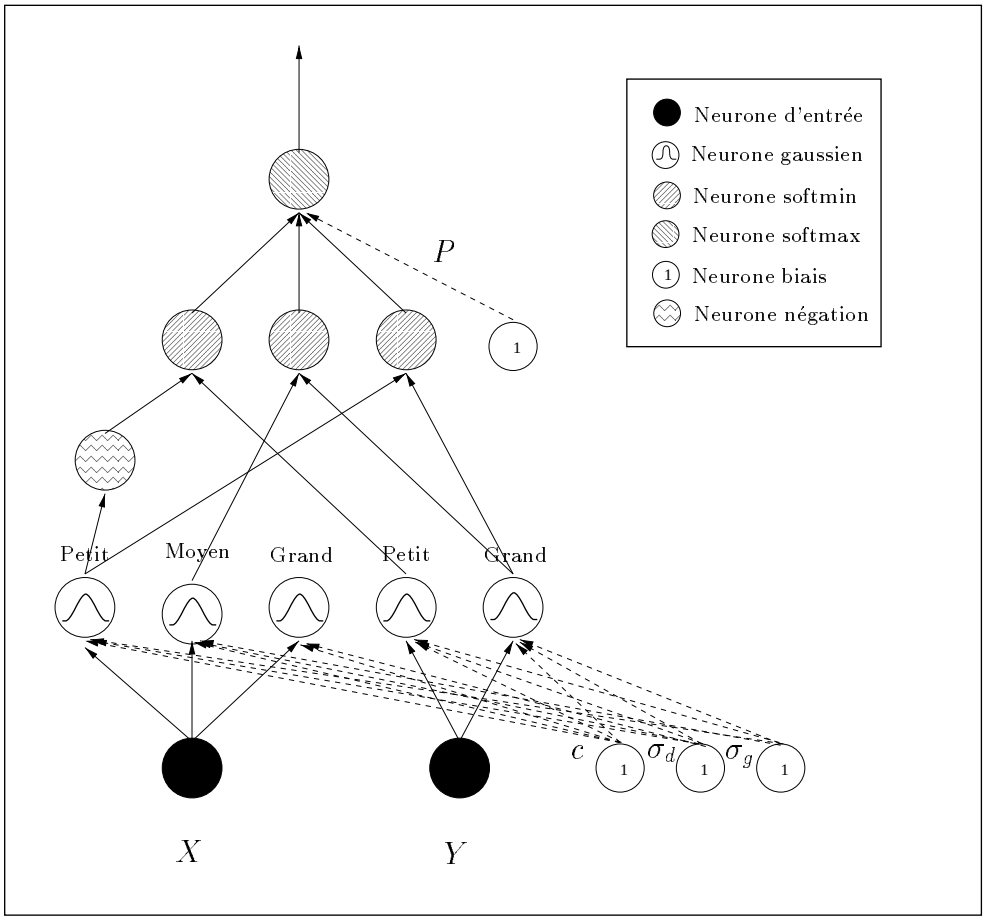

FIG. 4.13 – *Codage de la prémisse d'une règle complexe. Les connexions auxquelles sont associés des poids modifiables sont représentées en pointillés.*

Calcul des valeurs de sortie Les unités de la couche suivante (couche de sortie) concordent avec les sorties du système flou. Elles reçoivent en entrée l'activation de certains neurones règles de la couche précédente (toutes les règles n'agissent pas forcement sur toutes les sorties). Ces activations représentent les degrés de vérité des prémisse des règles correspondantes multipliées par les poids de ces règles. Les connexions entre

les neurones de sortie et les neurones règles se voient affecter des poids correspondant aux valeurs de sortie des règles.
L'activation a_i d'une unité de sortie i correspond à une somme normalisée[1] :

$$a_i = \frac{\sum_j a_j W_{ij}}{\sum_j a_j},$$

où j parcourt l'ensemble des neurones règles agissant sur la sortie i, a_j est l'activation du neurone règle j et W_{ij} est le poids de la connexion entre les unités j et i (i.e. valeur de la sortie i de la règle j).
En d'autres termes on obtient une défuzzification par la méthode du centre de gravité :

$$a_i = \frac{\sum_j P_j o_j \mu_j}{\sum_j P_j \mu_j} \tag{4.6}$$

Dans cette équation j parcourt les règles et P_j, μ_j et o_j correspondent respectivement au poids, au degré de vérité et à la valeur de sortie de la règle j.

Seuls trois types de poids sont modifiables dans notre réseau. Il s'agit des poids représentant les paramètres des fonctions d'appartenance des sous-ensembles flous, de ceux correspondants aux poids affectés aux règles, ainsi que de ceux représentants les sorties de ces règles. Toutes les autres connexions reçoivent un poids fixe de valeur 1.
La figure 4.14 montre un exemple d'architecture obtenue avec ce type de réseau pour un ensemble de 3 règles simples agissant sur une seule sortie.

4.4.2 Adaptation de l'algorithme de rétropropagation du gradient

Nous utilisons pour réaliser l'entraînement de notre réseau l'algorithme de rétropropagation du gradient présenté dans la section 2.1.2.1. Les équations présentées doivent cependant être adaptées aux fonctions de transfert particulières des neurones utilisés. La mesure de performance retenue reste l'erreur quadratique dont l'expression est donnée par l'équation 2.1.
Puisque nous n'utilisons pas de fonction d'entrée pour les neurones de notre réseau, la dérivée de l'erreur Q par rapport au poids W_{ij} de la connexion entre le neurone j et le neurone i peut se décomposer de la manière suivante :

$$\frac{\partial Q}{\partial W_{ij}} = \frac{\partial Q}{\partial a_i} \frac{\partial a_i}{\partial W_{ij}}$$

1. Cette formalisation ne respecte pas le mode opératoire classique des RNA puisqu'un neurone de sortie utilise l'activation d'un neurone règle à la fois directement et en la multipliant par le poids de la connexion entre ces deux neurones. Il est toutefois possible de créer un système exactement équivalent en reliant chaque neurone de sortie à chaque neurone règle par deux connexions, l'une étant pondérée et l'autre non.

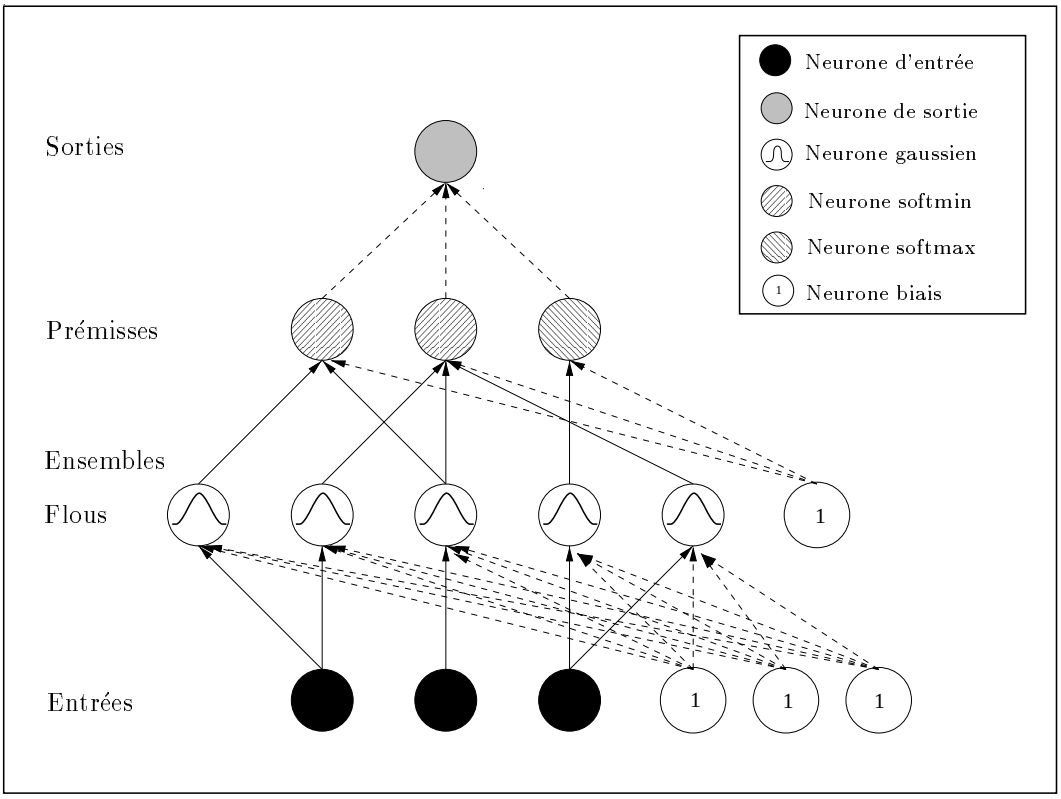

FIG. 4.14 – *Un exemple d'architecture neuro-floue. Les labels situés à gauche de la figure représentent le rôle des neurones utilisés. Les connexions auxquelles sont associés des poids modifiables sont représentées en pointillés.*

En posant $\delta_i = \frac{\partial Q}{\partial a_i}$ nous obtenons :

$$\frac{\partial Q}{\partial W_{ij}} = \delta_i \frac{\partial a_i}{\partial W_{ij}}$$

4.4.2.1 Calcul des contributions à l'erreur

Le calcul des valeurs de δ_i fait intervenir des opérations qui différent selon le type de neurone. Nous allons donc passer en revue les couches du réseau. La notation utilisée est celle de la section 2.1.2.1.

Couche de sortie

$$\delta_i = \frac{\partial Q}{\partial a_i} = a_i - s_i \tag{4.7}$$

Neurones règles Le calcul de δ_i pour les neurones de l'ensemble des couches cachées fait intervenir la décomposition suivante :

$$\delta_i = \frac{\partial Q}{\partial a_i} = \sum_k \frac{\partial Q}{\partial a_k}\frac{\partial a_k}{\partial a_i} = \sum_k \delta_k \frac{\partial a_k}{\partial a_i},$$

où k parcourt les indices des unités vers lequel le neurone i envoie des connexions. Seul le calcul des valeurs de $\frac{\partial a_k}{\partial a_i}$ diffère en fonction de la nature du neurone k.
L'indice k caractérise ici les neurones de la couche de sortie. Nous obtenons donc :

$$\frac{\partial a_k}{\partial a_i} = \frac{W_{ki}\sum_j a_j - \sum_j W_{kj}a_j}{(\sum_j a_j)^2},$$

où j parcourt l'ensemble des neurones règles agissant sur la sortie k etW_{kj} est le poids de la connexion entre les neurones j et k (valeur de la règle j pour la sortie k).

Neurones sous-règles Cette couche n'est pas nécessairement présente dans le réseau. On ne la retrouve qu'en cas de règle mélangeant les opérateurs de conjonction et de disjonction. L'indice k caractérise ici un neurone règle. Si celui-ci correspond à une règle utilisant un opérateur de conjonction on obtient :

$$\frac{\partial a_k}{\partial a_i} = P_k \frac{(e^{-Ka_i} - a_i K e^{-Ka_i})}{\sum_j e^{-Ka_j}} + \frac{(\sum_j a_j e^{-Ka_j})Ke^{-Ka_i}}{(\sum_j e^{-Ka_j})^2} \tag{4.8}$$

Si le neurone k correspond à une règle utilisant un opérateur de disjonction on obtient :

$$\frac{\partial a_k}{\partial a_i} = P_k \frac{(e^{Ka_i} + a_i K e^{Ka_i})}{\sum_j e^{Ka_j}} - \frac{(\sum_j a_j e^{Ka_j})Ke^{Ka_i}}{(\sum_j e^{Ka_j})^2} \tag{4.9}$$

Dans ces deux équations j parcourt les indices de l'ensemble des neurones envoyant des connexions vers le neurone k à l'exception du neurone biais. La valeur transmise par la connexion provenant de ce dernier joue un rôle particulier et est notée P_k (poids de la règle).

Neurones négations L'indice k peut correspondre ici soit à des neurones règles, soit à des neurones sous-règles. S'il caractérise un neurone règle, la valeur de $\frac{\partial a_k}{\partial a_i}$ est donnée par les équations 4.8 et 4.9.
Si k est l'indice d'un neurone sous-règle représentant une conjonction, on obtient :

$$\frac{\partial a_k}{\partial a_i} = \frac{(e^{-Ka_i} - a_i K e^{-Ka_i})}{\sum_j e^{-Ka_j}} + \frac{(\sum_j a_j e^{-Ka_j})Ke^{-Ka_i}}{(\sum_j e^{-Ka_j})^2} \tag{4.10}$$

Si k est l'indice d'un neurone sous-règle représentant une disjonction on obtient :

$$\frac{\partial a_k}{\partial a_i} = \frac{(e^{Ka_i} + a_i K e^{Ka_i})}{\sum_j e^{Ka_j}} - \frac{(\sum_j a_j e^{Ka_j})Ke^{Ka_i}}{(\sum_j e^{Ka_j})^2} \tag{4.11}$$

Dans ces deux équations j parcourt les indices de l'ensemble des neurones envoyant des connexions vers le neurone k.

Neurones gaussiens L'indice k peut représenter ici des neurones règles, sous-règles ou négations.
Dans les deux premiers cas la valeur de $\frac{\partial a_k}{\partial a_i}$ est donnée par les équations 4.8, 4.9, 4.10 et 4.11.

Si k caractérise un neurone négation on obtient par contre:

$$\frac{\partial a_k}{\partial a_i} = -1$$

4.4.2.2 Modification des poids

L'équation 2.2 donnant les modifications des poids du réseau doit également être légèrement modifiée :

$$\Delta W_{ij}(t) = -\eta \delta_i \frac{\partial a_i}{\partial W_{ij}} + \mu \Delta W_{ij}(t-1) \tag{4.12}$$

Encore une fois le calcul de $\frac{\partial a_i}{\partial W_{ij}}$ dépend de la nature des neurones i et j.

Valeur de sortie des règles L'unité j est ici un neurone règle, alors que i caractérise un neurone de sortie. W_{ij} représente donc la valeur de sortie i de la règle j.

$$\frac{\partial a_i}{\partial W_{ij}} = \frac{a_j}{\sum_k a_k}, \tag{4.13}$$

où k parcourt tous les neurones règles qui envoient des connexions vers cette sortie.

Poids des règles Les indices i et j caractérisent respectivement ici un neurone règle et un neurone biais. W_{ij} est donc le poids affecté à la règle représentée par le neurone i.

$$\frac{\partial a_i}{\partial W_{ij}} = \frac{\sum_k (a_k e^{-K a_k})}{\sum_k e^{-K a_k}}, \tag{4.14}$$

où k parcourt tous les neurones représentant les propositions de la partie prémisse de la règle. Ces derniers peuvent être des neurones sous-règles, des neurones négations ou des neurones gaussiens.

Paramètres des neurones gaussiens à une dimension Les indices i et j caractérisent ici respectivement un neurone gaussien et un neurone biais. W_{ij} peut être un des trois paramètres du sous-ensemble flou codé par le neurone i. Les modifications de ces poids dépendent de la valeur x de la variable d'entrée reçue par l'unité i[1]. En notant W_{i1} le poids correspondant à la moyenne et W_{i2} et W_{i3} les poids correspondant respectivement aux valeurs de σ_g et σ_d, on obtient :

1. Cette valeur correspond à la quatrième entrée du neurone.

$$\left.\begin{array}{l} \frac{\partial a_i}{\partial W_{i1}} = \frac{2(x-W_{i1})\exp\left(-\frac{(x-W_{i1})^2}{W_{i2}^2}\right)}{W_{i2}^2} \\ \frac{\partial a_i}{\partial W_{i2}} = \frac{2(x-W_{i1})^2\exp\left(-\frac{(x-W_{i1})^2}{W_{i2}^2}\right)}{W_{i2}^3} \end{array}\right\} \text{ si } x < W_{i1}$$

$$\left.\begin{array}{l} \frac{\partial a_i}{\partial W_{i1}} = \frac{2(x-W_{i1})\exp\left(-\frac{(x-W_{i1})^2}{W_{i3}^2}\right)}{W_{i3}^2} \\ \frac{\partial a_i}{\partial W_{i3}} = \frac{2(x-W_{i1})^2\exp\left(-\frac{(x-W_{i1})^2}{W_{i3}^2}\right)}{W_{i3}^3} \end{array}\right\} \text{ si } x \geq W_{i1} \qquad (4.15)$$

La valeur d'une seule des deux variances est ainsi modifiée lors de chaque pas d'apprentissage.

Paramètres des neurones gaussiens à deux dimensions Les indices i et j caractérisent ici respectivement un neurone gaussien à deux dimensions et un neurone biais. W_{ij} peut donc représenter un des six paramètres du sous-ensemble flou codé par le neurone i. Les modifications de ces poids dépendent de la valeur (x_1, x_2) de la variable d'entrée reçue par l'unité i. En notant W_{i11} et W_{i12} les poids correspondants aux moyennes et W_{i21}, W_{i22} et W_{i31}, W_{i32} les poids correspondant respectivement aux variances gauches et droites, on obtient :

$$\left.\begin{array}{l} \frac{\partial a_i}{\partial W_{i11}} = \frac{2(x-W_{i11})a_i}{W_{i21}^2} \\ \frac{\partial a_i}{\partial W_{i21}} = \frac{2(x-W_{i11})^2a_i}{W_{i21}^3} \\ \frac{\partial a_i}{\partial W_{i12}} = \frac{2(x-W_{i12})a_i}{W_{i22}^2} \\ \frac{\partial a_i}{\partial W_{i22}} = \frac{2(x-W_{i12})^2a_i}{W_{i22}^3} \end{array}\right\} \begin{array}{l} \text{si } x_1 < W_{i11} \\ \text{et } x_2 < W_{i12} \end{array} \qquad \left.\begin{array}{l} \frac{\partial a_i}{\partial W_{i11}} = \frac{2(x-W_{i11})a_i}{W_{i21}^2} \\ \frac{\partial a_i}{\partial W_{i21}} = \frac{2(x-W_{i11})^2a_i}{W_{i21}^3} \\ \frac{\partial a_i}{\partial W_{i12}} = \frac{2(x-W_{i12})a_i}{W_{i32}^2} \\ \frac{\partial a_i}{\partial W_{i32}} = \frac{2(x-W_{i12})^2a_i}{W_{i32}^3} \end{array}\right\} \begin{array}{l} \text{si } x_1 < W_{i11} \\ \text{et } x_2 \geq W_{i12} \end{array}$$

$$\left.\begin{array}{l} \frac{\partial a_i}{\partial W_{i11}} = \frac{2(x-W_{i11})a_i}{W_{i31}^2} \\ \frac{\partial a_i}{\partial W_{i31}} = \frac{2(x-W_{i11})^2a_i}{W_{i31}^3} \\ \frac{\partial a_i}{\partial W_{i12}} = \frac{2(x-W_{i12})a_i}{W_{i22}^2} \\ \frac{\partial a_i}{\partial W_{i22}} = \frac{2(x-W_{i12})^2a_i}{W_{i22}^3} \end{array}\right\} \begin{array}{l} \text{si } x_1 \geq W_{i11} \\ \text{et } x_2 < W_{i12} \end{array} \qquad \left.\begin{array}{l} \frac{\partial a_i}{\partial W_{i11}} = \frac{2(x-W_{i11})a_i}{W_{i31}^2} \\ \frac{\partial a_i}{\partial W_{i31}} = \frac{2(x-W_{i11})^2a_i}{W_{i32}^3} \\ \frac{\partial a_i}{\partial W_{i12}} = \frac{2(x-W_{i12})a_i}{W_{i31}^2} \\ \frac{\partial a_i}{\partial W_{i32}} = \frac{2(x-W_{i12})^2a_i}{W_{i32}^3} \end{array}\right\} \begin{array}{l} \text{si } x_1 \geq W_{i1} \\ \text{et } x_2 \geq W_{i12} \end{array} \qquad (4.16)$$

La valeur d'une seule des deux variances est ainsi modifiée pour chaque dimension d'entrée lors de chaque pas d'apprentissage. Comme nous l'avons déjà dit, lors des expériences réalisées nous n'avons pas étendu l'apprentissage à la deuxième dimension d'entrée des neurones de ce type pour les raisons citées plus haut. Seules les modifications des poids W_{i11}, W_{i21} et W_{i31} exposées dans cette équation sont donc utilisées.

4.4.3 Comparaison avec d'autres méthodes d'adaptation des systèmes flous

Nous avons choisi de présenter les systèmes hybrides neuro-flous en employant le formalisme des RNA. Il est cependant possible de les aborder comme une simple adaptation des algorithmes d'apprentissage des RNA à une forme particulière de systèmes d'inférence flous.

Il est donc nécessaire de comparer notre approche avec les autres méthodes d'apprentissage proposées dans la littérature pour les systèmes d'inférence flous. Il faut dans un premier temps distinguer dans ces travaux, l'apprentissage *paramétrique* qui consiste à modifier, comme nous cherchons à le faire les paramètres d'une base de règles existante, de l'apprentissage *structurel* qui consiste à essayer de créer une base de règles.

Cette dernière forme d'apprentissage ne nous intéresse pas directement puisque nous cherchons justement à exploiter les connaissances existantes. Des exemples de ce type pourront être trouvés dans les travaux de P. Reignier [136] ou de B. Kosko [81].

La majorité des méthodes d'apprentissage paramétrique rencontrées ne concerne que la modification d'une partie seulement des paramètres du système. Seul les poids des règles ou parfois leurs sorties sont adaptés par l'algorithme.

P. Garnier propose notamment de fixer les poids des règles par la méthode du simplexe [41] afin de régler l'influence des différents comportements utilisés pour piloter un robot.

P-Y. Glorennec [46] propose une méthode d'apprentissage par renforcement, dérivée du Q-Learning (voir §2.5.3.2), pour la modification des valeurs de sortie des règles.

L'utilisation du recuit simulé [85] ou des algorithmes génétiques [18, 61] à également été proposée pour permettre un réglage de l'ensemble des paramètres du système. Ces deux approches sont cependant lentes et assez lourdes à mettre en œuvre.

4.4.4 Implémentation du réseau

La programmation des algorithmes nécessaires à l'activation et à l'apprentissage de notre modèle de réseau a été réalisée en langage C, sous la forme d'une bibliothèque indépendante. Celle-ci permet la spécification et la sauvegarde d'un réseau à l'aide d'un langage de description que nous avons développé. Il est possible grâce à ce langage de spécifier plusieurs jeux de paramètres pour un réseau, correspondant chacun à une utilisation dans des conditions particulières.

L'utilisation d'une bibliothèque indépendante à permis une compilation directement sous le système VxWorks utilisé par l'architecture matérielle du robot. Nous avons également développé sur station UNIX et sur compatible PC une interface permettant de réaliser rapidement les opérations d'apprentissage, et de visualiser graphiquement l'évolution des paramètres (notamment les sous-ensembles flous).

La grammaire du langage de description ainsi que les fonctions de la bibliothèque sont décrits en annexe C.

4.5 Un exemple de manœuvre : le dépassement d'un véhicule

Nous présentons ici l'application sur simulateur de notre approche, pour une manœuvre assez représentative des résultats obtenus : le dépassement d'un obstacle constitué par un autre véhicule, et ce dans plusieurs situations. L'objectif peut être défini comme suit : *le robot doit quitter sa trajectoire de référence pour dépasser à une vitesse donnée un ou plusieurs autres véhicules. Ce dépassement doit se faire en conservant une distance de sécurité d'environ 1,5 mètres. Le robot doit ensuite rejoindre sa trajectoire.*
La base de règles floues utilisée est donnée en annexe de ce mémoire (annexe D). Elle se compose de 24 règles divisées en trois groupes correspondant à des comportements différents :

- le premier groupe (14 règles) concerne l'évitement d'obstacle à proprement parler. Ces règles s'activent lorsqu'un écho capteur se trouve dans l'une des zones virtuelles situées à l'avant du véhicule. Elles agissent sur la consigne de braquage de manière à diriger le robot dans une direction opposée à celle des obstacles. La consigne est d'autant plus forte que l'obstacle est proche ou qu'il se trouve en face du véhicule ;

- le deuxième groupe (5 règles) permet au robot de s'aligner sur le ou les véhicules dépassés lorsqu'il se trouve à une distance d'environ 1,5 mètres de celui-ci ou ceux-ci. Ces règles dépendent de la variable donnant l'orientation du robot par rapport aux obstacles latéraux et de la présence éventuelle d'autres obstacles à l'avant.

- le dernier groupe (5 règles) code un comportement d'attirance vers un but qui est le point de la trajectoire de référence à rejoindre. Celui-ci est choisi sur la trajectoire comme dans le chapitre précédent.

Les règles du premier groupe se voient affecter un poids initial double de celui des règles des deux autres groupes.
Durant la manœuvre, la consigne de vitesse demeure fixe et doit être fournie par le niveau supérieur du système.
Nous nous intéressons dans un premier temps aux résultats obtenus lors du dépassement d'un véhicule arrêté.
Le suivi de trajectoire avant et après la manœuvre utilise le contrôleur présenté dans le chapitre 3. Le choix de l'instant de déclenchement de la manœuvre sera détaillé dans le chapitre 5. Le système de commande est activé toutes les 50 millisecondes.
La base de règle utilisée sans aucun apprentissage fournit des résultats assez satisfaisants lorsqu'elle est employée à très faible vitesse. La distance entre les véhicules dépassés et le robot est supérieure à celle qui était attendue et que nous avons tenté d'obtenir par la spécification des règles. Toutefois nous n'observons pas d'oscillations et le dépassement s'effectue sans risque de collisions.

Les résultats se dégradent cependant fortement si nous augmentons la vitesse du robot. La figure 4.15 montre la trajectoire suivie, pour une situation similaire, à des vitesses de 0,5 $m.s^{-1}$, 3 $m.s^{-1}$ puis 5[1] $m.s^{-1}$.

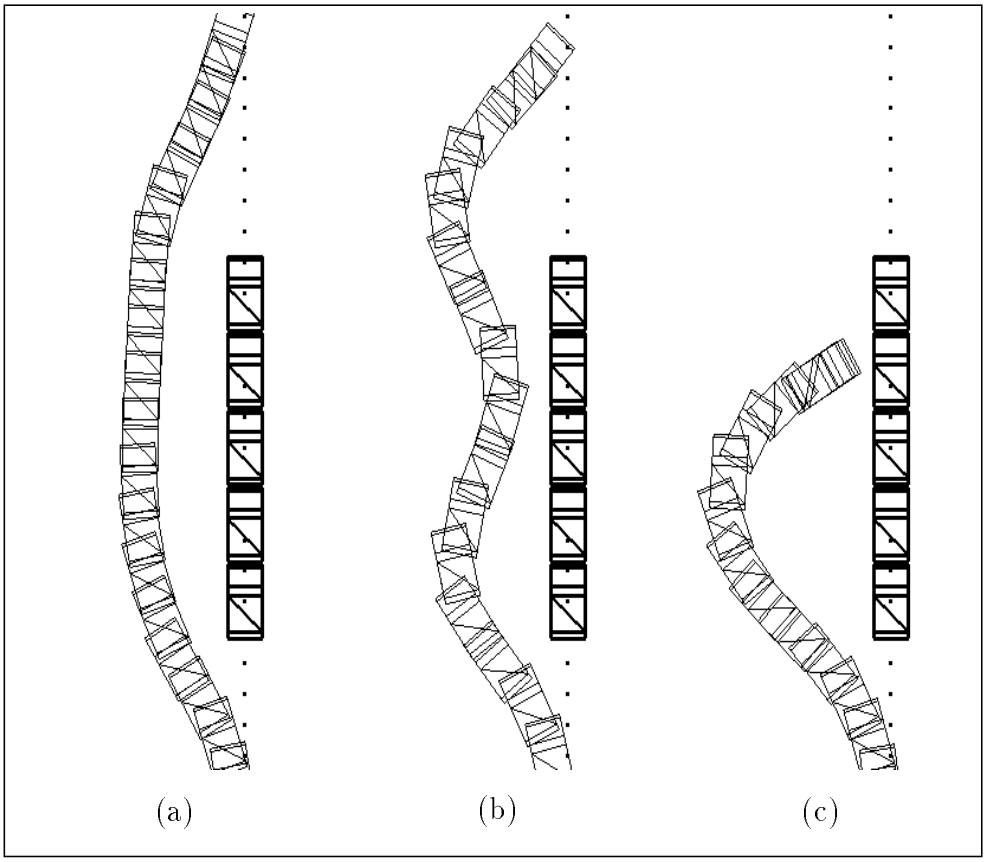

FIG. 4.15 – *Positions successives du robot lors du dépassement d'une file de véhicules à : (a) 0,5 $m.s^{-1}$, (b) 3 $m.s^{-1}$ puis (c) 5 $m.s^{-1}$. Les véhicules immobiles sont dessinés en gras. La trajectoire de référence est représentée en pointillés.*

La manœuvre effectuée à 5 $m.s^{-1}$ (figure 4.15 (c)) conduit à une collision si aucun mécanisme d'arrêt d'urgence n'est utilisé. Cette baisse de performances s'explique par l'inertie du véhicule qui ne peut changer instantanément son braquage. Le système nécessite un réglage fin des règles lorsque la vitesse du véhicule devient importante. Nous proposons donc l'utilisation de plusieurs jeux de paramètres, correspondant chacun à une utilisation dans des conditions de vitesse particulières[2].
Afin de réaliser l'adaptation des règles, nous avons collecté des exemples d'apprentissage correspondant à 3 situations (voir figure 4.16) :

– une manœuvre de dépassement d'un seul véhicule en environnement peu contraint

1. Cette dernière vitesse constitue une limite étant donnée la portée des capteurs utilisés.
2. Nous avons dans un premier temps tenté d'introduire la vitesse dans nos règles, mais cela a rendu leur conception plus complexe, et surtout leur nombre beaucoup plus important.

(le véhicule constitue le seul obstacle) ;

- une manœuvre de dépassement d'un véhicule dans un couloir offrant un espace d'évolution réduit ;
- un suivi d'une file de 5 véhicule en ligne droite à la distance 1,5 mètres.

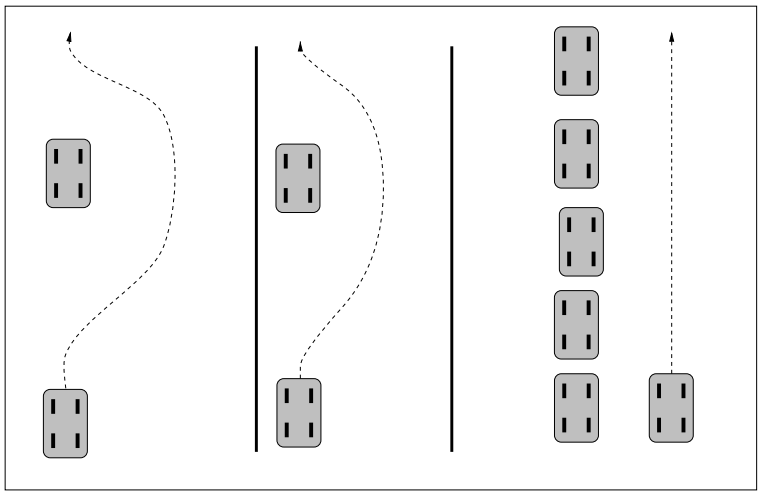

FIG. 4.16 – *Les trois situations d'apprentissage utilisées.*

Durant ces expériences nous avons enregistré à chaque pas de temps les valeurs des différentes entrées du système ainsi que la consigne de direction donnée par un opérateur humain. Ceci nous a permis de recueillir 360 exemples d'apprentissage en exécutant les manœuvres à la vitesse de 3 $m.s^{-1}$ et 215 en les exécutant à 5 $m.s^{-1}$.
Ces exemples ont été divisés en une base d'apprentissage (75 %) et une base de test (25 %). Nous reviendrons dans la section suivante sur les modalités de l'apprentissage.
La figure 4.17 expose l'évolution de l'erreur[1] commise par le réseau neuro-flou durant l'apprentissage des exemples correspondant à une évolution à 5 $m.s^{-1}$. L'apprentissage est arrêté lorsque l'erreur sur la base de test se stabilise.
Il est relativement difficile de donner une estimation quantitative du gain de performances apporté par l'apprentissage, sur ce type d'application, lorsque le système de commande est utilisé dans une situation nouvelle. Les figures 4.18 et 4.19 montrent par contre les différences de comportement entre deux véhicules évoluant à 5 $m.s^{-1}$ et commandés par le réseau neuro-flou respectivement avant et après l'apprentissage.
La figure 4.20 montre le comportement du véhicule dans la situation de la figure 4.15 (c) après l'apprentissage.
Les résultats obtenus restent qualitativement équivalents si l'on varie légèrement la vitesse d'exécution ou la taille des véhicules dépassés. Ils se dégradent par contre si le ou les véhicules dépassés sont en mouvement rapide. Ce cas constitue la deuxième expérience que nous présentons ci-dessous.

1. Moyenne de l'erreur quadratique sur l'ensemble des exemples.

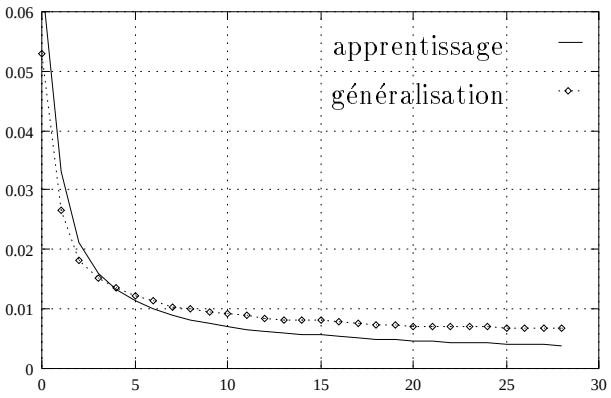

FIG. 4.17 – *Évolution de l'erreur sur les bases de test et d'apprentissage en fonction du nombre de pas d'entraînement.*

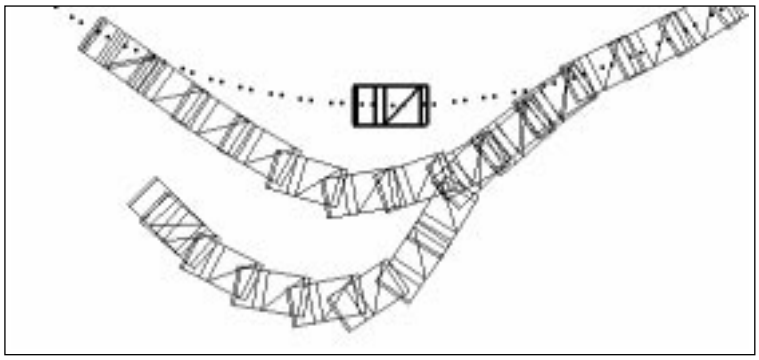

FIG. 4.18 – *Positions successives du véhicule lors d'un dépassement pendant le suivi d'une trajectoire circulaire. La trajectoire inférieure correspond à un véhicule utilisant le système neuro-flou original, l'autre correspond à un véhicule utilisant le système neuro-flou après apprentissage.*

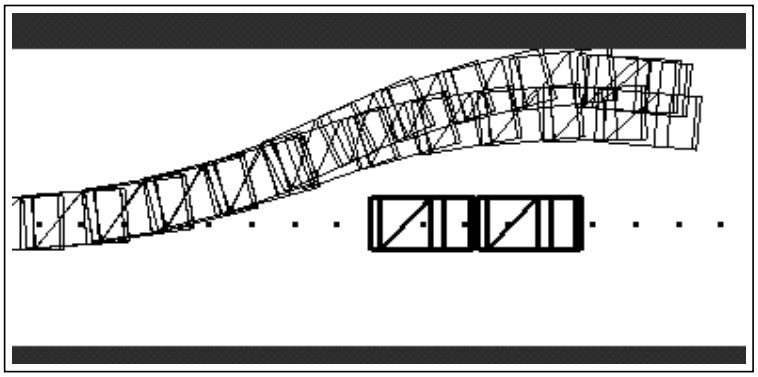

FIG. 4.19 – *Positions successives du véhicule lors d'un dépassement dans un environnement contraint. La trajectoire supérieure correspond à un véhicule utilisant le système neuro-flou original, l'autre correspond à un véhicule utilisant le système neuro-flou après apprentissage.*

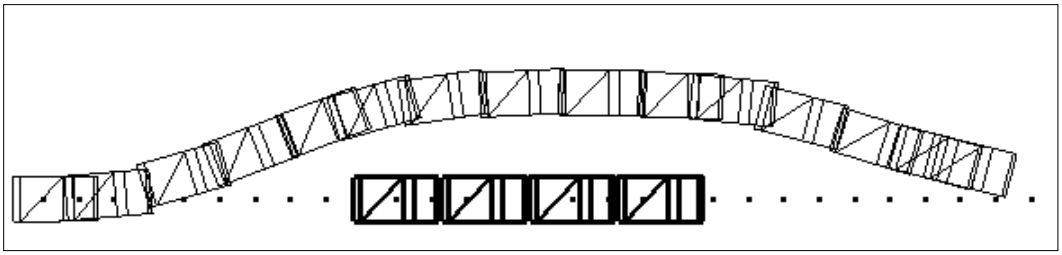

FIG. 4.20 – *Positions successives du véhicule lors d'un dépassement d'une file de véhicules à 5 $m.s^{-1}$ après apprentissage. La trajectoire avant apprentissage est représentée sur la figure 4.15 (c).*

Durant cette manœuvre la vitesse du robot est fixée 3 $m.s^{-1}$ au-dessus de celle de l'obstacle qui est supposée connue. La figure 4.21 montre le comportement du robot lors du dépassement d'un véhicule évoluant à 2 $m.s^{-1}$. Le système neuro-flou utilisé est l'un de ceux ayant été entraînés à partir du dépassement d'obstacles statiques, toutefois les résultats sont assez semblables si l'on utilise le réseau n'ayant subi aucun apprentissage. Les mauvaises performances observées s'expliquent par le traitement des échos capteurs sur la carte locale qui donne des résultats différents pour des obstacles immobiles ou en mouvement.

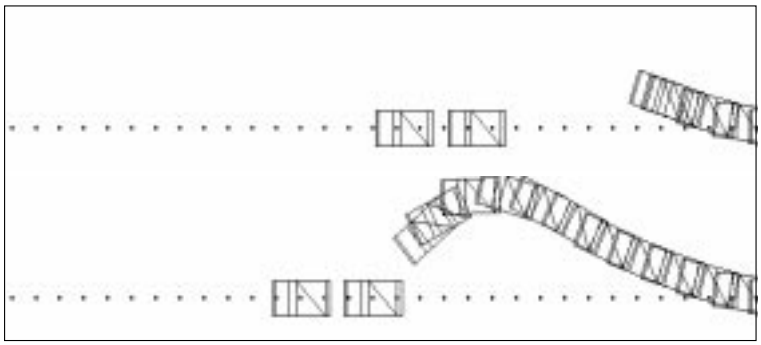

FIG. 4.21 – *Positions successives du robot lors du dépassement de deux véhicules roulant à 2 $m.s^{-1}$.*

Les exemples utilisés pour l'apprentissage correspondent ici à trois situations de dépassement : un véhicule roulant à 2 $m.s^{-1}$, un véhicule roulant à 1,5 $m.s^{-1}$ et trois véhicules se suivant à la vitesse de 2 $m.s^{-1}$.

La figure 4.22 présente les résultats obtenus après cet apprentissage.

Cette manœuvre sera reprise dans le chapitre 5.

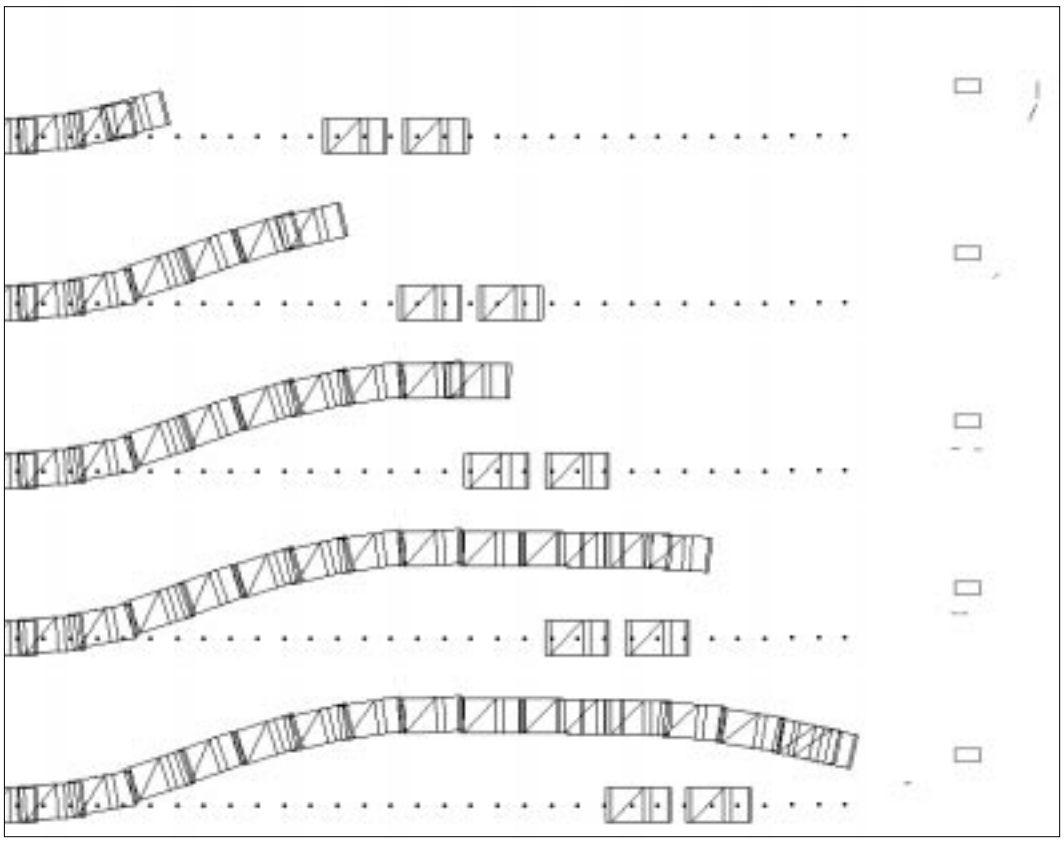

FIG. 4.22 – *Positions successives du robot lors du dépassement de deux véhicules, roulant à* $2\,m.s^{-1}$*, après apprentissage. La partie droite de la figure montre les informations présentes sur la carte locale autour du véhicule.*

4.6 Discussion sur l'apprentissage dans notre réseau

L'apprentissage concerne plusieurs types de paramètres dans notre modèle de réseau : les valeurs de sortie et les poids des règles ainsi que la forme et la position des sous-ensembles flous. L'influence de cette adaptation sur les performances globales du système diffère selon la famille de paramètres considérée.
Les possibilités de modification des sorties et des poids des règles sont indispensables pour permettre une spécification précise du traitement effectué dans toutes les conditions.
Le réglage des poids permet en donnant une importance relative aux différentes règles, de modifier leur influence en fonction du nombre et de la nature des autres règles activées. Une règle dirigeant le véhicule vers son but ne doit pas avoir la même influence sur la consigne de sortie si elle s'active seule ou si une règle d'évitement d'obstacle est également activée.
Le réglage des sorties permet par contre de modifier l'effet d'une règle indépendamment

des autres. Il est donc indispensable si l'on veut pouvoir intervenir sur les consignes délivrées par le système lorsqu'une seule règle ou un seul groupe de règles ayant toutes le même effet s'active.
La modification des paramètres des sous-ensembles flous permet un réglage plus facile et plus précis du comportement du système mais peut avoir dans certains cas un effet négatif sur les capacités de généralisation obtenues. Pour que le traitement réalisé par le réseau soit cohérent sur l'ensemble de l'espace d'entrée, les différents sous-ensembles flous doivent en effet se recouvrir[1] de manière importante et régulière. Puisque nous n'utilisons que peu d'exemples d'apprentissage, il existe un risque qu'un sous-ensemble flou soit déplacé pour concorder avec la zone de l'espace d'entrée dans laquelle se trouvent ces exemples, ou au contraire vers une zone où son degré de vérité reste toujours proche de zéro pour ces exemples. Il peut alors exister une plage de valeurs de la variable correspondante pour laquelle plus aucun sous-ensemble n'est activé.
Nous avons constaté ce comportement principalement lorsqu'une règle est en contradiction avec les exemples d'apprentissage ou avec les autres règles. Il peut donc constituer un moyen de vérifier la cohérence des règles employées, afin de les modifier si nécessaire.
La figure 4.23 montre par exemple l'évolution des sous-ensembles flous associés à la variable d'entrée correspondant au capteur virtuel «droite» (zone 6 sur la figure 4.5), si nous introduisons la règle suivante dans la base utilisée :

$$\textit{Si droite est moyen alors direction est } -1.0$$

La consigne de sortie donnée par cette règle est en contradiction avec le comportement spécifié par les autres règles et par les exemples d'apprentissage. Ceux-ci sont ceux utilisés dans la section précédente pour le dépassement d'un obstacle statique à la vitesse de 5 $m.s^{-1}$. Le sous-ensemble flou «moyen» est déplacé vers une zone ou son degré de vérité reste faible pour tous les exemples d'apprentissage. La figure 4.23 présente une coupe des sous-ensembles flous suivant la première dimension d'entrée. La valeur de la deuxième dimension (distance latérale) y est fixée à une valeur nulle.
Lorsque la base de règle est choisie suffisamment proche du résultat final attendu, les modifications apportées aux ensembles flous restent minimes. L'apparition d'une évolution du type de celle décrite par la figure peut donc être utilisée comme un signal indiquant la nécessité de modifier la base de règles avant de recommencer l'apprentissage[2].
De manière plus générale il est apparu, au cours des expériences réalisées, que ce type de réseau était incapable d'apprendre correctement les exemples présentés si son architecture ou ses paramètres initiaux étaient choisis trop éloignés des valeurs optimales. Ceci

1. Dans le cas où les sous-ensembles flous sont définis par des gaussiennes il est possible d'utiliser la définition du taux de recouvrement introduit à propos des unités d'entrée des réseaux de type RBF (voir équation 3.5).

2. Ceci introduit un cycle, *Compilation des règles dans le réseau* ⇒ *Apprentissage* ⇒ *Vérification et choix de nouvelles règles* ⇒ *Compilation des règles dans le réseau* , similaire à ce qui est fait à un tout autre niveau dans certains systèmes hybrides neuro-symboliques plus complexes comme par exemple le système INSS [122].

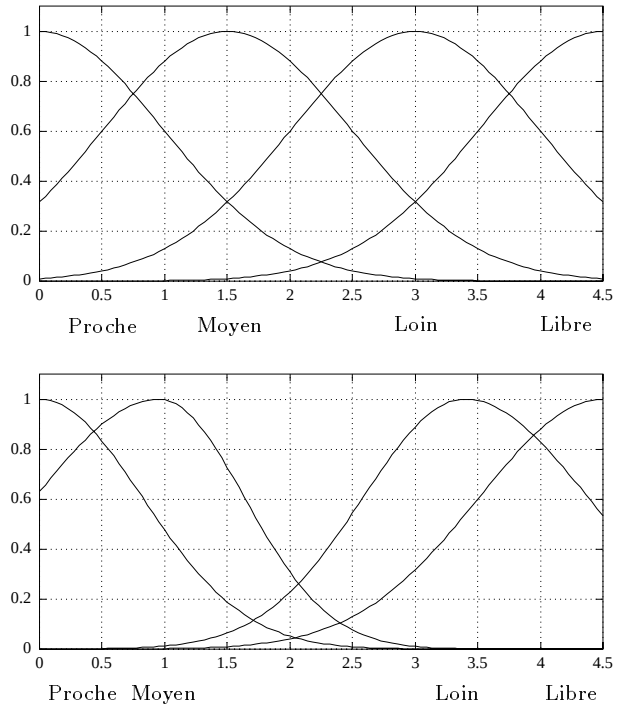

FIG. 4.23 – *Évolution des sous-ensembles flous dans le cas d'une base de règles contradictoire. Commentaires dans le texte.*

peut s'expliquer en partie par le fait que nous n'employons pas uniquement, contrairement a ce qui se fait dans les réseaux multi-couches classiques, des fonctions d'activations strictement monotones dans nos unités. Il existe donc un risque accru de rester bloqué dans un minimum local de la fonction d'erreur. De même puisque les sous-ensembles flous se comportent comme des champs recepteurs locaux, et que les règles ne traitent pas toutes les combinaisons possibles entre les différentes valeurs des variables d'entrée, il est bien sûr impossible de réaliser un apprentissage si les règles initiales ne sont pas activées de manière significative par les exemples.

Ces problèmes peuvent toutefois être évités en choisissant de manière adéquate l'architecture initiale du réseau.

Nous avons testé un grand nombre de valeurs des constantes d'apprentissage lors des expériences réalisées. Les meilleurs résultats ont été obtenus en utilisant pour l'adaptation des poids correspondant aux paramètres des sous-ensembles flous, une valeur de la constante η (pas du gradient) inférieure à celle utilisée pour les autres poids. Les résultats présentés dans la section précédente correspondent aux valeurs $\eta = 0.04$ pour les poids et les sorties des règles, $\eta = 0.02$ pour les ensembles flous et $\mu = 0.01$ dans tous les cas.

4.7 Conclusion

Dans ce chapitre nous avons présenté une approche permettant de réaliser par apprentissage supervisé, des systèmes de commande basés uniquement sur les capteurs extéroceptifs du véhicule, en n'utilisant qu'un nombre d'exemples d'apprentissage réduit.
Les contrôleurs réalisés peuvent être considérés à la fois comme des modèles de réseau de neurones artificiels d'un type spécial et comme des systèmes d'inférence flous utilisant des opérateurs originaux. Ils amènent quel que soit le point de vue adopté des apports particuliers :

- l'apprentissage permet d'obtenir de manière automatique un réglage fin du système d'inférence flou. Ce réglage reste bien entendu réalisable «à la main» mais cela implique de nombreux tâtonnements et des essais sur le véhicule qui peuvent être dangereux ;

- l'utilisation de la connaissance symbolique issue des règles floues permet de contraindre la structure initiale du réseau et de se contenter d'un nombre réduit d'exemples d'apprentissage. Le traitement effectué par le système de commande obtenu reste de plus compréhensible, contrairement à celui effectué par un réseau multi-couches classique qui se comporte comme une «boite noire». Il est en effet possible de connaître le sens des dépendances entre les variables d'entrée et les consignes de sortie. Cette possibilité d'explicitation permet au moins de garantir que le système ne réagira pas dans une situation nouvelle d'une manière totalement opposée à celle qui est attendue.

Nous sommes toutefois conscients que la validation de l'approche proposée ne pourra être effective qu'une fois réalisés des tests sur le véhicule réel. La pauvreté des informations capteurs utilisées durant les simulations (voir annexe B) nous laisse cependant optimistes quant aux résultats attendus pour ces tests qui sont actuellement en cours. Le travail réalisé ne constitue par contre qu'une première étude. Il nous semble notamment nécessaire d'utiliser une modélisation plus détaillée de la carte locale faisant le lien avec des données connues *a priori* sur l'environnement statique.
L'utilisation de capteurs plus riches (notamment la vision) nous semble de plus nécessaire pour garantir un système de commande véritablement robuste. L'utilisation des capteurs virtuels permettrait dans ce cas de ne rien changer à l'approche proposée puisqu'il suffirait d'integrer les données issues des nouveaux capteurs physiques à la carte locale.

Chapitre 5

Réseaux de neurones artificiels et enchaînement de manœuvres

Lors des deux chapitres précédents nous nous sommes intéressés à l'utilisation des réseaux de neurones artificiels pour la réalisation de deux types différents de systèmes de commande spécialisés. L'accomplissement de missions complexes impose, comme nous l'avons vu au début de ce mémoire, de pouvoir choisir parmi des contrôleurs de ce type en fonction d'un plan établi à l'avance mais aussi et surtout en réponse à des événements extérieurs. Cette tâche correspond au niveau intermédiaire d'une architecture de commande, défini dans la section 1.4.
Nous montrons dans ce chapitre les possibilités offertes par l'application d'un modèle particulier de RNA, développé au sein de l'équipe Réseaux d'Automates du laboratoire LEIBNIZ, pour la réalisation de ce niveau d'une architecture de commande.

5.1 Objectifs

L'exécution d'une manœuvre complexe peut demander l'enchaînement de plusieurs actions impliquant chacune l'utilisation d'un système de commande particulier ou d'une instance d'un de ces systèmes de commande. Ces actions doivent être sélectionnées en réponse à un contexte d'exécution donné.
Plusieurs approches simples permettent de réaliser un système de sélection de ce type. Il est possible d'utiliser un automate[1] dont les différents états correspondent à l'utilisation d'une loi de commande particulière. La sélection peut également être réalisée par un système de règles symboliques. Quelle que soit l'approche choisie, l'implémentation du système doit permettre de respecter les contraintes induites par l'évolution à forte vitesse du véhicule.
Le système doit être capable de réagir de manière très rapide à des événement provenant de sources diverses, travaillant dans des échelles de temps différentes. Les informations prises en compte peuvent en effet provenir de différents capteurs extéroceptifs, d'une

1. Celui-ci peut être par exemple représenté par un script (cf §1.3.3.3).

communication avec l'extérieur ou encore d'informations internes. Certaines de ces données peuvent varier à des fréquences très importantes (e.g. vitesse ou accélération du robot) ou n'être au contraire disponibles que plus rarement (e.g. communication avec les autres véhicules).
Afin de pouvoir prendre en compte en temps réel, n'importe quel événement ou changement de données, le système de sélection doit posséder un temps de réaction le plus faible possible.
Durant la période, appelée période d'autonomie, pendant laquelle le système calcule un nouvel état ou une nouvelle décision à partir des variables d'entrées et éventuellement de ses états passés, les nouveaux événements extérieurs ne peuvent en effet généralement pas être pris en compte. On dit alors que le système est autonome et isolé.
Dans le cas où cette période d'autonomie est importante, comme par exemple pour un système expert symbolique classique, il existe forcement un délai plus ou moins long entre le déclenchement d'un événement et sa prise en compte par le système.
Si la période entre deux variations de certaines données est inférieure à la période d'autonomie, il est également possible que certains événements ne soient pas traités du tout.
Une solution pour réduire la période d'autonomie d'un système peut consister à remplacer un seul traitement utilisant une suite plus ou moins longue d'opérations par *une mise à jour régulière et récurrente de l'état interne du système ne mettant en jeu que quelques opérations simples et rapides* [87]. Chacune de ces mises à jour utilisant les valeurs des données d'entrée, toute modification de ces données à beaucoup plus de chance d'être prise en compte.
Nous nous intéressons plus particulièrement ici, à la manière dont certains modèles de réseaux de neurones récurrents peuvent être utilisés pour cette tâche. En effet, dans ces modèles l'activation des unités évolue constamment et est calculée à partir d'équations relativement simples.

5.2 Un modèle de RNA récurrent non isolé

Les modèles de réseaux de neurones artificiels que nous avons utilisés ou présentés jusqu'ici, font partie de la famille des réseaux à propagation unilatérale. Les connexions qui propagent l'information dans ces modèles, définissent un sens unique allant des unités d'entrée vers celles de sortie, en passant par les neurones cachés. Le calcul des valeurs d'activation se fait donc, dans ces réseaux, en une seule passe lors de chaque présentation de nouvelles données d'entrée.
Dans les réseaux récurrents, dont le graphe orienté des connexions comporte des cycles, il se crée au contraire une dynamique d'évolution des activations des différentes unités. Ces activations peuvent se stabiliser sur des valeurs stationnaires[1] ou des cycles limites ou rester, à l'opposé, totalement chaotiques.

1. Ces états stationnaires sont appelés attracteurs.

Cette évolution progressive des valeurs des unités peut être exploitée dans certains modèles, contrairement à ce qui se passe par exemple dans les réseaux multi-couches pour lesquels seule la valeur finale de sortie est généralement utilisée.
Les réseaux récurrents peuvent donc être considérés comme des systèmes dynamiques [1] et être étudiés avec les outils classiques de ce domaine.
L'évolution du vecteur, noté A, des activations (ou états) des unités d'un de ces réseaux peut être décrit par un système d'équations différentielles, généralement non linéaires, de la forme suivante :

$$\frac{dA}{dt} = F(A, E), \tag{5.1}$$

où E est le vecteur des valeurs d'entrée. Le réseau est dit *isolé* si ce vecteur est considéré comme constant (i.e. si ses variations ne sont pas prises en compte) durant la convergence de l'activation des unités.
Nous pouvons noter que l'implémentation d'un réseau de ce type implique généralement une dicrétisation de l'équation 5.1 :

$$\Delta A = F(A, E)\Delta t \tag{5.2}$$

Puisque notre but est d'obtenir un système capable de réagir à tout moment à des changements de données, nous nous sommes plutôt intéressés aux modèles non isolés. Nous présentons plus particulièrement dans ce qui suit, la structure d'un modèle de réseau proposé par A. Labbi [86], dans l'équipe Réseaux Neuronaux du LEIBNIZ.

5.2.1 Dynamique du modèle

Le modèle de réseau auquel nous nous intéressons dans cette section dérive du Shunting Model de S. Grossberg [50]. Il a été introduit par A. Labbi, sous le nom de réseau *versatile*, pour modéliser le processus de décision d'un expert dans un environnement en continuel changement.
Il se compose de deux groupes (ou couches) d'unités (voir figure 5.1). La première de ces couches (couche d'entrée) reçoit les données extérieures, et les transmet aux unités de la deuxième couche, à travers des connexions à sens unique. Les unités de la deuxième couche (couche de sortie) sont par contre totalement interconnectées entre elles. Les connexions de la couche d'entrée vers la couche de sortie ont un effet positif (excitation) sur l'activation des unités de sortie. Les liaisons entre les neurones de sorties ont au contraire un effet négatif (inhibition) sur leurs activations.
L'évolution de l'activation a_i d'une unité de sortie d'indice i, dans un réseau comportant m entrées et n sorties, est donnée par l'équation suivante :

$$\begin{aligned} \frac{da_i}{dt} &= F(A, E) \\ &= -c_i a_i + (1 - a_i).(\sum_{k=1}^{m} w_{ik}.f_k(e_k)) - (1 + a_i).(\sum_{j=1}^{n} d_{ij}.g_j(a_j)) \end{aligned} \tag{5.3}$$

1. Un système dynamique peut être défini comme *un système dont l'état varie avec le temps*[56].

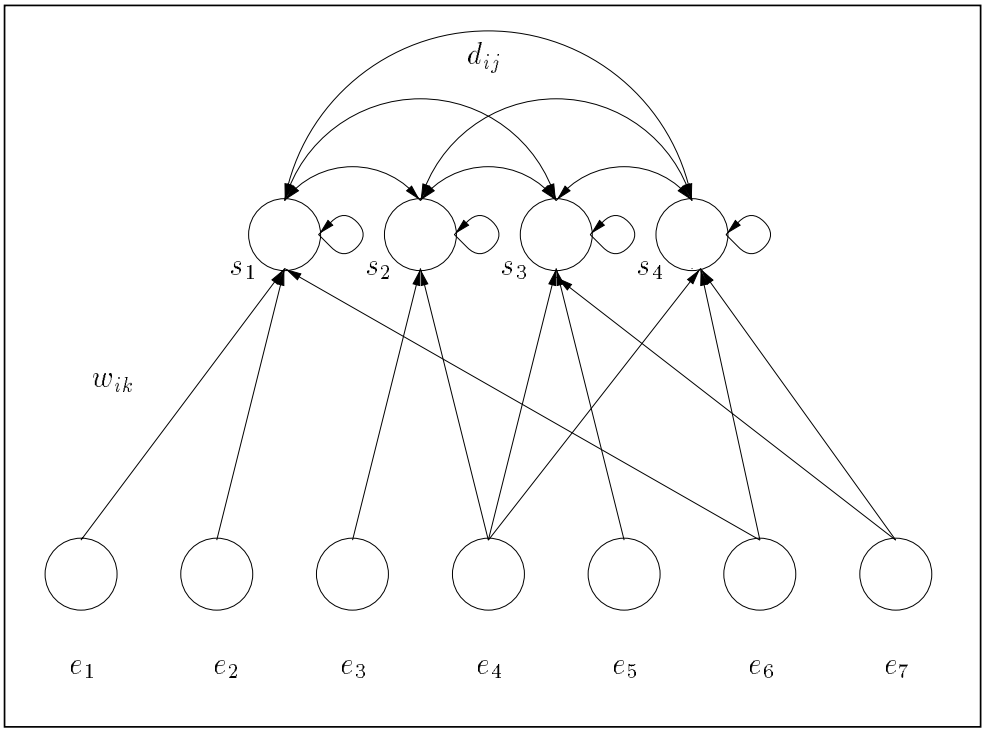

FIG. 5.1 – *Un exemple de réseau versatile.*

Dans cette équation, c_i est une constante positive appelée coefficient de fatigue, w_{ik} et d_{ij} sont les poids des connexions reliant l'unité i, respectivement à une unité d'entrée d'indice k et à une unité de sortie d'indice j, et e_k est l'activation du neurone d'entrée d'indice k. Les fonctions g_j et f_k sont des fonctions monotones positives qui peuvent être identifiées à des sigmoides.

On retrouve dans cette expression le terme inhibiteur $-(1+a_i).(\sum_{j=1}^{n} d_{ij}.g_j(a_j))$ qui dépend de l'activation des autres unités de sortie. Ce terme a pour effet de créer une compétition entre ces unités. Le facteur $(1+a_i)$ permet de limiter l'inhibition lorsque l'activation a_i approche de la valeur minimum -1.

Le terme $(1-a_i).(\sum_{k=1}^{m} w_{ik}.f_k(e_k))$ correspond au terme excitateur de l'expression. Il dépend des différentes entrées du réseau. Le facteur $(1-a_i)$ permet ici de limiter l'activation lorsque celle-ci approche de la valeur maximum +1.

Enfin, le terme $-c_i a_i$ permet de diminuer l'activation de l'unité lorsque celle-ci n'est plus excitée.

5.2.2 Propriétés

Les propriétés mathématiques de ce modèle ont été étudiées en détail par A. Labbi. Nous nous contentons de présenter ici les principaux résultats.

Pour pouvoir utiliser un réseau de ce type comme système décisionnel, il est tout

d'abord nécessaire que sa dynamique soit convergente[1] pour tout vecteur d'entrée E et pour toute condition d'activation initiale A_0. Ce résultat est obtenu sur ce modèle grâce au théorème de Cohen-Grossberg[25], à la condition que les poids d_{ij} soient symétriques (i.e. $d_{ij} = d_{ji}$).
Il est également nécessaire de s'assurer que le réseau construit bien une application, au sens mathématique du terme, de l'espace d'entrée vers l'espace des états d'équilibre stationnaires. Il faut pour cela vérifier que pour un vecteur d'entrées E donné, le réseau converge toujours vers le même état et ce quel que soit son état initial. Cette propriété baptisée *convergence globale* est prouvée pour ce modèle, à partir d'un théorème énoncé par M. W. Hirsh [60], dans le cas où les conditions suivantes sont respectées :

- les fonctions $g_i(x)$ utilisées dans le terme inhibiteur sont des fonctions sigmoides

$$g_i(x) = \frac{1}{1 + \exp(-x)} \; i \in \{1, ..., n\},$$

  où n est le nombre d'unités de sortie.

- les paramètres de fatigue c_i sont choisis de telle sorte que l'on ait :

$$Min_{i \in \{1,...,n\}} \; c_i \; > ||D||_1, \tag{5.4}$$

  où D est la matrice définie par $(D)_{ij} = d_{ij}$ et $||D||_1 = Max_{i \in \{1,...,n\}} \sum_{j=1}^{n} |d_{ij}|$.

Une démonstration complète de ces résultats pourra être trouvée dans la thèse de Labbi [87].
L'une des caractéristiques principales des réseaux de ce type est leur très faible période d'autonomie. Celle-ci se résume en effet au calcul d'un pas d'itération dans le système discret associé au réseau (voir équation 5.2). Ceci est d'autant plus vrai qu'il est facile de réaliser une version sur circuit imprimé d'un tel modèle. De nombreux travaux ont en effet été réalisés à ce sujet autour du Shunting Model de Grossberg dont le fonctionnement est très proche de celui de notre réseau [77].

Dans le cas ou l'une des valeurs d'entrée change durant la convergence vers un état d'équilibre, le réseau est donc capable de prendre en compte ce changement pour tendre vers un attracteur différent. La cohérence du choix de ce nouvel attracteur est garantie par la propriété de convergence globale énoncée ci-dessus. Le réseau peut donc *changer d'avis* à tout moment, et ce quel que soit son état. On dit qu'il est parfaitement *versatile*.

L'apprentissage dans un modèle de ce type est par contre problématique. Plusieurs extensions de l'algorithme de rétropropagation du gradient, adaptées à l'utilisation sur des réseaux récurrents, ont été proposées. Parmi celles-ci nous pouvons citer la rétropropagation dans le temps [164] ou la rétropropagation récurrente [130]. Aucun

1. On parle de dynamique convergente lorsque l'activation des unités se stabilise sur un état d'équilibre stationnaire.

de ces algorithmes ne permet toutefois de respecter les contraintes sur les différents paramètres qui permettent de garantir les propriétés de covergence énoncées ci-dessus. Nous allons voir toutefois, dans la section suivante, comment les poids du réseau peuvent être choisis pour reproduire un comportement particulier.

5.2.3 Codage de règles de décision symboliques

Notre but est d'utiliser le réseau versatile pour évaluer le résultat d'un ensemble de règles symboliques, régissant le choix d'un système de commande ou d'une instance particuliere d'un système de commande. Nous faisons donc concorder chaque unité de sortie avec une décision. Les entrées doivent correspondre, pour leur part, aux différents événements ou aux différentes informations qui conditionnent le déroulement de la manœuvre. Ces données peuvent prendre des valeurs binaires (e.g. la voie est libre ou non) ou réelles (e.g. la vitesse du véhicule). Dans le deuxième cas il est généralement suffisant, comme nous le verrons dans l'exemple présenté plus loin, de découper la plage des valeurs possibles d'une variable d'entrée en plusieurs intervalles. A chacun de ces intervalles peut alors être associée une valeur binaire, indiquant si la valeur de la variable se trouve ou non dans l'intervalle.
Nos règles sont donc de la forme :

$$Si\ entrée1\ \in\ [a_1, b_1]\ et\ entrée2\ \in\ [a_2, b_2]\ ...\ et\ entrée3\ alors\ décision1,$$

où $[a_1, b_1], ...[a_2, b_2]$ sont des intervalles définis respectivement sur les variables d'entrée réelles $entrée1$ et $entrée2$, et $entrée3$ est une variable binaire. Ceci permet d'associer à chaque entrée du réseau une valeur binaire correspondant à la valeur de vérité d'une proposition utilisée dans les prémisses des règles.
Labbi propose une méthode de codage de règles de ce type dans une version très légèrement modifiée du réseau [1]. Cette approche s'appuie sur la notion de site assez largement employée dans le domaine des RNA. Un site est un regroupement de connexions d'un même type. Chaque site d'une unité de sortie correspond à une règle, et reçoit des connexions depuis toutes les unités d'entrée intervenant dans la partie prémisse de cette règle. Les connexions vers le site d'indice k, du neurone d'indice i, se voient affecter un poids égal à $\frac{1}{p_{ik}}$, où p_{ik} est le nombre de propositions qui composent la prémisse de la règle correspondante.
Le terme excitateur de l'équation différentielle régissant l'évolution de l'activation de l'unité d'indice i devient alors :

$$(1 - a_i)(\sum_{k=1}^{r_i} f_i(S_{ik})),$$

où r_i est le nombre de sites de l'unité i. Dans cette équation S_{ik} représente la somme des valeurs transmises par les connexions parvenant au site k de l'unité i. Cette

1. Ceci n'affecte en rien les propriétés de convergence énoncées précédemment.

somme vaut 1 lorsque toutes les propositions qui composent la prémisse de la règle sont vérifiées.
Tous les poids correspondant aux connexions entre les unités de sortie sont fixés à la même valeur. Les coefficients de fatigue sont choisis pour respecter la contrainte donnée par l'équation 5.4.
Nous avons légèrement modifiée la méthode de choix des poids pour lui permettre de traiter des négations dans les prémisses des règles (i.e. des propositions du type *Si non entrée*1 ou *Si entrée*1 *est FAUX*). Les entrées du réseau reçoivent les valeurs +1 ou -1 suivant que la proposition correspondante est vraie ou fausse. Le poids des connexions parvenant à un site est fixé à $\pm\frac{1}{p_{ik}}$, où p_{ik} est le nombre de propositions qui composent la prémisse de la règle correspondante. Ce poids est négatif si la connexion correspondante provient d'une entrée qui apparaît accompagnée d'une négation dans la règle, et positif dans le cas contraire.
De même, nous utilisons en lieu et place des sigmoides la fonction $f_i(x) = (\frac{1+x}{2})^p$ dans le terme exitateur de l'équation 5.3 (p est une constante positive). Cette fonction permet de donner à un site une valeur assez faible, lorsque la quasi totalité des propositions composant la prémisse de la règle correspondante sont vérifiées, mais qu'une au moins de ces proposition est fausse. Le site prend par contre bien la valeur 1 lorsque toutes ces propositions sont vérifiées. Ceci permet d'éviter qu'une décision pour laquelle plusieurs règles ne sont qu'en partie vérifiées ne l'emporte sur une décision pour laquelle une seule règle est entièrement vérifiée.
L'utilisation d'un modèle de ce type pour représenter un système de règles symbolique entraîne une propriété très importante. Dans un système expert classique, lorsqu'une information est manquante en raison d'une panne ou qu'un cas particulier n'a pas été prévu, le système est incapable de donner une réponse.
Si un cas similaire se présente à notre réseau, dans lequel le choix se fait non pas par recherche d'une règle correspondant exactement à la situation mais par l'instauration d'une compétition entre les différentes solutions, la règle dont la prémisse est la plus fortement activée l'emporte sur les autres.
De même si plusieurs règles préconisant des choix différents sont activées, ce n'est pas le premier d'entre eux dans l'ordre des règles qui sera sélectionné mais celui qui est appuyé par le plus grand nombre de règles.

5.3 Un exemple de manœuvre de haut niveau

5.3.1 Objectifs

L'exemple auquel nous nous intéressons se place dans la continuité de ceux que nous avons présenté dans les deux chapitres précédents. Dans cette mission, le véhicule doit être capable de suivre une trajectoire de référence (correspondant à une voie de circulation), et de dépasser d'autres véhicules présents sur sa route lorsque leur vitesse est plus faible que la sienne et que le dépassement est possible. Il doit également pouvoir

se rabattre rapidement en cas de problème durant sa manœuvre. La vitesse maximum d'évolution du véhicule durant la manœuvre est de 5 mètres par seconde.
L'exécution de cette manœuvre fait intervenir plusieurs systèmes de commande spécialisés :

- un contrôleur chargé du suivi de trajectoires. Il s'agit dans notre simulation du système présenté dans le chapitre 3. Ce contrôleur peut prendre la commande du robot dans trois situations distinctes auxquelles sont associées des consignes de vitesses différentes :
  - la voie est libre : la vitesse est alors une consigne associée à la trajectoire de référence ;
  - un autre véhicule se trouve sur la même voie de circulation mais le dépassement est impossible : la vitesse est alors celle du véhicule obstacle ;
  - un autre véhicule se trouve sur la même voie de circulation mais à une distance trop proche pour permettre un dépassement sans risque : la vitesse est alors choisie inférieure d'un mètre par seconde à celle de ce véhicule ;
- un contrôleur chargé du dépassement à proprement parler que nous avons présenté dans le chapitre 4. Ce contrôleur admet plusieurs jeux de paramètres correspondants au dépassement, à des vitesses différentes, par la gauche ou par la droite, d'un véhicule immobile ou d'un véhicule en mouvement ;
- un contrôleur réalisé selon la même technique (réseau neuro-flou) ayant pour but le rabattement rapide du véhicule en cas de détection d'un nouvel obstacle durant la manœuvre ;
- un troisième système de commande du même type (réseau neuro-flou) chargé du pilotage lorsque le véhicule est dans l'obligation de reculer ;
- un système d'arrêt d'urgence activé en cas de risque de collision.

Le système de sélection doit donc être capable, en fonction de la situation courante, de donner la main à l'un de ces systèmes de commande et de sélectionner le cas échéant un jeu de paramètres. Le nombre de possibilités (sorties du réseau utilisé) est de 25 (16 instanciations différentes du réseau d'évitement, 3 situations de suivi de trajectoire, rabattement sur la gauche ou la droite, arrêt, recul, arrêt d'urgence et enfin appel au niveau supérieur en cas d'échec).
L'ensemble des règles codées par le réseau utilisent plusieurs informations, qui sont comme nous l'avons vu plus haut, soit des valeurs booléennes correspondant à la détection d'une caractéristique particulière, soit des variables continues découpées en plusieurs intervalles. Ces dernières sont, dans le cas de cette mission, les suivantes :

- la vitesse associée à la portion courante de la trajectoire de référence. Cette variable est découpée en 5 intervalles de tailles équivalentes correspondant à des valeurs entre 0 et 5 $m.s^{-1}$;

- la vitesse d'un éventuel véhicule obstacle. Cette valeur est également découpée en quatre intervalles. Elle peut être obtenue, sur le robot réel, soit par communication entre les véhicules, soit par dérivation de l'information de distance ;

- la distance entre le robot et le véhicule obstacle, découpée elle aussi en 5 intervalles. Cette information est calculée à l'aide des zones de la carte locale correspondant à l'avant du robot ;

Le réseau utilisé possède 14 autres entrées correspondant aux informations suivantes :

- détection d'un risque de collision si le robot suit la trajectoire de référence. Cette information est obtenue à partir des obstacles répertoriés sur la carte locale, en cherchant de manière géométrique les intersections entre ces derniers et un échantillon des positions futures du véhicule ;

- présence ou non d'obstacles dans une zone située à l'avant du véhicule ;

- présence ou non d'obstacles interdisant un dépassement sur la droite ou la gauche du robot. Cette information pourrait être enrichie par une communication avec le véhicule dépassé. Nous n'employons toutefois que les valeurs des capteurs dans notre simulation ;

- présence ou non d'obstacles à l'arrière du véhicule ;

- information sur la vitesse actuelle du robot (vitesse positive ou non, vitesse supérieure à celle de l'obstacle ou non) ;

- indication sur la manœuvre en cours de réalisation (i.e. rabattement d'urgence ou dépassement normal) ;

- présence d'obstacles sur la carte locale, dans une zone proche du robot, impliquant un arrêt d'urgence. Cette zone est choisie de forme rectangulaire et de taille proportionnelle à la vitesse. Il serait toutefois préférable de lui donner une forme variable dépendant du braquage du véhicule, sur le modèle des zones virtuelles déformables proposées par R. Zapata [172].

Le détail de ces entrées ainsi que les règles utilisées sont donnés en annexe E.
Le comportement du robot régit par ces règles est le suivant :

- lorsque aucun obstacle ne menace le véhicule, celui ci se contente de suivre sa trajectoire à la vitesse préconisée ;

- si un autre véhicule circule sur la même voie à une vitesse inférieure, le robot tente de le dépasser, en choisissant de doubler de préférence par la gauche ;

- si ce dépassement est impossible en raison de la présence d'autres obstacles, le robot se contente de suivre le deuxième véhicule jusqu'à ce que le dépassement soit possible (il s'arrête si l'obstacle est lui-même immobile) ;

- si le deuxième véhicule est trop proche pour être doublé sans risque, le robot ralentit ou recule pour augmenter l'espace le séparant de cet obstacle ;

- lors d'une manœuvre de dépassement, si un autre obstacle est détecté, le robot tente soit de se rabattre rapidement si le dépassement est suffisamment avancé, soit de ralentir avant de reprendre sa place derrière le premier véhicule.

Nous pouvons noter que la mauvaise précision des capteurs utilisés impose l'emploi de distances de sécurité importantes lors des mouvements.
Nous présentons dans la section suivante quelques exemples de résultats obtenus.

5.3.2 Comportement obtenu

Durant les expériences réalisées nous avons utilisé un pas de discrétisation $\Delta t = 10\ ms$ pour le calcul de l'activation des différentes unités du réseau.
La figure 5.2 montre un exemple de dépassement de deux véhicules roulant à la vitesse de 2 $m.s^{-1}$. La vitesse associée à la trajectoire de référence est de 3 $m.s^{-1}$. Dans la première partie du mouvement le robot ne peut effectuer sa manœuvre en raison des obstacles entourant la route et se contente de suivre le véhicule se trouvant devant lui. Il est ensuite libre de dépasser par la gauche et effectue cette manœuvre à 5 $m.s^{-1}$.

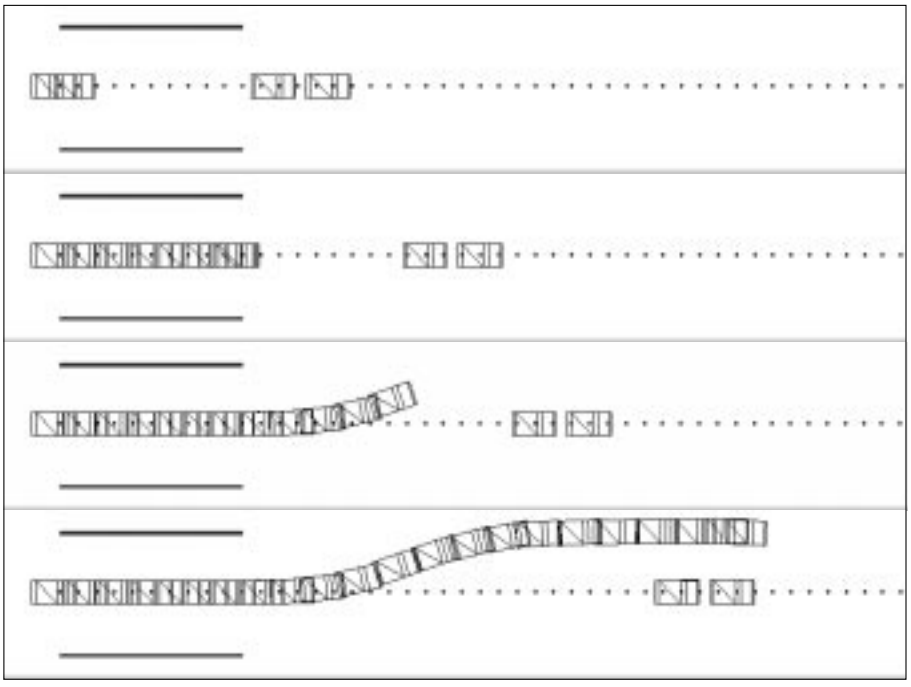

FIG. 5.2 – *Un exemple de dépassement.*

La figure 5.3 montre le comportement obtenu dans le cas où le robot rencontre un nouvel obstacle lors du dépassement. La manœuvre est suffisamment avancée lors de la

détection de l'obstacle pour pouvoir être menée à son terme. Le véhicule doit cependant effectuer un rabattement rapide.

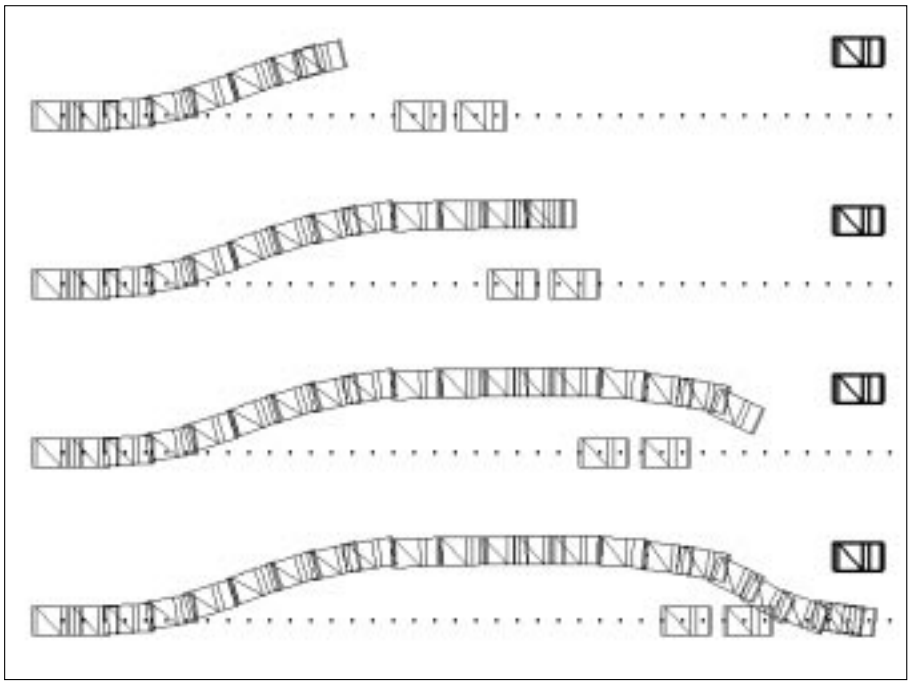

FIG. 5.3 – *Un exemple de dépassement avec rabattement rapide.*

La figure 5.4 présente l'évolution des sorties du réseau durant cette manœuvre. Le temps en secondes est porté en abscisse. Afin de privilégier la clarté de la figure, seules les valeur des quatre unités de sortie, qui sont à un moment donné les plus activées du réseau, sont représentés. Ces unités correspondent aux décision suivantes : suivi normal de trajectoire, dépassement par la gauche à 5 $m.s^{-1}$, arrêt et rabattement rapide sur la droite.
Le réseau préconise un suivi de trajectoire jusqu'à ce que les véhicules se trouvant sur la même voie que le robot soient détectés (après environ 0.8 secondes). L'unité de sortie correspondant au dépassement prend alors le dessus pendant environ 7 secondes. Lorsque le troisième véhicule obstacle est repéré, le robot n'a pas encore fini son dépassement (i.e. il détecte toujours des obstacles sur sa droite), le réseau demande donc un arrêt (période entre 7.8 et 8.1 secondes). Emporté par sa vitesse, le robot termine toutefois son dépassement avant d'avoir pu effectuer cet arrêt, et puisque sa vitesse est toujours suffisante et que l'espace le séparant du véhicule en face de lui est assez important, le réseau demande un rabattement rapide (période entre 8.1 et 8.8 secondes). Enfin dès que la zone en face du robot est libre de tout obstacle, le réseau demande un rattrapage de la trajectoire de référence et un suivi normal (après 8.8 secondes).
Dans le cas présenté par la figure 5.5, la manœuvre n'est pas suffisamment avancée lors de la détection de l'obstacle pour pouvoir être achevée. Le robot stoppe et recule légèrement jusqu'à ce qu'il puisse reprendre sa trajectoire de référence. L'arrêt est plus

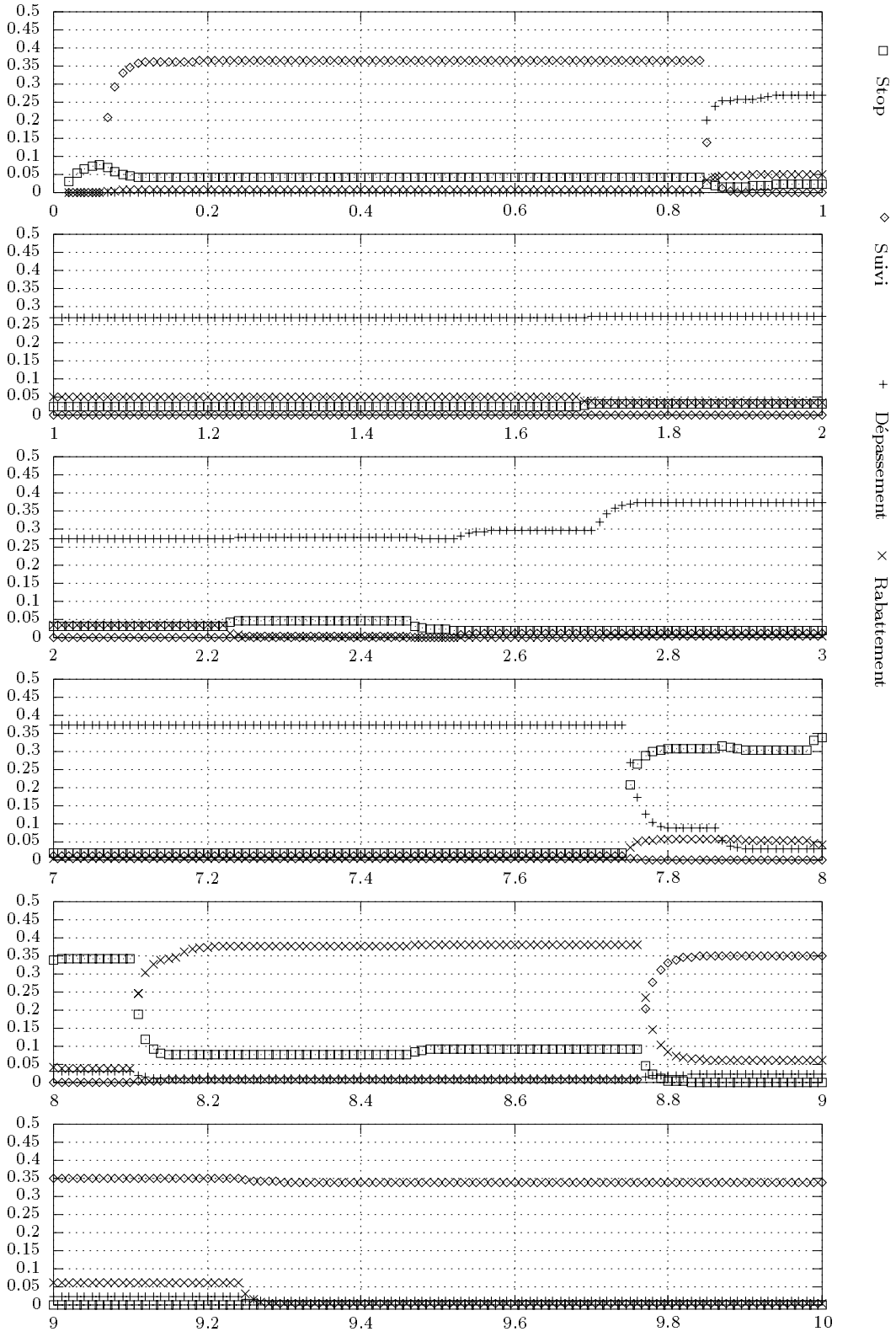

FIG. 5.4 – *Valeur des unités de sortie du réseau durant une manœuvre. Commentaires dans le texte.*

long que nécessaire car il faut attendre que les échos associés aux véhicules dépassés disparaissent de la carte locale, ce qui ne se produit que plusieurs secondes après la libération de la voie. Ceci peut poser des problèmes dans le cas ou l'obstacle rencontré durant le dépassement se déplace vers le robot. Ce dernier va alors reculer alors que rien ne l'empêche, théoriquement, de se rabattre. Une solution à ce problème pourrait passer par une amélioration de la gestion de la carte locale incluant une estimation des mouvements des autres véhicules et une prédiction de leurs positions futures.

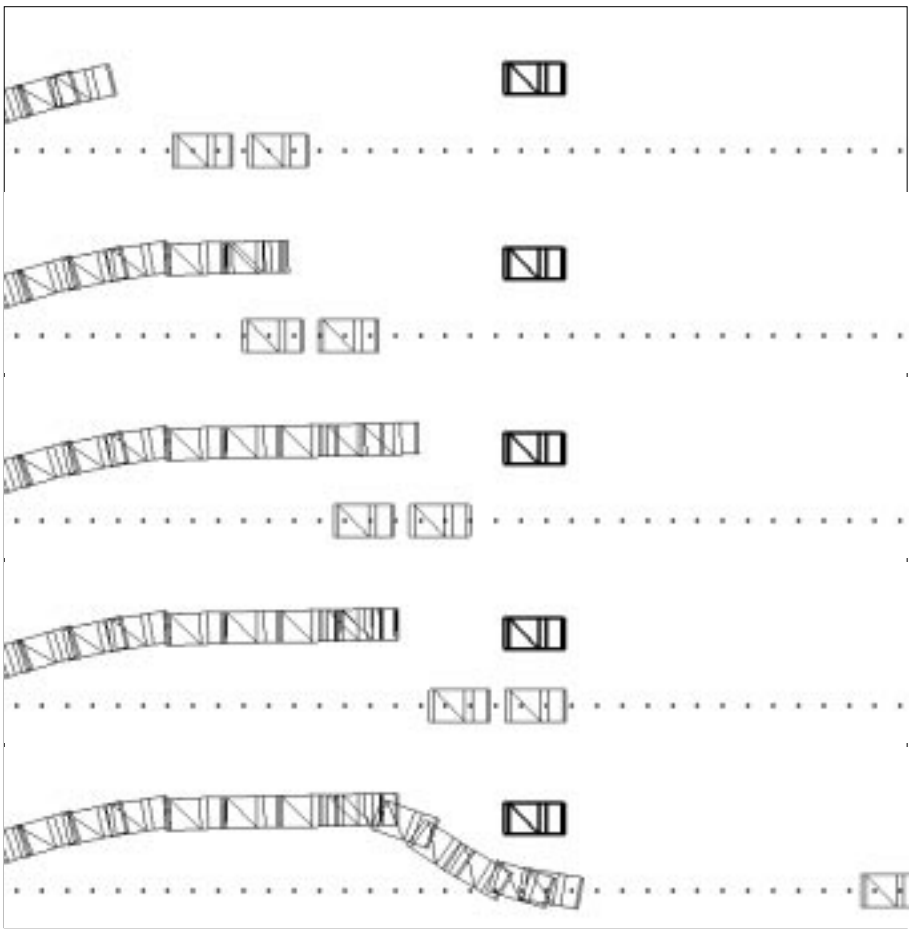

FIG. 5.5 – *Un exemple de dépassement avorté.*

5.4 Conclusion

Lors de ce chapitre nous avons présenté un exemple d'utilisation d'un modèle de réseau de neurones récurrent, pour la réalisation d'un système de sélection des contrôleurs impliqués dans une manœuvre complexe. L'architecture particulière de ce réseau permet de coder un ensemble de règles symboliques classiques ou les règles de transition d'un automate. Son fonctionnement permet de conserver la sémantique de ces règles tout en

garantissant des propriétés de rapidité d'exécution et de résistance face à la défaillance de certaines données d'entrée.

Il nous semble cependant que l'intérêt de notre approche n'apparaîtrait pleinement qu'en cas de réalisation «hardware» sur circuit imprimé du réseau. Une telle réalisation ne pose toutefois aucune difficulté théorique particulière vu la simplicité des équations régissant l'activation des unités.

La manœuvre que nous avons présentée ne constitue qu'un exemple des propriétés du réseau, et nous n'avons pas abordé dans ce chapitre le problème de la vérification de la complétude des règles, qui nous semble très important dans une application de ce type et ne doit pas être oublié.

Conclusion et perspectives générales

Au cours de cette thèse nous nous sommes intéressés aux possibilités offertes par les réseaux de neurones artificiels dans le cadre de la commande d'une voiture autonome.

Les robots de ce type posent, comme nous l'avons vu, des problèmes particuliers. Leur taille, leur poids, leur vitesse mais aussi leurs propriétés cinématiques impliquent en effet des contraintes spécifiques. Afin de minimiser les risques liés à ces caractéristiques, les systèmes de commande réalisés doivent notamment permettre de spécifier à l'avance, de manière très précise, les mouvements attendus. Ils doivent cependant conserver un maximum de flexibilité et permettre une réaction et une adaptation la plus rapide possible face aux événements extérieurs.

Une première étude des systèmes de commande pour robots mobiles proposés dans la littérature nous a permis de définir les différents niveaux fonctionnels qui doivent à notre avis se retrouver dans toute architecture de commande offrant une véritable autonomie à un véhicule :

- le plus bas niveau regroupe l'ensemble des systèmes ayant en charge la commande directe du véhicule, chaque système étant spécialisé dans une tâche ou une manœuvre particulière ;
- le niveau «contrôle d'exécution» a pour rôle de permettre le bon déroulement de la mission en choisissant et en instanciant les contrôleurs du niveau précédent ;
- le niveau planification est chargé de produire les plans de missions de haut niveau qui devront être suivis par le robot.

Les traitements effectués par les systèmes de planification étant principalement symboliques et donc peu adaptés aux RNA, c'est exclusivement sur les deux premiers niveaux qu'ont porté les travaux réalisés durant cette thèse. Les approches basées sur l'utilisation des réseaux de neurones artificiels sont en effet particulièrement bien adaptées aux problèmes qui s'y posent, grâce à leur robustesse, leur capacité de traiter des données bruitées ou incomplètes et surtout leurs capacités d'adaptation et d'apprentissage.

Nous avons donc proposé dans cette thèse plusieurs systèmes de commande permettant d'implémenter ces deux premiers niveaux et d'accroître progressivement l'autonomie du véhicule.

Le premier d'entre eux concerne le simple suivi d'une trajectoire issue d'un planificateur. L'approche que nous proposons permet au contrôleur de s'adapter en permanence aux changements des paramètres du véhicule ou de son environnement. Nous avons en effet cherché à tirer parti des possibilités d'apprentissage des RNA pour modéliser les réponses du robot aux commandes qui lui sont appliquées. De manière plus précise nous utilisons un réseau pour construire la fonction donnant à tout moment les commandes à appliquer pour obtenir le mouvement désiré. L'apprentissage de cette fonction est réalisé en continu, durant l'utilisation du contrôleur, à partir de l'observation des couples {mouvement mesuré, commande ayant provoquée ce mouvement}.

Dans un deuxième temps, nous nous sommes intéressés à des systèmes de commande offrant plus d'autonomie au robot en lui permettant de «choisir» ses mouvements en se basant sur ses données perceptives. Les RNA sont en effet un moyen simple et efficace de réaliser la fonction *commande=f(perception)* qui est à la base d'un contrôleur de ce type. Cette fonction peut notamment être obtenue par imitation du comportement d'un pilote humain.
Nous avons vu cependant, que notre robot pose des problèmes spécifiques liés notamment à sa taille et à sa vitesse, rendant très difficile la multiplication des expériences d'apprentissage ou de collecte de données. Cette limitation du nombre d'exemples disponibles pour réaliser l'apprentissage conduit généralement, lorsque des modèles classiques de réseaux sont utilisés, à des systèmes mémorisant «par cœur» les exemples, et qui se montrent incapables de généraliser ces données. La solution que nous avons cherché à appliquer consiste à utiliser les connaissances *a priori*, disponibles sur la tâche à réaliser, pour «initialiser» le réseau. Nous nous sommes plus particulièrement tournés vers les modèles hybrides neuro-flous qui permettent de fixer la structure du réseau à partir de connaissances exprimées sous forme de règles floues. L'apprentissage se réduit alors à un réglage fin des paramètres du système.
Nous avons donc proposé un modèle particulier de réseau neuro-flou adapté à notre application. Notre approche permet de plus, d'obtenir plusieurs jeux de paramètres pour le réseau, chaque jeu donnant une instanciation particulière du système adaptée à des conditions d'utilisation spécifiques. Nous présentons les résultats obtenus sur une manœuvre de dépassement d'un autre véhicule.

Ces deux types de contrôleurs correspondent à des éléments constitutifs du premier niveau d'une architecture de commande que nous avons évoqué ci-dessus. Cette thèse aborde également les apports possibles des RNA à la réalisation du deuxième niveau. Nous avons en effet montré comment un modèle de réseau récurrent, développé dans l'équipe Réseaux d'Automates du LEIBNIZ, pouvait être utilisé pour coder les règles de décision donnant les différents systèmes de commande à sélectionner en fonction de

la situation. Notre approche permet de conserver le sens de ces règles, qui sont mises en compétition dans le réseau, tout en gagnant des propriétés de robustesse et de rapidité d'exécution.

Les travaux présentés dans ce mémoire comportent de nombreux aspects qui n'ont pu être approfondis autant que nous l'aurions désiré. Si les systèmes de commande proposés démontrent à notre avis l'intérêt des réseaux de neurones artificiels et des systèmes hybrides neuro-symboliques dans le domaine de la robotique mobile, et plus particulièrement dans celui de la commande d'une voiture autonome, cette thèse ne constitue qu'une première étude qui demande à être poursuivie.
Les systèmes de commande proposés durant cette thèse ne pourront être réellement validés qu'une fois testés sur le véhicule expérimental plutôt que sur simulateur. Les premiers résultats de ces tests ainsi que le niveau de réalisme du simulateur utilisé nous rendent cependant optimistes à ce sujet.
Le deuxième point qui nous semble le plus important à approfondir concerne l'étude théorique de la stabilité des contrôleurs réalisés, qui n'a quasiment pas été abordée dans ce mémoire. Il s'agit d'un des points faibles traditionnels des RNA dans le domaine de la commande et donc d'une des voies de recherche où le plus de choses restent à faire.
Enfin plusieurs autres aspects nous paraissent importants à développer dans le futur. Il s'agit en particulier de la modélisation locale de l'environnement que nous avons abordé sans réellement l'approfondir, des possibilités d'intégration de nouveaux capteurs extéroceptifs plus riches tels que la vision, ou encore des méthodes de localisation du robot.

Annexe A

Présentation et modélisation du véhicule expérimental

A.1 Présentation

Le véhicule expérimental dont nous disposons est un véhicule de type «voiture sans permis» modifié (voir figure A.1). Cette voiture de marque Ligier équipée à l'origine d'un moteur thermique, a été transformée en véhicule électrique. Elle peut être conduite aussi bien en mode manuel comme un véhicule normal, qu'en mode automatique. Pour ce dernier mode, le véhicule a été équipé par la société ALEPH Technologies, de manière à pouvoir commander à partir d'un système informatique, le système de direction, la puissance moteur et le système de freinage.

FIG. A.1 – *Notre véhicule expérimental.*

Ce véhicule a également été équipé de plusieurs capteurs permettant d'avoir accès à des informations sur son environnement et sur son état interne. Il s'agit principalement

pour les capteurs extéroceptifs, de 14 capteurs ultrasons de la série 9000 de Polaroid (nous abordons dans le chapitre 4 les qualités et les défauts spécifiques de ce dispositif). Le véhicule sera de plus prochainement équipé d'une caméra linéaire permettant de repérer de façon très précise des balises disposées dans son environnement d'évolution. En complément de ces capteurs, l'utilisation d'un dispositif d'odométrie permettant d'estimer les déplacements du véhicule à partir du nombre de tours effectués par les roues arrières, autorise une localisation grossière. Cette méthode permet bien sûr d'obtenir la distance parcourue mais aussi les variations d'orientation du véhicule en utilisant la différence entre les distances parcourues par les roues gauche et droite. L'odométrie donne ainsi des résultats très précis pour de petits mouvements mais souffre d'une dérive qui augmente au fur et à mesure du déplacement.

A.2 Modélisation du véhicule

Nous nous intéressons dans ce qui suit au modèle de notre véhicule expérimental. Ce modèle s'applique de manière plus générale à tout véhicule de type voiture. Ce paragraphe s'inspire de [41].

A.2.1 Modèle géométrique

L'environnement dans lequel évolue le véhicule peut être assimilé au plan $\Re^2$. On associe à cet environnement un repère fixe $(O, \vec{i}, \vec{j})$ que nous désignerons comme le repère global. Le robot, noté $\mathcal{M}$, est représenté du point de vue géométrique par un rectangle comme indiqué sur la figure A.2.

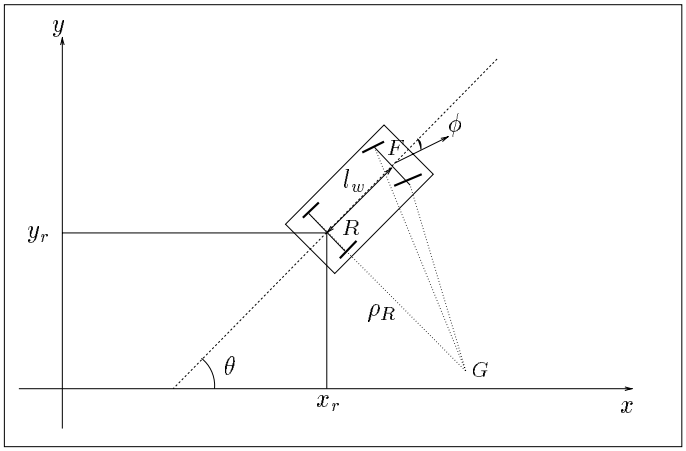

FIG. A.2 – *Le modèle géométrique du véhicule.*

Nous choisissons un point de référence fixe sur le robot. Ce point peut être par exemple le centre de l'essieu arrière, noté R. La configuration q du robot peut alors être décrite de manière unique par le triplet (x_R, y_R, θ), où (x_R, y_R) représentent les coordonnées

du point R dans le repère global et θ est l'angle de l'axe longitudinal du robot dans ce même repère. Dans le cas d'un mouvement de rotation, les axes des quatre roues se croisent en un point G qui est le centre de rotation du véhicule. On appelle angle de braquage, l'angle noté ϕ entre l'axe principal du véhicule et la perpendiculaire à la droite passant par G et par le centre de l'essieu avant, noté F.
La distance entre R et G représente le rayon de giration instantané ρ_R de R :

$$\rho_R = \frac{l_w}{tan(\phi)}, \tag{A.1}$$

où l_w est l'empatement du véhicule (distance entre R et F).
L'angle de braquage de $\mathcal{M}$ est borné pour des raisons mécaniques :

$$|\phi| < \phi_{max} \tag{A.2}$$

Ces butées se traduisent par l'existence d'un rayon de giration minimum pour R :

$$\rho_{Rmin} = \frac{l_w}{tan(\phi_{max})} \tag{A.3}$$

Il est de même possible d'exprimer le rayon de giration minimum du point F :

$$\rho_{Fmin} = \frac{l_w}{sin(\phi_{max})} \tag{A.4}$$

Nous pouvons noter que ces rayons minimum ne peuvent pas toujours être atteints par le véhicule à forte vitesse. Ceci est dû à la force centrifuge qui entraînerait alors un glissement des roues.

A.2.2 Modèle cinématique

Les équations du mouvement pour le point de référence R, dans le repère global sont les suivantes :

$$\dot{x_R} = v_R \cos(\theta) \tag{A.5}$$
$$\dot{y_R} = v_R \sin(\theta) \tag{A.6}$$
$$\dot{\theta} = \frac{v_R}{\rho_R} = v_R \frac{\tan(\phi)}{l_w}, \tag{A.7}$$

où v_R est la vitesse instantanée du point R, et la notation $\dot{x_R}$ correspond à la dérivé par rapport au temps de x_R.
Le mouvement du centre de l'axe des roues avant, F est donné par les équations :

$$\dot{x_F} = v_F \cos(\theta + \phi) \tag{A.8}$$
$$\dot{y_F} = v_F \sin(\theta + \phi) \tag{A.9}$$
$$\dot{\theta} = v_F \frac{\sin(\phi)}{l_w} \tag{A.10}$$

Les équations A.5 et A.6 nous permettent d'obtenir :

$$\dot{x_R}\sin(\theta) + \dot{y_R}\cos(\theta) = 0 \tag{A.11}$$

Cette équation constitue une contrainte *non holonome*[1]. Ce type de contrainte restreint l'ensemble des vecteurs vitesse que le véhicule peut atteindre. Elle exprime le fait que le vecteur vitesse instantané du point R est forcement colinéaire à l'axe longitudinal de $\mathcal{M}$. Le cas contraire entraînerait un glissement des roues. On parle donc de *contrainte de non-glissement des roues* pour la qualifier.
En choisissant les équations A.8 et A.9 nous obtenons la contrainte équivalente pour le point F :

$$\dot{x_F}\sin(\theta+\phi) + \dot{y_F}\cos(\theta+\phi) = 0 \tag{A.12}$$

Les équations A.7 et A.3 permettent d'exprimer une deuxième contrainte non holonome limitant les valeurs possibles de la dérivée $\dot{q}$ de la configuration du robot :

$$|\dot{\theta}| \leq \frac{|v|}{\rho_{min}} \tag{A.13}$$

Ces deux contraintes sont très importantes, elles expriment le fait intuitif qu'une voiture ne peut se déplacer dans toute les directions. Ceci rend la commande d'un tel véhicule plus complexe mais ne limite en aucun cas l'ensemble des configurations qui peuvent être atteintes. Il a été en effet démontré par J. P. Laumond [91] que tout chemin reliant deux configurations quelconques pouvait être transformé en chemin faisable[2] pour un véhicule de type voiture, au prix d'un nombre fini de manoeuvres. On parle pour traduire ce résultat de la *commandabilité complète* d'un robot de type voiture.

1. Une contrainte non holonome est une contrainte non intégrable faisant intervenir la dérivée par rapport au temps des coordonnées du robot [89].

2. Chemin que le véhicule peut suivre.

Annexe B

Simulation d'un véhicule mobile

Afin de valider notre travail, il s'est avéré nécessaire de disposer d'un simulateur reproduisant le plus fidèlement possible le comportement du véhicule et de ses interactions avec l'environnement. Celui ci à été développé en langage C++, en parallèle sur station de travail Silicon Graphics et sur compatible PC dans l'environnement Windows 95. Des efforts tout particuliers ont porté sur la réalisation d'une interface simple et conviviale, tant du point de vue de la présentation graphique des résultats que de celui des possibilités de programmation par l'utilisateur d'une situation particulière.

B.1 Interface utilisateur

La partie graphique du simulateur s'appuie sur la bibliothèque de visualisation en 3 dimensions OpenGL. Elle permet de choisir le mode de visualisation, le point de vue ainsi que le niveau de détail du rendu. La figure B.1 présente quelques copies d'écrans du simulateur correspondant à différents types d'affichage. La description des situations de simulation (e.g. position des véhicules, lois de commande utilisées, position et type des obstacles, trajectoires de référence,...) peut se faire soit à l'aide d'un langage simple, soit à l'aide des menus du simulateur. Durant la conception de ce simulateur nous avons tenté de tirer parti de l'approche orientée objet pour privilégier les possibilités d'extension futures. Il est ainsi possible, par exemple, de rajouter de nouvelles lois de commande ou de nouveaux types d'obstacles en dérivant respectivement les classes génériques contrôleur ou obstacle.

B.2 Simulation des mouvements

La simulation des véhicules utilise le modèle cinématique que nous avons présenté en annexe A. Les aspects dynamiques intégrés dans la simulation sont assez restreints. Seuls les risques de glissement lorsque la force centrifuge devient trop importante, ainsi que les bornes des accélérations $\dot{v}$ et $\ddot{\phi}$ dont les véhicules sont capables, sont pris en

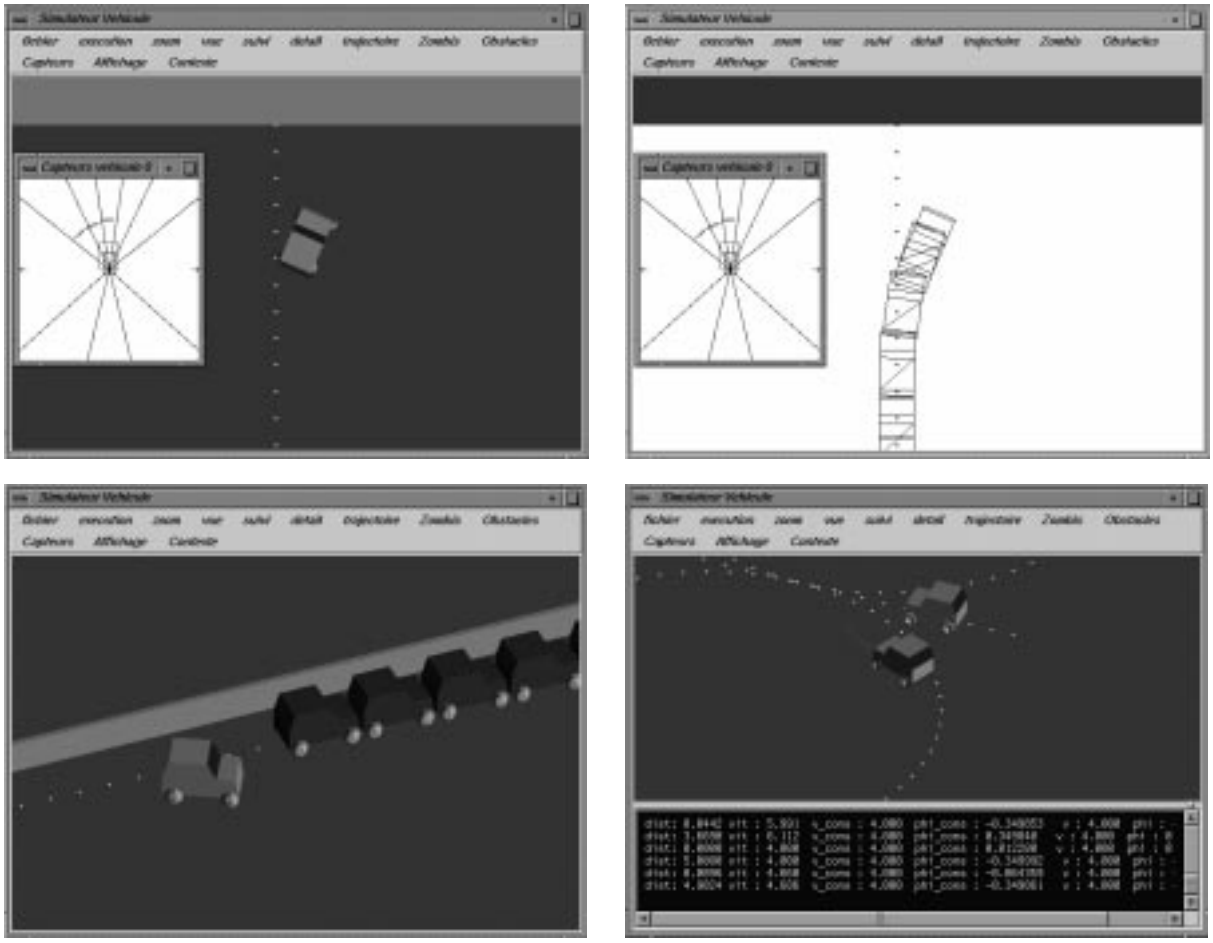

FIG. B.1 – *Quelques copies d'écrans de notre simulateur.*

compte. Ces dernières valeurs, tout comme les bornes sur la vitesse, ou l'angle de braquage, ont été choisies les plus proches possible de celles mesurées sur le véhicule expérimental.
Le simulateur autorise l'introduction de perturbations du fonctionnement du véhicule, permettant de tester la robustesse des lois de commande programmées.

B.3 Simulation de la perception

La reproduction exacte des résultats fournis par des capteurs ultrasonores, en un temps compatible avec une simulation en temps réel, est à notre avis impossible.
Le fonctionnement de ces capteurs dépend en effet d'un nombre très important de facteurs comme la taille, l'orientation ou la matière composant la surface des obstacles. Notre but n'est donc pas de réaliser un simulateur suffisamment réaliste pour autoriser l'entraînement d'un système de commande destiné à une utilisation directe sur le robot. Nous cherchons plutôt à obtenir un démonstrateur permettant de tester la faisabilité

d'une approche ou d'un type d'apprentissage particulier.
Nous nous sommes donc attachés à créer des capteurs simulés délivrant une information qui ne soit pas plus riche que celle disponible sur le robot, même si ces deux informations ne sont pas exactement similaires.
Notre simulation prend donc en compte les facteurs suivants pour déterminer si un obstacle est détecté ou non :

- l'orientation de la surface de l'obstacle par rapport à l'axe principal du cône d'émission ;
- la nature de l'obstacle. Celle ci est prise en compte à travers la définition d'un coefficient de détectabilité ;
- la distance entre l'obstacle et le capteur.

Ces différentes données nous permettent de définir une probabilité de détection. La décision finale quand à la détection de l'obstacle fait donc intervenir un tirage aléatoire.

Annexe C

Grammaire du langage de description neuro-flou

C.1 La notation BNF

La description et la sauvegarde des réseaux neuro-flous que nous utilisons se fait par l'intermédiaire d'un langage qui respecte la grammaire non contextuelle présentée ci-dessous.
Nous utilisons la notation BNF [1] largement adoptée dans le domaine des compilateurs.
Les unités lexicales aussi appelées symboles terminaux (signes et mots clés du langage) sont représentés entourées de guillemets.
Les symboles en italiques correspondent à des suites d'unités lexicales et sont appelées non-terminaux.
Les règles de production suivent une syntaxe du type A $\rightarrow$ B, où la flèche signifie que le symbole A peut être remplacé par le symbole B.
Le signe | permet de spécifier plusieurs possibilités pour la partie droite de ces règles. Ainsi la règle A $\rightarrow$ B | C peut aussi s'écrire en utilisant les deux règles : A $\rightarrow$ B et A $\rightarrow$ C.
Le signe ε est utilisé pour indiquer une liste de symbole vide dans une règle de production.
Enfin dans ce qui suit les signes /* et */ entourent des commentaires.

C.2 Règles de production

fichier_réseau $\rightarrow$ *decl_variables sep partie_ensembles sep partie_règles*

/* déclaration des variables d'entrée et de sortie */
decl_variables $\rightarrow$ *decl_entrée liste_noms_entrée decl_sortie liste_noms_sortie*

1. Acronyme de Backus-Naur Form.

/* nombre d'entrées et nombre de sorties */
decl_sortie → *n_sorties* « : » *valeur*
decl_entrée → *n_entrées* « : » *valeur*

/* Déclaration du nom des variables d'entrée */
liste_noms_entrée → *nom_entrée liste_noms_entrée* | ε
nom_entrée → *chaîne*
/* La valeur 1 passée en paramètre permet de définir une variable de type angle[1]
La valeur 2 permet de définir une variable possédant deux dimensions d'entrée*/
nom_entrée → *chaîne* « [» *valeur* «] »
/* Permet de définir les valeurs minimum et maximum de la variable */
nom_entrée → *chaîne* « [» *valeur valeur* «] »
/* Idem dans le cas d'une variable à deux dimensions */
nom_entrée → *chaîne* « [» *valeur valeur* «] » « [» *valeur valeur* «] »

/* Déclaration du nom des variables de sortie */
liste_noms_sortie → *nom_sortie liste_noms_sortie* | ε
nom_sortie → *chaîne*
/* Permet de définir les valeurs minimum et maximum de la variable */
nom_sortie → *chaîne* « [» *valeur valeur* «] »

/* Déclarations des sous-ensembles flous */
partie_ensembles → *decl_ensembles liste_ensembles*
/* Nombre de sous-ensembles flous */
decl_ensembles → *n_ensembles* « : » *valeur*
liste_ensembles → *liste_ensembles ensemble* | ε
/* Les 3 paramètres correspondent à la variance gauche au centre puis à la variance droite */
ensemble → *variable qualificatif* « : » *paramètres paramètres paramètres*
variable → *identificateur*
/* Nom du sous-ensemble flou */
qualificatif → *identificateur*

/* Il est possible de spécifier plusieurs paramètres correspondant à des contextes différents */
paramètres → « [» *liste_param* «] »
liste_param → *liste_param valeur* | *valeur*

/* Déclaration des règles */
partie_règles → *decl_règles liste_règles*
/* Nombre de règles */
decl_règles → *n_règles* « : » *valeur*
liste_règles → *liste_règles règle* | *règle*

/* Les propositions floues doivent précéder la partie sous règles */
règle → *deb_r liste_prémisses partie_sous_règles*
implication conclusions partie_poids

/* Il est possible de spécifier plusieurs poids correspondant à des contextes différents */
partie_poids → « [» *liste_poids* «] »
liste_poids → *liste_poids valeur* | *valeur*

/* liste de proposition floues de la partie prémisse */
liste_prémisses → *prémisse*
liste_prémisses → *liste_prémisses conjonction prémisse*
liste_prémisses → *liste_prémisses disjonction prémisse*
/* Déclaration d'une proposition floue */
prémisse → *variable est qualificatif* | *variable est négation qualificatif*

/* Déclaration des sous-règles d'une règle*/
partie_sous_règles → *disjonction liste_sous_règles* | *conjonction liste_sous_règles* | ε
liste_sous_règles → *sous_règle*
liste_sous_règles → *liste_sous_règles conjonction sous_règle*
liste_sous_règles → *liste_sous_règles disjonction sous_règle*

/* Un sous règle est entourée de parenthèses */
sous_règle → « (» *prémisse partie_sous_règles* «) »

/* Déclaration des conclusions d'une règles */
/* Une règle peut avoir plusieurs conclusions */
conclusions → *conclusions conjonction conclusion* | *conclusion*
conclusion → *variable est paramètres*

/* Un identificateur peut contenir des chiffres mais commence forcement par une lettre */
identificateur → *lettre liste_car_opt*
liste_car_opt → *liste_car* | ε
liste_car → *liste_car car* | *car*
car → *lettre* | → *chiffre*
valeur → *signe_opt liste_chiffre virgule_opt liste_chiffre_opt*
signe_opt → « - » | ε
virgule_opt → « . » | ε
liste_chiffre_opt → *liste_chiffres* | ε
liste_chiffres → *chiffre liste_chiffres* | ε

C.3 Mots clés du langage

n_entrées → «nb_entrees»
n_entrées → «nb_input»
n_sorties → «nb_sorties»
n_sorties → «nb_output»
n_ensembles → «nb_ensembles»
n_ensembles → «nb_set»
n_règles → «nb_regles»
n_règles → «nb_rules»
est → «est»|«EST»|«is»|«IS»
deb_r → «si»|«SI»|«if»|«IF»
conjonction → «et»|«ET»|«and»|«AND»
disjonction → «or»|«OU»|«or»|«OR»
négation → «non»|«NON»|«not»|«NOT»
implication → «alors»|«ALORS»|«then»|«THEN»
sep → «=====»
lettre → «a»|«b»|«c» .....|«z»|«A»|«B»|«C» .....|«Z»
chiffre → «0»|«1»|«2»|«3»|«4»|«5»|«6»|«7»|«8»|«9»

Les lignes commençant par le signe # ne sont pas prises en compte (commentaires).

C.4 Fonctions de la bibliothèque

Le type NF_reseau correspond à une structure contenant la description d'un réseau neuro-flou.

*NF_reseau *NF_lit_reseau(char *fichier_reseau)*
Lecture d'un réseau.
Paramètres :
fichier_reseau : nom du fichier de description du réseau.
Retour : pointeur vers la structure réseau neuro-flou.

*int NF_sauve_reseau(NF_reseau *res, char *fichier_reseau)*
Sauvegarde d'un réseau.
Paramètres :
res : pointeur vers la structure réseau ;
fichier_reseau : nom du fichier de sauvegarde.
Retour : 1 si la sauvegarde s'est déroulée normalement 0 sinon.

*int NF_libere_reseau(NF_reseau *res)*
Libère la mémoire utilisée par le réseau.
Paramètres :
res : pointeur vers la structure réseau.
Retour : 1 si l'opération s'est déroulée correctement 0 sinon.

*int NF_fixe_entree(NF_reseau *res, char *nom, float valeur)*
Spécification de la valeur d'une variable d'entrée.
Paramètres :
res : pointeur vers la structure réseau ;
nom : nom de la variable d'entrée ;
valeur : valeur de la variable d'entrée.
Retour : 1 si l'opération s'est déroulée correctement 0 sinon.

*int NF_fixe_entree2(NF_reseau *res, char *nom, float valeur1, float valeur2)*
Spécification de la valeur d'une variable d'entrée à deux dimensions.
Paramètres :
res : pointeur vers la structure réseau ;
nom : nom de la variable d'entrée ;
valeur1 : valeur de la première dimension de la variable d'entrée ;
valeur2 : valeur de la deuxième dimension de la variable d'entrée.
Retour : 1 si l'opération s'est déroulée correctement 0 sinon.

*int NF_active_reseau(NF_reseau *res, int contexte)*
Activation du réseau.
Paramètres :
res : pointeur vers la structure réseau ;
contexte : numéro du jeu de paramètres à utiliser.
Retour : 1 si l'opération s'est déroulée correctement 0 sinon.

*float NF_lit_sortie(NF_reseau *res, char *nom)*
Lecture de la valeur d'une sortie.
Paramètres :
res : pointeur vers la structure réseau ;
nom : nom de la variable de sortie.
Retour : valeur de la sortie.

int NF_fixe_parametres(float eta1, float eta2, float eta3, float mu)
Fixe les paramètres de l'apprentissage.
Paramètres :
res : pointeur vers la structure réseau ;
eta1 : valeur du pas du gradient pour les poids des règles ;
eta2 : valeur du pas du gradient pour les sorties des règles ;
eta3 : valeur du pas du gradient pour les sous-ensembles flous ;
mu : valeur du momentum.
Retour : 1 si l'opération s'est déroulée correctement 0 sinon.

*void NF_parametres(float *eta_1, float *eta_2, float *eta_3, float *mu)*
Lecture des paramètres de l'apprentissage.
Paramètres :
res : pointeur vers la structure réseau ;
eta1 : valeur du pas du gradient pour les poids des règles ;
eta2 : valeur du pas du gradient pour les sorties des règles ;
eta3 : valeur du pas du gradient pour les sous-ensembles flous ;
mu : valeur du momentum.
Retour :

*int NF_apprentissage(NF_reseau *res, float *des, int sorties, int poids, int ensembles, int contexte)*
Réalise un pas d'apprentissage.
NF_active_réseau doit avoir été appelé auparavant.
Paramètres :
res : pointeur vers la structure réseau ;
des : pointeur vers la zone mémoire contenant les sorties désirées ;
sorties : indique si l'apprentissage doit concerner les valeurs de sortie ;
poids : indique si l'apprentissage doit concerner les poids des règles ;
ensembles : indique si l'apprentissage doit concerner les sous-ensembles flous ;
contexte : numéro du jeu de paramètres à modifier.
Retour : 1 si l'opération s'est déroulée correctement 0 sinon.

*float NF_erreur(NF_reseau *res, float *des)*
Retourne la valeur de l'erreur quadratique.
NF_active_réseau doit avoir été appelé auparavant.
Paramètres :
res : pointeur vers la structure réseau ;
des : pointeur vers la zone mémoire contenant les sorties désirées.
Retour : valeur de l'erreur quadratique.

*int NF_init_app(NF_reseau *res)*
Initialise des structures de données nécessaires à l'apprentissage.
A réaliser une fois uniquement avant de commencer l'apprentissage.
Paramètres :
res : pointeur vers la structure réseau.
Retour : 1 si l'opération s'est déroulée correctement 0 sinon.

*int NF_libere_app(NF_reseau *res)*
Libère la mémoire occupée par les structures nécessaires pour l'apprentissage.
Paramètres :
res : pointeur vers la structure réseau.
Retour : 1 si l'opération s'est déroulée correctement 0 sinon.

Annexe D

Base de règles floues

D.1 Définition des sous-ensembles flous

Les différentes entrées associées aux capteurs virtuels sont nommées de la manière suivante[1] :

Zone 1	: *avant*		
Zone 2	: *avant_droite*	Zone 3	: *avant_gauche*
Zone 4	: *avant_droite1*	Zone 5	: *avant_gauche1*
Zone 6	: *droite*	Zone 7	: *gauche*
Zone 8	: *droite1*	Zone 9	: *gauche1*
Zone 10	: *arriere_droite*	Zone 11	: *arriere*
Zone 12	: *arriere_gauche*		

Les entrées *orientation_gauche* et *orientation_droite* correspondent à l'orientation du véhicule par rapport aux obstacles présents sur sa gauche et sur sa droite. L'orientation du but dans le repère lié au véhicule est donnée par *direction_but*.
Le tableau D.1 décrit les paramètres des sous-ensembles flous définis pour les entrées associées aux zones 1, 2, 3, 10, 11 et 12.
Le tableau D.2 décrit les paramètres des sous-ensembles flous définis pour les entrées associées aux zones 4, 5, 6, 7, 8 et 9.
Les tableaux D.3 et D.4 montrent les sous-ensembles associés respectivement aux variables d'orientation et de direction du but.

D.2 Règles d'évitement d'obstacle

si avant est proche alors direction est 1.00 [2.000]
si avant est moyen alors direction est 0.5 [2.000]

1. Les numéros correspondent à ceux de la figure 4.5.

nom	type	dimension 1			dimension 2		
		σ gauche	centre	σ droite	σ gauche	centre	σ droite
Proche	demi gaussienne	2.8	0	2.8	0.65	0	0.65
Moyen	gaussienne	2.8	3	2.8	0.65	0	0.65
Loin	gaussienne	2.8	6	2.8	0.65	0	0.65
Libre	demi gaussienne	2.8	9	2.8	0.65	0	0.65

TAB. D.1 – *Sous-ensembles flous associés aux zones 1, 2, 3, 10, 11 et 12.*

nom	type	dimension 1			dimension 2		
		σ gauche	centre	σ droite	σ gauche	centre	σ droite
Proche	demi gaussienne	1.4	0	1.4	0.65	0	0.65
Moyen	gaussienne	1.4	1.5	1.4	0.65	0	0.65
Loin	gaussienne	1.4	3	1.4	0.65	0	0.65
Libre	demi gaussienne	1.4	4.5	1.4	0.65	0	0.65

TAB. D.2 – *Sous-ensembles flous associés aux zones 4, 5, 6, 7, 8 et 9.*

si avant est loin alors direction est 0.3 [2.000]

si avant_droite est proche alors direction est 1.000 [2.000]
si avant_droite est moyen alors direction est 0.400 [2.000]
si avant_droite est loin alors direction est 0.200 [2.000]

si avant_gauche est proche alors direction est -1.00 [1.000]
si avant_gauche est moyen alors direction est -0.4 [1.000]

si avant_droite1 est proche alors direction est 1.0 [2.000]
si avant_droite1 est moyen alors direction est 0.3 [2.00]

si avant_gauche1 est proche alors direction est -1.0 [1.000]
si avant_gauche1 est moyen alors direction est -0.3 [1.000]

nom	type	σ gauche	centre	σ droite
Negative grand	demi gaussienne	0.2	-0.4	0.2
Negative petit	gaussienne	0.2	-0.2	0.2
Nulle	gaussienne	0.2	0	0.2
Positive petit	gaussienne	0.2	0.2	0.2
Positive grand	demi gaussienne	0.2	0.4	0.2

TAB. D.3 – *Sous-ensembles flous associés aux variables orientation_gauche et orientation_droite.*

nom	type	σ gauche	centre	σ droite
devant	gaussienne	0.500	0	0.500
devant_gauche	gaussienne	0.500	0.730	0.500
devant_droite	gaussienne	0.500	-0.730	0.500
gauche	gaussienne	0.500	1.570	0.500
droite	gaussienne	0.500	-1.570	0.500
derriere_gauche	gaussienne	0.500	2.300	0.500
derriere_droite	gaussienne	0.500	-2.300	0.500
derriere	gaussienne	0.500	Π	0.500

TAB. D.4 – *Sous-ensembles flous associés à la variable direction_but.*

si droite est proche alors direction est 1.0 [2.000]

si gauche est proche alors direction est -1.0 [1.000]

D.3 Règles d'alignement sur une file de véhicules

si orientation_droite est positif_petit et droite est moyen et avant_droite est libre et avant est libre alors direction est -0.200 [1.00]
si orientation_droite est positif_grand et droite est moyen et avant_droite est libre et avant est libre alors direction est -0.4000 [1.00]
si orientation_droite est nulle et droite est moyen et avant_droite est libre et avant est libre alors direction est 0.0000 [1.00]
si orientation_droite est negatif_petit et droite est moyen et avant_droite est libre et avant est libre alors direction est 0.200 [1.00]
si orientation_droite est negatif_grand et droite est moyen et avant_droite est libre et avant est libre alors direction est 0.400 [1.00]

D.4 Règles d'attraction pour le but

si dir_but est devant et avant est libre alors direction est 0.0000 [1.000]

si dir_but est devant_droite et droite est non proche et avant_droite1 est non proche et (avant_droite est libre ou avant_droite est loin) alors direction est -0.5 [1.000]

si dir_but est droite et avant_droite1 est non proche et droite est non proche et (avant_droite est libre ou avant_droite est loin) alors direction est -0.75 [1.000]

si dir_but est derriere_droite et (avant_droite est libre ou avant_droite est loin) et (avant_droite1 est libre ou avant_droite1 est loin) et (droite est libre ou droite est loin) alors direction est -1.0 [1.000]

si dir_but est derriere et (avant_droite est libre ou avant_droite est loin) et (avant_droite1 est libre ou avant_droite1 est loin) et (droite est libre ou droite est loin) alors direction est -1.0 [1.000]

Annexe E

Base de règles de sélection des contrôleurs

Nous présentons dans cette annexe les règles régissant le choix d'un système de commande que nous avons utilisées pour la manœuvre présentée dans le chapitre 5.

E.1 Entrées et sorties des règles

Le nombre de choix possibles parmi les différentes actions (i.e nombre de sorties des règles) est de 25. Ces choix sont les suivants (nous indiquons entre parenthèses les noms utilisés par la suite pour désigner chacune de ces situations) :

- suivi de trajectoire en respectant la vitesse associée à cette trajectoire (normal) ;
- suivi de trajectoire en adoptant la vitesse du véhicule précédant le robot (suivi) ;
- suivi de trajectoire à une vitesse inférieure à celle du véhicule précédent le robot (suivi lent) ;
- dépassement par la gauche, à différentes vitesses, d'un véhicule immobile (évitement gauche1, évitement gauche2, évitement gauche3, évitement gauche4, évitement gauche5, évitement gauche6). Ces 6 choix correspondent respectivement à des manœuvres effectuées à 0,5 $m.s^{-1}$, 1 $m.s^{-1}$, 2 $m.s^{-1}$, 3 $m.s^{-1}$, 4 $m.s^{-1}$ ou 5 $m.s^{-1}$;
- dépassement par la droite, à différentes vitesses, d'un véhicule immobile (évitement droite1, évitement droite2, évitement droite3, évitement droite4, évitement droite5, évitement droite6) ;
- dépassement par la gauche, à différentes vitesses, d'un véhicule en mouvement (dépassement gauche1, dépassement gauche2). Ces deux choix correspondent à des dépassements effectués à des vitesses de 4 $m.s^{-1}$ ou 5 $m.s^{-1}$;

- dépassement par la droite, à différentes vitesses, d'un véhicule en mouvement (dépassement droite1, dépassement droite2) ;
- rabattement rapide sur la gauche ou sur la droite (rabattement gauche, rabattement droite) ;
- arrêt normal (stop), arrêt d'urgence (urgence) ou recul (recul) ;
- appel au niveau supérieur (échec).

Pour déterminer le contrôleur à choisir, les règles utilisent 31 entrées différentes. Ces entrées sont les suivantes (nous donnons ici encore entre parenthèses, les noms désignant ces différentes entrées dans la suite de cette annexe) :

- 3 entrées indiquant respectivement si le robot se trouve sur sa trajectoire de référence ou à gauche ou à droite de cette trajectoire (voie normale, voie gauche et voie droite). Le robot est considéré comme étant sur sa trajectoire si il se trouve à une distance inférieure à 1.5 mètres de celle-ci ;
- 4 entrées indiquant si les zones situées à l'avant, à droite, à gauche ou à l'arrière du robot sont libres (voie libre, droite libre, gauche libre et arrière libre). Ces données sont obtenues à partir de la carte locale de l'environnement ;
- 5 entrées donnant la vitesse associée à la trajectoire de référence (vitesse1, vitesse2, vitesse3, vitesse4 et vitesse5). Ces entrées indiquent respectivement si la vitesse se trouve dans un des intervalles $[0; 1m.s^{-1}]$, $[1; 2m.s^{-1}]$, $[2; 3m.s^{-1}]$, $[3; 4m.s^{-1}]$, $[4; 5m.s^{-1}]$;
- 4 entrées donnant la vitesse du véhicule à dépasser (vitesse obstacle1, vitesse obstacle2, vitesse obstacle3, vitesse obstacle4). Ces entrées correspondent respectivement aux intervalles $[0; 0,5m.s^{-1}]$, $[0,5; 1,5m.s^{-1}]$, $[1,5; 2,5m.s^{-1}]$, $[2,5; \infty\, m.s^{-1}]$;
- 5 entrées donnant la distance entre le robot et le véhicule à dépasser (distance1, distance2, distance3, distance4 et distance5). Ces entrées ne sont plus mises à jour une fois la manœuvre de dépassement entamée ;
- 2 entrées indiquant si le dépassement par la gauche ou par la droite est autorisé (gauche possible et droite possible). Ces données dépendent de la présence d'obstacles mais peuvent également être utilisées pour forcer le robot à effectuer ou non une manœuvre à partir d'informations a priori ;
- 3 entrées indiquant qu'un obstacle se trouve à l'avant du robot à des distances respectivement inférieures à 4 m, 7,5 m et 9 m (obstacle proche1, obstacle proche2 et obstacle proche3) ;
- 1 entrée indiquant que les échos présents sur la carte locale ne présentent pas de risque de collision si le robot suit sa trajectoire de référence (trajectoire libre) ;

- 1 entrée indiquant qu'un obstacle se trouve dans la zone d'arrêt d'urgence (collision) ;
- 2 entrées indiquant si la vitesse du robot est positive et si elle est supérieure à celle du véhicule dépassé (vitesse positive et vitesse supérieure) ;
- 1 entrée indiquant que le robot est en cours de rabattement rapide (rabattement).

E.2 Règles de décision

Dépassement d'un véhicule immobile. Cas où la vitesse associée à la trajectoire de référence se situe entre 0 et 1 $m.s^{-1}$:

```
Si voie normale et non trajectoire libre et gauche possible et vitesse obstacle1
et vitesse1 et distance1 alors évitement gauche1
Si voie normale et non trajectoire libre et gauche possible et vitesse obstacle1
et vitesse1 et non distance0 et non distance1 alors évitement gauche2
Si voie gauche et non droite libre et voie libre et vitesse obstacle1 et vitesse1
et distance0 alors évitement gauche1
Si voie gauche et non droite libre et voie libre et vitesse obstacle1 et vitesse1
et non distance0 alors évitement gauche2
Si voie normale et non trajectoire libre et non gauche possible et droite possible
et vitesse obstacle1 et vitesse1 et distance1 alors évitement droite1
Si voie normale et non trajectoire libre et non gauche possible et droite possible
et vitesse obstacle1 et vitesse1 et non distance0 et non distance1
alors évitement droite2
Si voie droite et non gauche libre et voie libre et vitesse obstacle1
et vitesse1 et distance0 alors évitement droite1
Si voie droite et non gauche libre et voie libre et vitesse obstacle1
et vitesse1 et non distance0 alors évitement droite2
```

Dépassement d'un véhicule immobile. Cas où la vitesse associée à la trajectoire de référence se situe entre 4 et 5 $m.s^{-1}$[1]:

```
Si voie normale et non trajectoire libre et gauche possible et vitesse obstacle1
et vitesse5 et distance1  et arrière libre alors recul
Si voie normale et non trajectoire libre et gauche possible et vitesse obstacle1
et vitesse5 et distance1  et non arrière libre alors échec
Si voie normale et non trajectoire libre et gauche possible et vitesse obstacle1
et vitesse5 et distance2 alors évitement gauche2
Si voie normale et non trajectoire libre et gauche possible et vitesse obstacle1
et vitesse5 et distance3 alors évitement gauche4
Si voie normale et non trajectoire libre et gauche possible et vitesse obstacle1
et vitesse5 et distance4 alors évitement gauche6
Si voie gauche et non droite libre et voie libre et vitesse obstacle1
et vitesse5 et distance0 alors évitement gauche2
```

1. Pour des raisons de place et de clarté, nous ne donnons pas les règles correspondant à des vitesses de référence entre 1 et 4 $m.s^{-1}$. Leur principe est équivalent à celui des règles que nous présentons ici.

```
Si voie gauche et non droite libre et voie libre et vitesse obstacle1
et vitesse5 et distance1 alors évitement gauche2
Si voie gauche et non droite libre et voie libre et vitesse obstacle1
et vitesse5 et distance2 alors évitement gauche3
Si voie gauche et non droite libre et voie libre et vitesse obstacle1
et vitesse5 et distance3 alors évitement gauche4
Si voie gauche et non droite libre et voie libre et vitesse obstacle1
et vitesse5 et distance4 alors évitement gauche6
Si voie normale et non trajectoire libre et non gauche possible et droite possible
et vitesse obstacle1 et vitesse5 et distance1 et arrière libre alors recul
Si voie normale et non trajectoire libre et non gauche possible et droite possible
et vitesse obstacle1 et vitesse5 et distance1 et non arrière libre alors échec
Si voie normale et non trajectoire libre et non gauche possible et droite possible
et vitesse obstacle1 et vitesse5 et distance2 alors évitement droite2
Si voie normale et non trajectoire libre et non gauche possible et droite possible
et vitesse obstacle1 et vitesse5 et distance3 alors évitement droite4
Si voie normale et non trajectoire libre et non gauche possible et droite possible
et vitesse obstacle1 et vitesse5 et distance4 alors évitement droite6
Si voie droite et non gauche libre et voie libre et vitesse obstacle1
et vitesse5 et distance0 alors évitement droite2
Si voie droite et non gauche libre et voie libre et vitesse obstacle1
et vitesse5 et distance1 alors évitement droite2
Si voie droite et non gauche libre et voie libre et vitesse obstacle1
et vitesse5 et distance2 alors évitement droite2
Si voie droite et non gauche libre et voie libre et vitesse obstacle1
et vitesse5 et distance3 alors évitement droite4
Si voie droite et non gauche libre et voie libre et vitesse obstacle1
et vitesse5 et distance4 alors évitement droite6
```

Dépassement d'un véhicule roulant à moins de 1 $m.s-1$:

```
Si voie normale et non trajectoire libre et gauche possible et vitesse
obstacle2 et non vitesse1 et distance4 alors dépassement gauche1
Si voie gauche et non droite libre et voie libre et vitesse obstacle2
alors dépassement gauche1
Si voie normale et non trajectoire libre et non gauche possible et droite possible
et vitesse obstacle2 et non vitesse1 et distance4 alors dépassement droite1
Si voie droite et non gauche libre et voie libre et vitesse obstacle2
alors dépassement droite1
```

Dépassement d'un véhicule roulant à une vitesse entre 1 et 2 $m.s-1$:

```
Si voie normale et non trajectoire libre et gauche possible et vitesse obstacle3
et distance4 et non vitesse1 et non vitesse2 alors dépassement gauche2
Si voie gauche et non droite libre et voie libre et vitesse obstacle3
alors dépassement gauche2
Si voie normale et non trajectoire libre et non gauche possible et droite possible
et non vitesse1 et non vitesse2 et vitesse obstacle3 et distance4
alors dépassement droite2
Si voie droite et non gauche libre et voie libre et vitesse obstacle3
alors dépassement droite2
```

Suivi normal de la trajectoire :

```
Si trajectoire libre et voie normale et voie libre alors normal
Si voie gauche et droite libre et voie libre alors normal
Si voie droite et gauche libre et voie libre alors normal
```

Suivi du véhicule précédent le robot :

```
Si voie normale et non trajectoire libre et non droite possible et non gauche possible
et non obstacle proche1 et non vitesse obstacle1 alors suivi
Si voie normale et non trajectoire libre et vitesse obstacle2 et vitesse1 alors suivi
Si voie normale et non trajectoire libre et vitesse obstacle3 et vitesse2 alors suivi
Si voie normale et non trajectoire libre et vitesse obstacle3 et vitesse1 alors suivi
Si voie normale et non trajectoire libre et vitesse obstacle4 alors suivi
```

Suivi de trajectoire à une vitesse inférieure à celle du véhicule précédent le robot :

```
Si voie normale et non trajectoire libre et non obstacle proche1 et vitesse obstacle3
et non distance4 et non vitesse1 et non vitesse2 et gauche possible alors suivi lent
Si voie normale et non trajectoire libre et non obstacle proche1 et vitesse obstacle3
et non distance4 et non vitesse1 et non vitesse2 et droite possible alors suivi lent
Si voie normale et non trajectoire libre et non obstacle proche1 et vitesse obstacle2
et non distance4 et non vitesse1 et gauche possible alors suivi lent
Si voie normale et non trajectoire libre et non obstacle proche1 et vitesse obstacle2
et non distance4 et non vitesse1 et droite possible alors suivi lent
```

Arrêt :

```
Si voie normale et non trajectoire libre et obstacle proche1 et vitesse obstacle3
 alors stop
Si voie normale et non trajectoire libre et obstacle proche1 et vitesse obstacle2
alors stop
Si voie normale et non trajectoire libre et non droite possible et non gauche possible
et obstacle proche1 alors stop
Si voie normale et non trajectoire libre et non droite possible et non gauche possible
et vitesse obstacle1 alors stop
Si voie droite et non voie libre et obstacle proche3 et non obstacle proche2
et non vitesse supérieure et non vitesse obstacle1 alors stop
Si voie gauche et non voie libre et obstacle proche3 et non obstacle proche2
et non vitesse supérieure et non vitesse obstacle1 alors stop
Si voie droite et non voie libre et non gauche libre et non obstacle proche2
alors stop
Si voie gauche et non voie libre et non droite libre et non obstacle proche2
alors stop
```

Rabattement rapide :

```
Si voie droite et non voie libre et gauche libre et vitesse obstacle1
et vitesse positive  et obstacle proche2 et non obstacle proche1
alors rabattement gauche
Si voie gauche et non voie libre et droite libre et vitesse obstacle1
```

```
et vitesse positive et obstacle proche2 et non obstacle proche1
alors rabattement droite
Si voie droite et non voie libre et gauche libre et vitesse obstacle1
et non obstacle proche2 alors rabattement gauche
Si voie gauche et non voie libre et droite libre et vitesse obstacle1
et non obstacle proche2 alors rabattement droite
Si voie droite et non voie libre et gauche libre et vitesse supérieure
et non obstacle proche2 alors rabattement gauche
Si voie gauche et non voie libre et droite libre et vitesse supérieure
et non obstacle proche2 alors rabattement droite
Si voie droite et non voie libre et gauche libre et rabattement
et obstacle proche2 alors rabattement gauche
Si voie gauche et non voie libre et droite libre et rabattement
et obstacle proche2 alors rabattement droite
Si voie droite et non voie libre et gauche libre et non obstacle proche3
alors rabattement gauche
Si voie gauche et non voie libre et droite libre et non obstacle proche3
alors rabattement droite
```

Recul :

```
Si voie normale et non trajectoire libre et gauche possible et vitesse obstacle1
et distance0 et arrière libre alors recul
Si voie normale et non trajectoire libre et non gauche possible et droite possible
et vitesse obstacle1 et vitesse1 et distance0 et arrière libre alors recul
Si voie droite et non voie libre et non gauche libre et obstacle
proche2  et arrière libre alors recul
Si voie gauche et non voie libre et non droite libre et obstacle
proche2  et arrière libre alors recul
Si voie droite et non voie libre et obstacle proche1 et arrière libre alors recul
Si voie gauche et non voie libre et obstacle proche1 et arrière libre alors recul
Si voie droite et non voie libre et obstacle proche2 et non vitesse positive
et arrière libre alors recul
Si voie gauche et non voie libre et obstacle proche2 et non vitesse positive
et arrière libre alors recul
Si voie droite et non voie libre et obstacle proche2 et non vitesse obstacle1
et non rabattement et arrière libre alors recul
Si voie gauche et non voie libre et obstacle proche2 et non vitesse obstacle1
et non rabattement et arrière libre alors recul
```

Arrêt d'urgence[1] :

```
Si collision et vitesse positive alors urgence
```

Échec et appel au niveau supérieur :

```
Si voie normale et non trajectoire libre et gauche possible et vitesse obstacle1
et distance0 et non arrière libre alors échec
Si voie normale et non trajectoire libre et non gauche possible et droite possible
et vitesse obstacle1 et vitesse1 et distance0 et non arrière libre alors échec
Si voie droite et non voie libre et obstacle proche1 et non arrière libre alors échec
Si voie gauche et non voie libre et obstacle proche1 et non arrière libre alors échec
```

1. Il est possible de donner à cette règle une importance supérieure aux autres en la plaçant plusieurs fois dans la base.

Bibliographie

[1] **S. Abdou**. *Spécification, vérification et implémentation de missions appliquées à des véhicules automatiques*. Thèse de doctorat, Inst. Nat. Polytechnique de Grenoble, 1997.

[2] **Y. S. Abu-Mostapha**. Hints. *Neural Computation*, 7(4):639–671, 1995.

[3] **M. Agarwal**. A systematic classification of neural-network-based control. *IEEE Control Systems*, 17(2):75–93, 1997.

[4] **R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand**. An architecture for autonomy. *Int. Journal of Robotics Research*, 17(4):315–337, 1998.

[5] **J. S. Albus**. A new approach to manipulator control: The cerebellar model articulation controller. *Trans. ASME, Journal of Dynamic Systems, Measurement and Control*, 97:220–227, 1975.

[6] **T. L. Anderson and M. Donath**. Autonomous robots and emergent behavior: A set of primitive behaviors for mobile robot control. In *Proc. of the IEEE-RSJ Int. Workshop on Intelligent Robots and Systems*, volume 2, pages 723–730, Tsuchiura (JP), 1990.

[7] **R. C. Arkin**. Motor schema based navigation for a mobile robot. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 1, pages 264–271, San Franciso, CA (US), 1987.

[8] **R. C. Arkin**. Dynamic replanning for a mobile robot based on internal sensing. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 3, pages 1416–1421, Scottsdale, AZ (US), 1989.

[9] **A. Assoum, N-E. Radi, R. Velazco, F. Elie, and R. Escoffet**. Robustness against SEU of an artificial neural network space application. *IEEE Trans. on Nuclear Science*, 43(3):973–978, 1996.

[10] **W. L. Baker and J. A. Farrell**. An introduction to connectionist learning control systems. In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control*, pages 35–64. Van Nostrand Reinhold, 1992.

[11] **A. G. Barto**. Connectionist learning for control. In W. T. Miller, R. S. Sutton, and P. J. Werbos, editors, *Neural Networks for Control*, pages 5–58. MIT Press, 1990.

[12] **A. G. Barto, R. S. Sutton, and C. W. Anderson**. Neuronlike adaptive elements that can solve difficult learning problems. *IEEE Trans. Systems, Man and Cybernetics*, 13:834–846, 1983.

[13] **E. B. Baum and D. Haussler**. What size net gives valid generalization? *Neural Computation*, 1(2):151–160, 1989.

[14] **H. R. Berenji and P. Khedar**. Learning and tuning fuzzy logic controllers through reinforcement. *IEEE Trans. on Neural Networks*, 3(5):724–740, 1992.

[15] **P. Bessiere**. Interprétation versus description: Proposition pour une théorie probabiliste des systèmes cognitifs sensori-moteurs. *Intellectica*, 1999. Soumis.

[16] **P. Bessiere and E. Dedieu**. The «beam in the bin» an illustration of a probabilistic approach to robotics. In *Int. Proc. of the Conf. on Recent Advances in Mechatronics*, Istanbul (TU), 1995.

[17] **R. Biewald**. A neural network controller for the navigation and obstacle avoidance of a mobile robot. In A. M. S. Zalzala and A. S. Morris, editors, *Neural Networks for Robotic Control: Theory and Applications*, pages 163–191. Helis Horwood, 1996.

[18] **A. Bonarini and F. Basso**. Learning to compose fuzzy behaviors for autonomous agents. *International Journal of Approximate Reasonning*, 17(4):409–432, 1997.

[19] **R. P. Bonasso, D. Kortenkamp, D. P. Miller, and M. Slack**. Experiences with an architecture for intelligent, reactive agents. *Lecture Notes in Computer Science*, 1037:187–202, 1996.

[20] **J. Borenstein and Y. Koren**. Real-time obstacle avoidance for fast mobile robot in cluttered environments. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 1, pages 572–577, Cincinatti, OH (US), 1990.

[21] **R. A. Brooks**. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[22] **R. A. Brooks**. Elephants don't play chess. In P. Maes, editor, *Designing Autonomous Agents*. MIT Press, Cambridge, MA, 1990.

[23] **R. A. Brooks and A. M. Flynn**. Fast, cheap and out of control: A robot invasion of the solar system. *Journal of the British Interplanetary System*, 42(10):478–485, 1989.

[24] **M. Cayer**. *Vocabulaire de La Robotique*. Cahiers de l'office de la langue française. Eyrolles, 1993.

[25] **M. A. Cohen and S. Grossberg**. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Trans. Systems, Man and Cybernetics*, 13:815–826, 1983.

[26] **J. H. Connel**. SSS: A hybrid architecture applied to robot navigation. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 2, pages 2719–2724, Nice (FR), 1992.

[27] **M. Costaftis**. *Comportement et contrôle des systèmes complexes : introduction aux méthodes algébriques, qualitatives et fonctionnelles*. Diderot éditeur, arts et sciences, 1997.

[28] **J. L. Crowley**. World modeling and position estimation for a mobile robot using ultrasonic ranging. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 2, pages 674–680, Scottsdale, AZ (US), 1989.

[29] **J. L. Crowley**. Mathematical foundations of navigation and perception for an autonomous mobile robot. In *Workshop on Reasoning with Uncertainty in Robotics*, Amsterdam (NL), 1995.

[30] **G. Cybenko**. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signal and Systems*, 2(4):303–314, 1989.

[31] **P. Daviet and M. Parent**. Longitudinal and lateral servoing of vehicles in a platoon. In *Proc. of the IEEE Int. Symp. on Intelligent Vehicles*, pages 41–46, Tokyo (JP), 1996.

[32] **P. Déplanques, C. Novales, R. Zapata, and B. Jouvencel**. Sensor-based control versus neural network technology for the control of fast mobile robot behaviors in unstructured environment. In *Int. Conf. on Industrial Electronics, Control, Instrumentation and Automation*, pages 694–699, San Diego (CA), 1992.

[33] **J. Droulez and A. Berthod**. Concept of dynamic memory in sensorimotor control. In D. R. Humphrey and H. J. Freund, editors, *Motor Control: concepts and Issues*. John Wiley, 1991.

[34] **L. E. Dubins**. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79:497–516, 1957.

[35] **S. E. Fahlman**. An empirical sudy of learning speed in back-propagation networks. Technical Report CMU-CS-88-162, CMU, 1988.

[36] **J. Ferber**. *Les systemes multi-agents, vers une intelligence collective*. Intereditions, 1995.

[37] **F. Fessant, E. Gauthier, I. Mekrez, and B. Amy**. Application des réseaux de neurones à l'intelligence artificielle : architectures modulaires pour le contrôle en robotique. Technical report, LEIBNIZ-IMAG, 1997. Rapport de fin d'étude. Convention DRET numéro 96.34.074.00.470.75.65.

[38] **R. J. Firby**. Adaptive execution in complex dynamic domains. PHD thesis YALEU/CSD RR 672, Yale University, 1989.

[39] **T. Fraichard**. Dynamic trajectory planning with dynamic constraints: a 'state-time space' approach. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, volume 2, pages 1394–1400, Yokohama (JP), 1993.

[40] **H. Fritz**. Model-based neural distance control for autonomous road vehicles. In *Proc. of the IEEE Int. Symp. on Intelligent Vehicles*, volume 1, pages 29–34, Tokyo (JP), 96.

[41] **P. Garnier**. *Contrôle d'execution réactif de mouvements de véhicules en environnement dynamique structuré*. Thèse de doctorat, Inst. Nat. Polytechnique de Grenoble, 1995.

[42] **E. Gat**. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proc. of the Tenth Nat. Conf. on Artificial Intelligence*, pages 809–815, San Jose (CA), 1992.

[43] **E. Gat**. On three-layer architectures. In D. Kortenkamp, R. P. Bonnasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*. MIT/AAAI Press, 1997.

[44] **E. Gat, M. Slack, D. P. Miller, and R. J. Firby**. Path planning and execution monitoring for a planetary rover. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 1, pages 20–25, Cincinatti, OH (US), 1990.

[45] **P. Y. Glorennec**. Un réseau neuro-flou évolutif. In *Neuro-Nimes*, pages 301–314, Nimes (FR), 1991.

[46] **P-Y. Glorennec**. Fuzzy Q-learning. In *Proc. of FUZZ-IEEE'97, Sixth Int. Conf. on Fuzzy Systems*, Barcelona (SP), 1997.

[47] **J. B. Gomm, J. T. Evans, and D. Williams**. Development and performance of a neural-network predictive controller. *Control Engineering Practice*, 5(1):49–59, 1997.

[48] **M. Gori and A. Tesi**. On the problem of local minima in backpropagation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14:76–86, 1992.

[49] **C. Goutte and C. Ledoux**. Synthése des techniques de commande connexionnistes. Technical report, LAFORIA, 1996.

[50] **S. Grossberg**. Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural detectors. *Biological cybernetics*, 23:121–134, 1976.

[51] **V. Gullapalli, J. A. Franklyn, and H. Benbrahim**. Acquiring robot skills via reinforcement learning. *IEEE Control Systems Magazine*, 14(1):13–24, 1994.

[52] **K. S. Halgamuge and M. Glesner**. Neural networks in designing fuzzy systems for real world applications. *Fuzzy Sets and Systems*, 65(1):1–12, 1994.

[53] **D. Harel and A. Pnueli**. On the development of reactive systems. In K. R. Apt, editor, *Logics and Models of Concurrent Systems*, pages 477–498. Springer-Verlag, 1985.

[54] **T. Hashiyama, T. Furuhashi, and Y. Uchikawa**. An interval fuzzy model using a fuzzy neural network. In *Proc. of the Int. Joint Conf. on Neural Networks*, volume 2, pages 745–750, Baltimore (MD), 1992.

[55] **M. Hassoun**. *Contrôle d'exécution de mouvements d'un robot mobile : application à l'assistance à la conduite automobile.* Thèse de doctorat, Inst. Nat. Polytechnique de Grenoble, 1994.

[56] **S. Haykin**. *Neural Networks : A Comprehensive Foundation.* Macmillan College Publisher Comp., 1994.

[57] **R. Hecht-Nielsen**. *Neurocomputing.* Addison-Wesley, 1990.

[58] **K. Heesche, W. Hauptmann, and K. Goser**. A neuro-fuzzy topology with complexity reduction for an automotive application. In *Neurap*, Marseille (FR), 1997.

[59] **P. Henaff**. *Mise en œuvre de commandes neuronales par rétropropagation indirecte : application à la robotique mobile.* Thèse de doctorat, Université Pierre et Marie Curie, 1994.

[60] **M. W. Hirsh**. Convergent activation dynamic in continuous time networks. *Neural Networks*, 2:331–349, 1989.

[61] **F. Hoffmann and G. Pfister**. Evolutionary design of a fuzzy knowledge base for a mobile robot. *Int. Journal of Approximate Reasoning*, 17:447–469, 1997.

[62] **A. A. Holenstein and E. Badreddin**. Collision avoidance in a behavior-based mobile robot design. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 1, pages 898–903, Sacramento, CA (US), 1991.

[63] **S. I. Horikawa, T. Furuhashi, and Y. Uchikawa**. On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm. *IEEE Trans. on Neural networks*, 3(5):801–805, 1992.

[64] **K. Hornik, M. Stinchcombe, and H. White**. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[65] **D. F. Hougen, F. Fischer, M. Gini, and J. Slagle**. Fast connectionist learning for trailer backing using a real robot. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 3, pages 1917–1922, Minneapolis, MN (US), 1996.

[66] **M. Humphrys**. W-learning: Competition among selfish Q-learners. Technical Report 362, University of Cambridge Computer Laboratory, 1995. Disponible à l'adresse *ftp://ftp.cl.cam.ac.uk/papers/reports/TR362-mh10006-w-learning.ps.gz*.

[67] **K. Ishii, T. Ura, and T. Fujii**. A feed forward neural network for identification and adaptive control of autonomous underwater vehicles. In *Proc. of the Int. Conf. on Neural Networks*, pages 3216–3221, Orlando (FL), 1994.

[68] **S. Ishikawa**. A method of indoor mobile robot navigation by using fuzzy control. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, volume 2, pages 1013–1018, Osaka (JP), 1991.

[69] **J-S. R. Jang and C-T. Sun**. Functionnal equivalence between radial basis function networks and fuzzy inference systems. *IEEE Trans. on Neural Networks*, 4(1):156–159, 1993.

[70] **J-S. R. Jang and C-T. Sun**. Neuro-fuzzy modeling and control. *Proc. of the IEEE*, 83(3):378–406, 1995.

[71] **B. Janusz and M. Riedmiller**. Self-learning neural control of a mobile robot. In *Proc. of the Int. Conf. on Neural Networks*, pages 2358–2363, Perth (AU), 1995.

[72] **T. Jochem, D. Pomerleau, and C. E. Thorpe**. Vision guided lane transition. In *Proc. of the IEEE Int. Symp. on Intelligent Vehicles*, pages 30–35, Detroit, MI (US), 1995.

[73] **T. Jochem, D. A. Pomerleau, and C. E. Thorpe**. Maniac: a next generation neurally based autonomous road follower. In *Image Understanding Workshop*, pages 473–479, 1993.

[74] **M. I. Jordan and D. E. Rumelhart**. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16:307–354, 1992.

[75] **Y. Kananyama and N. Miyake**. Trajectory generation for mobile robots. In *Proc. of the Int. Symp. on Robotics Research*, pages 16–23, Gouvieux (FR), 1985.

[76] **Y. Kanayama, Y. Kimura, F. Myazaki, and T. Noguchi**. A stable tracking control method for a non-holonomic mobile robot. In *Proc. of the IEEE-RSJ Int. Workshop on Intelligent Robots and Systems*, volume 2, pages 1236–1241, Osaka (JP), 1991.

[77] **J. S. Kane and T. G. Kincaid**. Optoelectronic winner-take-all vlsi shunting neural network. *IEEE Trans. on Neural Networks*, 6(6):1275–1279, 1995.

[78] **M. Kawato**. Schemes and models for control of arm trajectory. In W. T. Miller, R. S. Sutton, and P. J. Werbos, editors, *Neural Networks for Control*, pages 197–228. MIT Press, 1990.

[79] **O. Khatib**. *Commande Dynamique dans l'espace opérationnel des robots manipulateurs en présence d' obstacles.* Thèse de doctorat, École Nationale Supérieure de l'aeronautique et de l'espace, Toulouse France, 1980.

[80] **B. Kosko**. Fuzzy function approximation. In *Proc. of the Int. Joint Conf. on Neural Networks*, volume 1, pages 209–213, Baltimore (MD), 1992.

[81] **B. Kosko**. *Neural networks and fuzzy systems: a dynamical approach to machine intelligence.* Prentice Hall Int., 1992.

[82] **L. G. Kraft, Miller W. T., and D. Dietz**. Development and application of CMAC based meural network-based control. In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control*, pages 215–228. Van Nostrand Reinhold, 1992.

[83] **B. H. Krogh and C. E. Thorpe**. Integrated path planning and dynamic steering control for autonomous vehicles. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 2, pages 1664–1669, San Franciso, CA (US), 1986.

[84] **Y. Kuroe, Y. Nakai, and T. Mori**. A new neural network learning of inverse kinematics of robot manipulator. In *Proc. of the Int. Conf. on Neural Networks*, Orlando (FL), 1994.

[85] **R. La, F. Guély, and P. Siarry**. Apprentissage d'une base de règles floues par la méthode du recuit simulé. In *Troisièmes journées Nationales : Les applications des ensembles flous*, Nimes (FR), 1993.

[86] **A. Labbi**. Neural networks for decision making in dynamic environments. In *Proc. of the Int. Conf. on Artificial Neural networks*, Amsterdam (NL), 1993.

[87] **A. Labbi**. *Sur l'approximation et les systèmes dynamiques dans les réseaux neuronaux*. Thèse de doctorat, Inst. Nat. Polytechnique de Grenoble, 1993.

[88] **A. Labbi and E. Gauthier**. Combining fuzzy knowledge and data for neuro-fuzzy modeling. *Journal of Intelligent Systems*, 7(1-2):145–163, 1997.

[89] **J. C. Latombe**. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[90] **C. Laugier, T. Fraichard, P. Garnier, and I. Paromtchik**. Sensor-based control architecture for a car-like vehicle. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, Vancouver (CA), 1998.

[91] **J. P. Laumond**. Feasible trajectories for mobile robots with kinematic and environment constraints. In *Proc. of the Int. Conf. on Intelligent Autonomous Systems*, pages 346–354, Amsterdam (NL), 1986.

[92] **C. C. Lee**. Fuzzy logic in control systems: Fuzzy logic controller - part I. *IEEE Trans. Systems, Man and Cybernetics*, 20(2):404–418, 1990.

[93] **C. C. Lee**. Fuzzy logic in control systems: Fuzzy logic controller - part II. *IEEE Trans. Systems, Man and Cybernetics*, 20(2):419–435, 1990.

[94] **F. Lewis, K. Liu, and A. Yesildirek**. Neural net robot controller with guaranteed tracking performance. *IEEE Transactions on Neural Networks*, 6(3):703–715, 1995.

[95] **L. J. Lin**. Reinforcement learning for robots using neural networks. Technical Report CMU-CS-93-103, Carnegie Mellon University, 1993.

[96] **L. Magdalena**. Analysis of hybrid models: fuzzy logic neural nets. Technical report, Universitad Politécnica de Madrid, 1995. MIX Esprit Project 9119.

[97] **E. H. Mamdani**. Applications of fuzzy algorithms for simple dynamic plants. *Proc. of the IEEE*, 121(12):1585–1588, 1974.

[98] **D. L. Marruedo and J. G. Ortega**. Neural mobile robot navigation based on a 2D laser range sensor. In *Proc. of the 3rd IFAC Symp. on Intelligent Autonomous Vehicles*, volume 1, pages 298–303, MADRID (SP), 1998.

[99] **D. McFarland**. *The Oxford Companion to Animal Behavior*. Oxford University Press, 1987.

[100] **X. Menage and R. Hartani**. Synthèse des méthodes d'association des techniques neuronales et des techniques floues. Technical report, LAFORIA Institut Blaise Pascal, Paris, 1993.

[101] **A. Micaelli and C. Samson**. Trajectory tracking for unicycle-type and two-steering-wheels mobile robots. Technical Report 2097, INRIA, 1993.

[102] **J. R. Millan and C. Torras**. Efficient reinforcement learning of navigation strategies in an autonomous robot. *Machine Learning*, 8(3):363–395, 1992.

[103] **P. M. Mills, A. Y. Zomaya, and M. O. Tadé**. Adaptive model-based control using neural networks. *International Journal of Control*, 60(6):1163–1192, 1994.

[104] **J. Moddy and C. Darken**. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[105] **H. P. Moravec**. The stanford cart and the CMU rover. *Proceedings of the IEEE*, 71(7):872–884, 1983.

[106] **M. T. Musawi, W. Ahmed, K. H. Chan, K. B. Faris, and D. M. Hummels**. On the training of radial basis function classifiers. *Neural Networks*, 5(4):595–603, 1992.

[107] **P. Musilek**. Neural networks in navigation of mobile robots: a survey. *Neural Networks World*, 6:917–928, 1995.

[108] **H. Myamoto, M. Kawato, T. Setoyama, and R. Suzuki**. Feedback-error-learning for trajectory control of a robotic manipulator. *Neural Networks*, 1:251–265, 1988.

[109] **K. Narendra and A. M. Annaswamy**. *Stable Adaptive Systems*. Prentice-Hall, Englewood, NJ, 1989.

[110] **K.S. Narendra and K. Parthasarathy**. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–5, 1990.

[111] **D. Nauck and R. Kruse**. A fuzzy neural network learning fuzzy control rules and menbership functions by fuzzy error backpropagation. In *Proc. of the Int. Conf. on Neural Networks*, volume 2, pages 1022–1027, San Francisco (CA), 1993.

[112] **W. L. Nelson**. Continuous curvature paths for autonomous vehicle. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 3, pages 1260–1264, Scottsdale, AZ (US), 1989.

[113] **D. Nguyen and B. Widrow**. The truck backer-upper: An example of self-learning in neural networks. In *Proc. of the Int. Joint Conf. on Neural Networks*, volume 2, pages 357–362, Washington (DC), 1989.

[114] **N. J. Nilsson**. Shakey the robot. Technical Report 323, Artificial Intelligence Center, SRI, Menlo Park, California, 1984.

[115] **F. Noreils**. Integrating error recovery in a mobile robot control system. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 1, pages 396–401, Cincinatti, OH (US), 1990.

[116] **C. Novales, D. Pallard, and C. Laugier**. Controlling the motions of an autonomous vehicle using a local navigator. In *Proc. of the Int. Symp. on Robotics and Manufacturing*, Montpellier (FR), 1996.

[117] **H. Ohno, T. Suzuki, K. Aoki, A. Takahasi, and G. Sugimoto**. Neural network control for automatic braking control system. *Neural Networks*, 7(8):1303–1312, 1994.

[118] **B. Orsier**. *Étude et application de systèmes hybrides neurosymboliques.* Thèse de doctorat, Université Joseph Fourrier Grenoble, 1995.

[119] **B. Orsier and A. Labbi**. NESSY3L: a neurosymbolic system with 3 levels. In R. Sun and F. Alexandre, editors, *Working Notes of the IJCAI Workshop on Connectionist-Symbolic Integration.* , Montreal (CA), 1995.

[120] **J. G. Ortega and E. F. Camacho**. Mobile robot navigation in a partially structured static environment, using neural predictive control. *Control Engineering Practice*, 4(12):1669–1679, 1996.

[121] **M. Ortiz and P. J. Zufiria**. Evaluation of reinforcement learning autonomous navigation systems for a nomad 200 mobile robot. In *Proc. of the 3rd IFAC Symp. on Intelligent Autonomous Vehicles*, volume 1, pages 292–297, MADRID (SP), 1998.

[122] **F. S. Osorio and B. Amy**. INSS: an hybrid system for constructive machine learning. In *Les réseaux neuromimétiques et leurs applications (NEURAP)*, pages 369–375, Marseille (FR), 1998.

[123] **I. Paromtchik and C. Laugier**. Motion generation and control for parking an autonomous vehicle. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 3, pages 3117–3122, Minneapolis, MN (US), 1996.

[124] **O. Patrouix**. *Modelisation multi-niveaux de donnees ultrasonores : application à la robotique mobile.* Thèse de doctorat, Montpellier, 1994.

[125] **H. Paugam-Moisy**. Multiprocessor simulation of neural networks. In M. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 605–608. MIT Press, San Antonio (TX), 1995.

[126] **D. W. Payton**. An architecture for reflexive autonomous vehicle control. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 3, pages 1838–1845, San Franciso, CA (US), 1986.

[127] **D. W. Payton, J. K. Rosenblatt, and D. M. Keirsey**. Plan guided reaction. *IEEE Trans. Systems, Man and Cybernetics*, 20(6):1370–1382, 1990.

[128] **N. Pican and F. Alexandre**. Highly adaptative neural networks for adaptive neuro-control: The OWE architecture. In *Proc. of the International Conf. IEEE on Systems, Man and Cybernetics*, 1994.

[129] **F. Pin, H. Watanabe, J. Symon, and R. Pattay**. Autonomous navigation of a mobile robot using custom-designed qualitative reasonning VLSI chips and boards. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 1, pages 123–128, Nice (FR), 1992.

[130] **F. J. Pineda**. Recurrent backpropagation and the dynamical approach to adaptive neural computation. *Neural Computation*, 1(2):161–172, 1989.

[131] **D. A. Pomerleau**. *Neural Network Perception for Mobile Robot Guidance*. Kluwer Academic Press, 1993.

[132] **D. A. Pomerleau, J. Gowdy, and C. E. Thorpe**. Combining artificial neural networks and symbolic processing for autonomous robot guidance. *Engineering Applications of Artificial Intelligence*, 4(4):279–285, 1991.

[133] **D. Psaltis, A. Sideris, and A. Yamamura**. Neural controllers. In *Proc. of the Int. Conf. on Neural Networks*, volume 4, pages 551–558, San Diego (CA), 1987.

[134] **D. Psaltis, A. Sideris, and A. Yamamura**. A multi-layer neural network controller. *IEEE Control Systems Magazine*, 8:17–21, 1988.

[135] **P. Reignier**. *Pilotage réactif d'un robot mobile : étude du lien entre la perception et l'action*. Thèse de doctorat, Inst. Nat. Polytechnique de Grenoble, 1994.

[136] **P. Reignier**. Supervised incremental learning of fuzzy rules. *Robotics and Autonomous Systems*, 16(1):57–72, 1995.

[137] **P. Reignier, V. Hansen, and J. L. Crowley**. Incremental supervised learning for mobile robot reactive control. In *Proc. of the Int. Conf. on Intelligent Autonomous Systems*, volume 1, pages 287–295, Karlsruhe (DE), 1995.

[138] **J-M. Renders**. *Algorithmes génétiques et réseaux de neurones*. Hermes, 1995.

[139] **M. Riedmiller and H. Braun**. A direct adaptive method for faster backpropagation learning : the RPROP algorithm. In *Proc. of the Int. Conf. on Neural Networks*, San Francisco (CA), 1993.

[140] **I. Rivals, L. Personnaz, G. Dreyfus, and D. Canas**. Real-time control of an autonomous vehicle : A neural network approach to the path following problem. In *Neuro-Nimes*, pages 219–229, Nimes (FR), 1994.

[141] **J. K. Rosenblatt and D. W. Payton**. A fine-grained alternative to the subsumption architecture for mobile robot control. In *Proc. of the Int. Joint Conf. on Neural Networks*, volume 2, pages 317–323, Washington (DC), 1989.

[142] **D. E. Rumelhart and J. L. McClelland**. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. M.I.T. Press, 1986.

[143] **A. Saffiotti**. The uses of fuzzy logic in autonomous robot navigation. *Soft Computing*, 1(4):180–197, 1997.

[144] **C. Samson**. Path following and time-varying feedback stabilization of a wheeled mobile robot. In *Proc. of the Int. Conf. on Control, Automation, Robotics and Vision*, pages RO–13.1.1–RO–13.1.5, Singapore (SG), 1992.

[145] **A. Scheuer and T. Fraichard**. Continuous curvature path planning for car-like vehicles. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, volume 2, pages 997–1003, Grenoble (FR), 1997.

[146] **B. Schiele and J. L. Crowley**. Certainty grids: Perception and localisation for a mobile robot. *Robotics and Autonomous Systems*, 12(3-4):163–172, 1994.

[147] **A. Segovia**. *Étude des déplacements d'un robot mobile dans un environnement peu contraint*. Thèse de doctorat, Université de Compiègne, 1993.

[148] **R. G. Simmons**. Structured control for autonomous robots. *IEEE Trans. Robotics and Automation*, 10(1):34–43, 1994.

[149] **K. T. Song and J. C. Tai**. Fuzzy navigation of a mobile robot. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, volume 1, pages 621–627, Raleigh, NC (US), 1992.

[150] **M. Sugeno and M. Nishida**. Fuzzy control of a model car. *Fuzzy Sets Systems*, 13:103–113, 1985.

[151] **S. M. Sulzberger, N. N. Tschichlod-Gurman, and S. J. Vestli**. Fun: Optimization of fuzzy rule based systems using neural networks. In *Proc. of the Int. Conf. on Neural Networks*, volume 1, pages 312–316, San Francisco (CA), 1993.

[152] **R. Sun**. Robust reasonning: integrating rule-based and similarity-based reasoning. *Artificial Intelligence*, 75(2):241–296, 1995.

[153] **R. Sun and F. Alexandre**. *Connectionist- Symbolic Integration: from Unified to Hybrid Approaches*. Lawrence Erlbaum Associates, 1997.

[154] **R. Sun and T. Peterson**. A hybrid learning model for reactive sequential decision making. In R. Sun and F. Alexandre, editors, *Working Notes of the IJCAI Workshop on Connectionist-Symbolic Integration.* , Montreal (CA), 1995.

[155] **R. S. Sutton**. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[156] **R. S. Sutton**. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proc. of the Seventh Int. Conf. on Machine Learning*, Austin (TX), 1990.

[157] **S. Thrun**. The role of exploration in learning control. In D. A. White and D. A. Sofge, editors, *Handbook of intelligent control*, pages 527–560. Van Nostrand Reinhold, 1992.

[158] **S. Thrun**. Exploration in active learning. In M. A. Arbib, editor, *Handbook of brain theory and neural networks*, pages 381–384. MIT Press, 1995.

[159] **C. Torras**. Motion planning and control: Symbolic and neural levels of computation. In *third Cognitiva conf.*, pages 207–218, Madrid (SP), 1990.

[160] **C. Touzet and S. Sehad**. Neural reinforcement path planning for the miniature robot khepera. In *Proc. of the World Congress on Neural Networks*, volume 2, pages 350–354, Washington (DC), 1995.

[161] **Y. Tsukamoto**. An approach to fuzzy reasoning method. In M. M. Gupta, R. K. Ragade, and R. R. Yager, editors, *Advances in fuzzy-set theory and application*, pages 137–149. North-Holland, Amsterdam, 1979.

[162] **K. P. Venugopal, A. S. Pandya, and R. Sudhakar**. A recurrent neural network controller and learning algorithm for the on-line learning control of autonomous underwater vehicles. *Neural Networks*, 7(5):833–846, 1994.

[163] **C. Watkins and P. Dayan**. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[164] **P. J. Werbos**. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10), 1990.

[165] **P. J. Werbos**. Approximate dynamic programming for real-time control and neural modeling. In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control*, pages 493–526. Van Nostrand Rheinold, 1992.

[166] **D. A. White and D. A. Sofge**. Applied learning optimal control for manufacturing. In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control*, pages 259–282. Van Nostrand Rheinold, 1992.

[167] **B. Widrow and F. W. Smith**. Pattern-recognizing control systems. In *Proceeding of Computer and Information Sciences*, 1964.

[168] **Q. H. Wu, B. W. Hogg, and G. W. Irwin**. A neural network regulator for turbogenerators. *IEEE Transactions on Neural Networks*, 3(1):95–100, 1992.

[169] **K. Yoshizawa, H. Hashimoto, M. Wada, and S. Mori**. Path tracking control of mobile robots using a quadratic curve. In *Proc. of the IEEE Int. Symp. on Intelligent Vehicles*, volume 1, pages 58–63, Tokyo (JP), 96.

[170] **L. A. Zadeh**. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[171] **R. Zapata, P. Lepinay, and P. Déplanques**. Collision avoidance using an egocentric memory of proximity. *Lecture Notes in Computer Science*, 929:614–624, 1995.

[172] **R. Zapata, P. Lépinay, P. Thomson, and B. Jouvencel**. Reactive behaviors of fast mobile robots. *Journal of Robotic Systems*, 11(1):13–20, 1994.

[173] **Y. Zhang, P. Sen, and G. E. Hearn**. An on-line trained adaptive neural controller. *IEEE Control Systems Magazine*, 15(5):67–75, 1995.

Index des auteurs cités

Table des figures

Utilisation des Réseaux de Neurones Artificiels pour la Commande d'un Véhicule Autonome

Résumé : Le sujet de cette thèse se situe à l'intersection des domaines de la robotique mobile et des réseaux de neurones artificiels (RNA). Notre objectif est d'étudier les solutions que peuvent apporter les techniques connexionnistes aux problèmes particuliers posés par la commande automatique d'un robot de type voiture. Ce mémoire se compose de deux parties principales. La première d'entre elles traite des aspects fondamentaux de la commande d'un robot mobile et de l'utilisation des réseaux de neurones artificiels pour la commande de systèmes complexes. Cette première étude nous permet de mettre en évidence les différents points sur lesquels les réseaux de neurones peuvent jouer un rôle dans une architecture de commande conférant une véritable autonomie de mouvements au véhicule, tout en respectant les contraintes de robustesse et de rapidité de réaction induites par l'utilisation d'un robot de la taille et de la vitesse d'une voiture. Nous proposons dans la deuxième partie du mémoire plusieurs contrôleurs permettant d'accroître progressivement l'autonomie du robot. Nous nous intéressons tout d'abord à une tâche simple consistant uniquement à asservir le robot sur une trajectoire de référence issue d'un planificateur. Notre approche autorise une adaptation continue du système face à d'éventuels changements des paramètres du robot ou de son environnement. Afin de permettre la réalisation de manœuvres sans consignes extérieures, nous proposons également une méthodologie pour la réalisation de contrôleurs basés sur l'utilisation des capteurs externes du véhicule. Notre appoche utilise un modèle alliant des caractéristiques issues de la logique floue et des RNA. Enfin nous montrons comment des tâches complexes peuvent être réalisées à partir de l'enchaînement de plusieurs contrôleurs simples. Notre réalisation du système de sélection de ces contrôleurs, utilisant un RNA récurrent, possède des capacités de robustesse et autorise des réactions très rapides face à l'ensemble des événements extérieurs qui doivent pouvoir être pris en compte.

Mots clés : Réseaux de neurones artificiels, Robots mobiles, Systèmes hybrides, Logique floue, Non-holonomie.

Artificial Neural Networks for Automatic Vehicle Control

Abstract : The subject of this thesis covers both mobile robotic and artificial neural networks (ANN) fields. Our aim is to study solutions that connectionist techniques can bring to particular problems raised by the automatic control of a car-like vehicle. This report is composed of two main parts. The first of them processes fundamental aspects of mobile robot control and of the use of artificial neural networks for control of complex systems. This first study allows us to underline the different points where ANN can contribute in a control architecture providing a real autonomy to the vehicle while respecting the robustness and rapidity constraints induced by the utilisation of a robot of the size and the speed of a car. We propose in the second part of this report several controllers allowing gradual increase of the robot autonomy. First of all, we are interested in a simple task consisting only in enslaving the robot on a reference path given by a planner. Our approach enables a continuous adaptation of the system facing possible changes of the parameters of the robot or its environment. So as to allow the execution of manoeuvres without external orders, we also propose a methodology for the realisation of controllers based on external sensors of the vehicle. Our approach uses a model allying characteristics from both fuzzy logic and ANN. Finally we show how complex tasks can be realised using a sequence of several simple controllers. Our realisation of the selection system for these controllers, which uses a recurrent ANN, exhibits some characteristics of robustness and very fast reactions when faced to the external events that must be taken into account.

Keywords : Artificial neural networks, Mobile robots, Hybrid systems, Fuzzy logic, Non-holonomy.