Pilotage reactif d'un robot mobile : etude du lien entre la perception et l'action

Patrick Reignier

THESE

présentée à

l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

pour obtenir le grade de

DOCTEUR spécialité INFORMATIQUE

Arrêté ministériel du 23 novembre 1988

—

PILOTAGE REACTIF D'UN ROBOT MOBILE ETUDE DU LIEN ENTRE LA PERCEPTION ET L'ACTION

—

Patrick REIGNIER

—

Thèse soutenue le 13 Décembre 1994

Composition du jury :

Président :	Christian Laugier
Rapporteurs :	M. Dominique Meizel
	M. Pierre-Yves Glorennec
Examinateurs :	M. José Millan
	M. Rachid Alami
	M. James L. Crowley

Laboratoire d'Informatique Fondamentale et d'Intelligence Artificielle
46Γav. Félix Viallet. 38031 Grenoble Cedex

Remerciements

Je remercie tout d'abord les membres du jury qui ont accepté de juger ce travail et d'y apporter leur caution :

- M. Christian Laugier Directeur de recherche à l'INRIA pour l'honneur qu'il me fait de présider le jury.
- M. Pierre Yves Glorennec Chargé de recherche à l'INSA de Rennes M. Dominique Meizel Professeur à l'Université Technologie de Compiègne ainsi que M. Laurent Foulloy Professeur à l'Ecole Supérieure d'Ingénieurs d'Annecy d'avoir accepté la lourde tâche d'être rapporteurs.
- M. José Millan chargé de recherche au Joint Research Center d'Ispra Italie d'avoir accepté d'être examinateur.
- M. Rachid Alami chargé de recherche au Laboratoire d'Automatique et d'Analyse des Systèmes de Toulouse d'avoir accepté d'être examinateur.
- et enfin M. James L. Crowley Professeur à l'ENSIMAG examinateur mais également et surtout directeur de cette thèse. Je tiens à le remercier pour son dynamisme pour ses nombreuses discussions et pour m'avoir permis d'accomplir ces travaux dans les meilleures conditions.

Une thèse c'est tout d'abord un laboratoire d'accueil. Je tiens à remercier M. Philippe Jorrand Directeur du LIFIA de m'avoir accepté au sein de son laboratoire. Je tiens également à remercier l'ensemble du personnel administratif (Laurence Danièle Mirella Claudia Claude Lionel Henri) pour leur gentillesse et leur compétence.

Une thèse c'est également une équipe de recherche. Je tiens à ce titre à remercier tous les membres de l'équipe PRIMA (Augustin Olivier et Olivier Christophe Patrice Philippe Cordélia Claude Bruno Mouafak Bernt Jaime Nils Yves ...) pour leur enthousiasme et pour permettre à ce groupe d'être un groupe convivial où il fait bon travailler.

J'ajouterai une pensée émue au robot du LIFIA pour cet enthousiasme cette incroyable créativité ainsi qu'un acharnement à faire ce qu'on lui dit et non pas ce qu'on voudrait qu'il fasse qui force le respect et brise les moments les plus monotones.

Je profite de cette occasion qui m'est offerte pour remercier tous mes amis qui m'ont beaucoup aidé durant cette période et qui me sont très chers (Alain Valérie Thierry les deux Pascal Mouna Artur Janusz ...) et tant d'autres que je ne peux tous les citer.

Enfin les derniers mots sont pour remercier toute ma famille pour leur aide et leurs encouragements constants et pour te remercier toi Nina pour ta présence pour ton aide inestimable ta patience et ton soutien permanent.

Patrick

Table des matières

1 Introduction **7**
1.1 L'autonomie de déplacement 8
1.2 Etudes réalisées 9
1.3 Organisation du rapport 14

2 Le projet MITHRA **17**
2.1 La robotique mobile autonome 17
2.2 Architectures hiérarchiques de contrôle 19
2.3 Le système MITHRA 21
2.3.1 Le contrôleur de véhicule 22
2.3.2 Représentation de l'environnement 25
2.3.3 La modélisation 30
2.3.4 La perception 32
2.3.5 La navigation 40
2.4 Résultats expérimentaux 41
2.4.1 Le contrôleur de véhicule 41
2.4.2 La modélisation 41
2.4.3 La perception et la navigation 41
2.5 Conclusion 43

3 La navigation réactive **45**
3.1 Motivations 45
3.2 Approches comportementales 47
3.2.1 Le comportement animal 47
3.2.2 Les architectures purement comportementales 49
3.2.3 Les architectures hybrides 52
3.3 Cadre de notre étude 56
3.3.1 Architecture retenue 56
3.3.2 Définition d'un comportement de navigation 57

4 Les champs de potentiels **61**
4.1 Génération de commandes par potentiels 61
4.1.1 Evitement en ligne d'obstacles imprévus 61

4.1.2 Les fonctions de navigation 63
4.1.3 Les potentiels généralisés 63
4.1.4 La méthode des schémas 64
4.2 Génération de chemins par potentiels 65
4.2.1 Planification en Profondeur 65
4.2.2 Planification par le Meilleur d'Abord 66
4.2.3 Planification par fonction de navigation numérique 66
4.2.4 Utilisation de fonctions harmoniques 67
4.2.5 Planification variationnelle 69
4.2.6 Planification par déformation d'un chemin 69
4.3 Utilisation de champs de forces 70
4.3.1 La méthode des champs de forces virtuels 71
4.4 Approche proposée 73
4.4.1 Principe général 73
4.4.2 Détermination de $\vec{F}_{but}$ 74
4.4.3 Détermination de $\vec{F}_{obstacles}$ 74
4.4.4 Commande générée 76
4.4.5 Gestion des minima locaux 76
4.4.6 Gestion des rotations 78
4.4.7 Gestion de la marche arrière 80
4.5 Evaluation 80
4.5.1 Evitement d'un obstacle simple 80
4.5.2 Illustration de l'adaptation dynamique du champ 82
4.5.3 Exemple de marche arrière 83
4.5.4 Courbe de vitesse 86
4.5.5 Situation d'échec 87
4.6 Conclusion 88

5 Le Contrôle Flou **93**
5.1 Introduction et motivations 93
5.2 Le contrôle flou 93
5.2.1 Historique 94
5.2.2 Les ensembles flous 94
5.2.3 La logique floue : représentation de connaissances et inférence 98
5.2.4 Le contrôle flou 102
5.3 Le système réactif flou 111
5.3.1 Choix des données d'entrée et de sortie 112
5.3.2 Module de codage 115
5.3.3 Module de décodage 115
5.3.4 Détermination de la logique de décision 118
5.3.5 Détermination des règles 120
5.4 Résultats expérimentaux 127
5.4.1 Edition de la surface de contrôle 127

5.4.2 Exemples d'exécution 129
5.5 Conclusion 133

6 L'apprentissage en contrôle 135
6.1 Introduction 135
6.2 Le Contrôle adaptatif 136
6.2.1 Définition 136
6.2.2 Commande adaptative à paramètres pré-programmés . . . 137
6.2.3 Commande adaptative à modèle de référence 137
6.2.4 Commande adaptative avec identification du système à régler 140
6.3 L'apprentissage en contrôle 141
6.3.1 Définition 141
6.3.2 Informations disponibles : paradigmes d'apprentissage . . . 142
6.3.3 Méthodes d'apprentissage 144
6.3.4 Paradigmes et algorithmes 147
6.4 Application à l'apprentissage en robotique 150
6.4.1 Problèmes particuliers 150
6.4.2 Paradigmes possibles 152

7 Naviguer par l'observation 155
7.1 Introduction 155
7.2 Apprentissage neuronal de fonctions 157
7.2.1 Utilisation d'un modèle fixe 157
7.2.2 Apprentissage d'un modèle 163
7.3 Exemples d'applications en robotique 175
7.3.1 Apprentissage par rétro-propagation du gradient 176
7.3.2 Apprentissage par méthodes locales 181
7.4 Apprentissage direct de la loi de commande 182
7.4.1 Acquisition des données 182
7.4.2 Apprentissage incrémental 183
7.4.3 Généralisation 187
7.4.4 Conclusion 187
7.5 Pré-traitement des données sensorielles 187
7.5.1 Réduction de dimension par extraction de vecteurs propres 191
7.6 Pré-traitement des données sensorielles par réduction linéaire . . . 195
7.6.1 Evaluation des indices extraits 195
7.6.2 Choix de la dimension de représentation 198
7.7 Pré-traitement des données sensorielles par réduction non linéaire 200
7.8 Conclusion 203

8 Naviguer par l'échec **205**
8.1 La méthode des Différences Temporelles 205
8.1.1 Apprendre à prédire 205
8.1.2 L'approche $TD(\lambda)$ 206
8.1.3 Exemples 208
8.1.4 Extensions de l'algorithme 210
8.2 L'apprentissage renforcé 211
8.2.1 Algorithme général 211
8.2.2 Evaluation de la transformation apprise 212
8.2.3 Le Critique Heuristique Adaptatif 215
8.2.4 Le *Q-Learning* 216
8.3 Réduction du temps d'apprentissage 218
8.3.1 Exploration et exploitation 218
8.3.2 Utilisation d'expériences mentales 219
8.3.3 Apprentissage par explication 220
8.3.4 Utilisation de connaissances initiales expertes 221
8.4 Choix de l'approche retenue 224
8.5 Formulation du problème 225
8.6 L'apprentissage en logique floue 227
8.6.1 L'apprentissage des paramètres 228
8.6.2 L'apprentissage structurel 229
8.7 Apprentissage supervisé incrémental de règles floues 232
8.7.1 Création d'une nouvelle règle 235
8.7.2 Adaptation d'une règle 239
8.7.3 Généralisation d'une règle 240
8.7.4 Algorithme 241
8.8 Application à l'apprentissage de fonctions 243
8.8.1 Exemple de déformation de surface 243
8.8.2 Incrémentalité de l'apprentissage 243
8.8.3 Capacités de généralisation et de mémorisation 247
8.9 Observation du comportement du véhicule 250
8.10 Conclusion 253

9 Conclusion **255**
9.1 Travaux réalisés 256
9.2 Discussion et perspectives 261
9.2.1 Reformulation de l'approche 261
9.2.2 Perspectives 262

A Le système Molusc **265**
A.1 Mots clés de Molusc Clips 265

B Le système réactif flou **273**
B.1 Définition des données linguistiques 273
B.2 Règles du système de navigation réactive 274
B.2.1 Entête 274
B.2.2 Détection de la situation courante 274
B.2.3 Gestion du but en tenant comptes des obstacles 275
B.2.4 Gestion des obstacles 276

C LMS et incrémentalité **279**
C.1 Apprentissage par descente de gradient 279
C.2 Apprentissage par la loi *LMS* 279

Table des figures **285**

Liste des tables **293**

Bibliographie **295**

Chapitre 1

Introduction

L'étude que nous avons menée au cours de ce travail s'est inscrite initialement dans le cadre du projet de recherche Européen Mithra Eureka EU 110. Le but de ce projet était la conception et la réalisation d'une famille de robots mobiles possédant un centre décisionnel embarqué et destinés à des tâches de télé-surveillance et de première intervention. Ces robots sont conçus pour évoluer dans un environnement structuré de type industriel.

Les robots Mithra doivent pouvoir être en mesure d'effectuer des missions de surveillance décrites par un opérateur humain non informaticien grâce à un langage de haut niveau. Par exemple :
le robot A doit être devant l'entrée à 20h00. Il restera ici un quart d'heure en détectant la présence d'intrus. Il rejoindra ensuite la pièce 1c du laboratoire en détectant sur son parcours la présence de chaleur anormale. Il rejoindra ensuite le premier étage où il patrouillera dans les couloirs durant 2 heures en détectant la présence d'intrus.

Les premiers travaux en robotique mobile en milieu industriel ne conférait au véhicule qu'une autonomie de déplacement très limitée et nécessitait la réalisation d'importants et coûteux travaux d'infrastructure. C'était le cas en particulier des chariots filoguidésΠlimitant le déplacement du robot à des voies lui étant réservées. De manière à se libérer de ces contraintesΓla tendance actuelleΓdans laquelle s'inscrit MithraΓa pour but d'essayer d'intégrer le plus facilement possible le système robotique dans le cadre de l'entreprise en lui permettant de se déplacer dans un environnement non spécialement préparé pour lui parmi un ensemble d'obstacles imprévus.

La section suivante nous permettra de définir le problème que nous avons abordé. Elle sera suivie par une présentation des différentes études que nous avons réalisées. Nous terminerons enfin cette introduction par une brève description des différents chapitres de ce document.

1.1 L'autonomie de déplacement

Le problème abordé dans cette étude est celui de l'autonomie de déplacement telle qu'elle a été décrite lors de la section précédente. Il peut être décomposé en deux niveaux distincts :

1. le niveau cartographique. Il permet de gérer l'organisation du bâtiment ainsi que les divers noms symboliques utilisés pour la désignation des pièces.

2. le niveau géométrique. Il permet de gérer le déplacement du véhicule au milieu d'obstacles afin de rejoindre les emplacements intermédiaires (connus par leurs coordonnées absolues) fournis par la cartographie.

Nous nous intéresserons uniquement à ce second niveau au cours de notre étude.

Vouloir rejoindre un point P de l'espace d'évolution désigné par ses coordonnées absolues signifie que le robot est en mesure :

- de connaître avec suffisamment de précision sa position courante.

- de détecter la présence d'obstacles éventuels le séparant du but (le robot doit pouvoir se déplacer dans un environnement dynamique non spécialement préparé pour lui).

- de trouver un passage entre ces obstacles.

Un robot mobile autonome est un système mécaniqueÎélectronique et informatique complexe mettant en œuvre en particulier :

- un ensemble de capteurs. Ils peuvent être de deux types différents :

  1. extéroceptifs (télémètresÎcaméras etc).
  2. intéroceptifs (odomètres par exemple).

  Les capteurs extéroceptifs ont pour objectif d'acquérir des informations sur l'environnement proche du véhicule. Les capteurs intéroceptifs fournissent des données sur l'état interne du robot (telles que sa vitesse ou sa position).

- un ensemble d'effecteurs.

L'objectif du robot est d'atteindre un point de l'espace en évitant les obstacles. Le problème que l'on doit résoudre est de déterminer en fonction des données capteurs quelles commandes doivent être envoyées à chaque instant au robot pour atteindre cet objectif.

1.2 Etudes réalisées

Nous avons tout d'abord mis en place dans le cadre du projet Mithra un premier système de navigation pour le robot mobile. Ce système est basé sur une architecture hiérarchique de type à la fois fonctionnel et fréquentiel. Nous avons dans ce cadre conçu et réalisé les modules suivants :

- le contrôleur de véhicule. Son rôle est d'asservir le moteur droit et gauche du véhicule afin de permettre des mouvements simultanés de translation et de rotation. Il a également pour tâche de maintenir la position estimée.

- le processus de modélisation. Ce processus a pour objectif de réaliser une description de l'environnement en terme de segments de droite à partir des données fournies par les capteurs ultrasons. L'objectif de ce modèle est double :

  1. permettre au robot de corriger l'erreur commise par les odomètres et donc de se relocaliser dans son environnement.
  2. permettre de détecter les obstacles.

- le module de perception. Son rôle est d'interpréter le modèle construit de manière à extraire un chemin permettant au robot de rejoindre le but tout en évitant les obstacles détectés.

- le module de navigation. Ce module a pour tâche d'exécuter le chemin déterminé par la perception.

Nous avons réalisé à partir de ce système une série d'expériences sur le robot ROBUTER du LIFIA. Elles nous ont permis de mettre en évidence les points suivants :

- Le modèle de l'environnement en terme de segments de droite permet de relocaliser le robot de manière satisfaisante si le domaine d'évolution contient suffisamment de surfaces planes visibles (murs dégagés par exemple).

- les données contenues dans le modèle sont en revanche insuffisantes pour résoudre le problème de la navigation : beaucoup d'obstacles ne sont pas suffisamment réguliers pour être perçus sous forme d'ensemble de segments de droites.

- le délai nécessaire entre l'apparition d'un nouvel obstacle dans le champ de vue du robotΓson intégration dans le modèle et la génération d'un nouveau chemin permettant de l'éviter est trop important et représente un danger pour le véhicule.

– le plan généré repose sur le contenu du modèle de l'environnement. Ce modèle est en constante évolution et se modifie au fur et à mesure que le robot se déplace et peut être en mesure de mieux apercevoir les différents obstacles. Le plan est donc susceptible d'être fréquemment remis en cause.

Nous avons jugé les performances de navigation de notre système insuffisante : les plans générés ne sont pas suffisamment fiables et le temps de réponse de l'ensemble est nettement insuffisant. Ceci est dû en particulier au recours à une série d'opérations trop complexes entre la perception et l'action. Nous avons alors orienté nos recherches vers les systèmes de navigation réactive.

Depuis les premiers travaux de Brooks[les systèmes comportementaux ont connu un très grand succès en robotique mobile comme en robotique de manipulation. D'inspiration éthologique ou physiologique[ils tentent de reproduire les principales propriétés du comportement des êtres vivants telles que la robustesse ou l'adaptabilité face à de nouvelles situations. La plupart des recherches dans ce domaine s'intéressent à la construction de comportements complexes à partir de comportements élémentaires. Nous avons préféré laisser l'élaboration de comportements complexes à des planificateurs[adaptés à ce type de tâche[pour nous intéresser explicitement à la réalisation du comportement élémentaire nous intéressant : la navigation vers un but en évitant les obstacles.

Un comportement de navigation peut être défini comme une transformation f entre un espace de perception et un espace d'action. L'espace de perception est un espace dans lequel chaque axe est associé à un capteur réel ou virtuel.

Nous nous sommes tout d'abord intéressés aux méthodes à base de potentiels comme approche possible pour décrire f. La direction et la vitesse du véhicule sont déterminées en appliquant directement aux mesures capteurs une fonction classique d'attraction vers le but et de répulsion par les obstacles. Le problème classique de telles méthodes est la présence d'oscillations et de minima locaux. Les minima locaux peuvent être provoqués par une configuration d'obstacles imposant au robot de devoir s'éloigner momentanément du but afin de pouvoir l'atteindre. De manière à éviter ce type de blocages présents dans une zone généralement restreinte de l'espace[nous avons adopté un algorithme favorisant le mouvement. Le robot cherche à réaliser en permanence des déplacements en translation (sous surveillance d'un module de réflexe)[le champs de force ne s'occupant principalement que de gérer la direction. Les minima peuvent également être provoqués par la présence d'un passage étroit que le robot doit emprunter. Lorsqu'une telle situation est détectée[nous modifions dynamiquement et temporairement les forces de répulsion de manière à ce que le véhicule puisse franchir la zone.

Les problèmes de minima locaux donnent au robot un comportement hésitant et le laissent bloqué dans un certain nombre de situations. Le problème d'oscillations[que nous n'avons pas abordé[est également nuisible au comportement général. Tous ces problèmes sont inhérents aux fonctions choisies et modifier les

paramètres (comme l'ajustement dynamique des forces de répulsion) ne fait que déplacer les problèmes sans les résoudre. Nous avons cherché à réaliser la transformation perception - action non plus à l'aide d'une famille fixe de fonctions mais à l'aide d'un approximateur universel capable d'approcher n'importe quelle fonction continue. L'approximateur que nous avons choisi est la logique floue pour sa capacité à traduire des connaissances symboliques du problème en une fonction numérique.

Dans l'approche que nous avons réalisée les données sont tout d'abord prétraitées de manière à réduire la dimension de l'espace perceptif et donc la complexité du problème et le nombre de règles nécessaires. Nous avons identifié huit situations perceptives différentes (comme présence d'un couloir présence d'un obstacle près sur la gauche ...). On associe à chacune de ces situations A_i la réaction appropriée sous forme d'un ensemble de règles R_i. Le système final est obtenu en réunissant la totalité des règles créées.

Ce type d'approche est classique. Les résultats que nous avons obtenus sont comparables aux approches à base de potentiels (présence de minima locaux et d'oscillations). Le formalisme flou de part ses propriétés d'approximateur universel a la possibilité de décrire la transformation recherchée mais l'expression sous forme de règles de notre compréhension du mécanisme de navigation ne permet pas de générer une solution acceptable.

Après avoir essayé de coder manuellement la transformation recherchée nous nous sommes tournés vers l'apprentissage automatique en contrôle. On distingue dans ce domaine trois grandes catégories d'approches selon le type d'informations disponibles :

1. l'apprentissage supervisé.

2. l'apprentissage distant.

3. l'apprentissage renforcé.

L'apprentissage distant est plus particulièrement conçu pour les systèmes devant apprendre à suivre une trajectoire de référence. Il n'est donc pas adapté à notre problème. Nous nous sommes plus particulièrement intéressés à l'apprentissage supervisé et à l'apprentissage renforcé.

Le principe de l'apprentissage supervisé est de permettre à un contrôleur d'apprendre un comportement à partir d'une base d'exemples représentatifs de la forme *(situation perçue, action correspondante)*. L'apprentissage se réalise généralement en trois étapes : le robot réalise tout d'abord en télé-opération la tâche pour laquelle on souhaite le programmer. Les couples d'exemples sont stockés et ensuite présentés à l'algorithme d'apprentissage. Le système résultant remplace finalement l'opérateur humain pour le contrôle du véhicule. De part les dimensions de l'espace de perception les exemples recueillis au cours de la télé-opération

sont rarement représentatifs de l'ensemble des situations possibles. L'algorithme d'apprentissage devra donc :

1. posséder de bonnes propriétés de généralisation face à une situation inconnue.

2. être incrémental. Les trois étapes décrites précédemment doivent être répétées chaque fois que le contrôleur est face à une situation qu'il ne sait pas traiter correctement. Il est alors nécessaire de lui montrer la solution. Pour des raisons de place mémoire, il n'est pas possible de stocker l'ensemble des exemples rencontrés depuis le début du système. Le contrôleur doit donc être en mesure d'apprendre à partir des nouveaux exemples sans oublier les précédents.

3. la complexité du problème que l'on cherche à résoudre est mal maîtrisée. Il est important que le formalisme choisi soit susceptible de représenter le plus grand nombre de fonctions possibles.

La plupart des approches robotique basées sur l'apprentissage supervisé font appel à une architecture neuronale très classique : le réseau à propagation unilatérale. Ces réseaux ne sont néanmoins pas incrémentaux et possèdent une architecture fixe limitant les fonctions qu'ils peuvent représenter (pour une architecture donnée). Nous avons réalisé une étude des principaux réseaux de neurones existants et avons sélectionné un réseau de type *Grow and Learn* possédant les propriétés que nous nous étions fixées. Ces propriétés ont pu être vérifiées lors d'expérimentations réalisées à l'aide d'un robot simulé, associant les mesures provenant directement des capteurs ultrasons aux commandes de vitesse linéaire et angulaire. Dans un deuxième temps, nous avons cherché à accélérer et fiabiliser l'apprentissage en réalisant une (ou plusieurs) transformations des données capteurs brutes permettant de réduire la dimension de l'espace et de faire ressortir leurs caractéristiques. Chacune de ces transformations réalisent un codage. Elles sont basées sur une analyse en composantes principales d'un ensemble de données représentatives d'une situation. Le principe de l'algorithme est le suivant :

- les données servant à l'apprentissage sont lues séquentiellement.

- on détermine pour chaque donnée lue s'il existe une transformation parmi celles déjà créées permettant de la coder (test réalisé en évaluant la perte d'information lors de l'opération de codage) .

- si aucune transformation existante n'est adaptée au codage d'une donnée, on se trouve alors dans une nouvelle situation perceptive et une nouvelle transformation est générée.

Cet algorithme permet de créer automatiquement des processus de perception adaptés chacun à une situation particulière. On associe ensuite à chacun de ces

processus de perception un réseau de type *Grow and Learn* permettant ainsi d'implémenter le comportement lui correspondant.

Nous nous sommes ensuite intéressés à l'apprentissage renforcé. Le principe de cet apprentissage est de laisser le contrôleur trouver lui-même quelle action doit être associée à chaque situation de manière à augmenter ses performances. Les performances sont mesurées par une fonction de renforcement fournie par le concepteur et codant les objectifs du système. Par opposition à l'approche supervisée, il n'y a pas de séparation entre la phase d'apprentissage et la phase d'exploitation. Le système est opérationnel (plus ou moins efficacement) dès le départ et peut apprendre tout au long de son existence. Parmi les différentes formes d'apprentissage renforcé, nous nous sommes plus particulièrement intéressés à l'apprentissage renforcé associatif, permettant d'apprendre le contrôleur tout en ajustant la fonction de renforcement par un mécanisme d'apprentissage supervisé (les exemples pour l'apprentissage supervisé n'étant plus fournis par un opérateur humain mais par le robot au cours de son évolution). Le temps d'apprentissage d'une telle approche est reconnu comme long par de nombreux travaux. De manière à le réduire, nous proposons d'utiliser la logique floue afin de coder des connaissances initiales, ne laissant ainsi pas le système apprendre de zéro. Cette connaissance initiale peut être fournie aussi bien pour le contrôleur que pour la fonction de renforcement. Comme nous l'avons indiqué précédemment, le système est composé de deux parties :

1. l'apprentissage du contrôleur.
2. l'ajustement de la fonction de renforcement.

Nous nous sommes plus particulièrement intéressés à la seconde partie. La fonction de renforcement est codée sous forme de règles floues qu'il s'agit d'être en mesure d'ajuster par un apprentissage supervisé. Cet apprentissage doit :

- présenter des propriétés de robustesse face aux données incohérentes (l'algorithme fournit par sa nature des données incohérentes jusqu'à la convergence).
- être incrémental.

Les algorithmes d'apprentissage supervisé pour les systèmes flous nécessitent la transformation préalable de l'ensemble des règles en réseaux de neurones. Ils ne possèdent pas les deux propriétés énoncées précédemment. De plus, ces algorithmes imposent un nombre fixe de règles, limitant ainsi fortement les modifications possibles lors de l'apprentissage sur la fonction représentée. Nous avons alors conçu un algorithme d'apprentissage incrémental pour système flou travaillant directement sur les règles :

- en généralisant leur partie condition.

– en adaptant leur conclusion.

– en créant de nouvelles règles.

En conclusion les deux approches mettant en œuvre un codage manuel du comportement présentent des problèmes similaires de minima locaux et d'oscillations. Ces problèmes sont liés à la manière d'aborder le problème. Le comportement de navigation est décomposé en un ensemble d'experts chargés de proposer une solution pour l'aspect de la situation dont ils sont spécialistes. Chacune de ces solutions est basée sur une hypothèse (un capteur situé sur le côté droit du robot indiquera une direction de déplacement vers la gauche s'il détecte un obstacle alors qu'il ne possède aucune information sur la présence éventuelle d'un obstacle sur la gauche). L'action finale est obtenue par fusion de ces hypothèses plus ou moins valides. Il n'y a donc pas de garantie qu'elle soit correcte. Le problème peut être abordé différemment. Il s'agit au lieu de générer un ensemble d'hypothèses de fournir un ensemble de certitudes (directions libres). La commande finale peut alors être fournie non plus par fusion mais par simple choix parmi les différentes propositions. Chaque proposition étant correcte la commande finale sera également correcte. Cette manière d'aborder le problème se retrouve dans les travaux de Borenstein (avec néanmoins quelques difficultés) et a été appliquée très récemment avec beaucoup de succès par la société Siemens.

Nos travaux en apprentissage automatique et plus particulièrement en apprentissage renforcé ne comportent pas un système complet et ne nous permettent donc de conclure sur la validité de l'approche. Nous tenons néanmoins à indiquer que cette voie semble très intéressante en offrant la possibilité au robot de s'adapter en permanence aux nouveaux types d'environnements (tout ne peut être prévu par avance) ainsi qu'à ses propres changements de fonctionnement (décalibration des capteurs par exemple). Nous pensons en particulier qu'il serait très intéressant de coupler des techniques d'apprentissage à des approches de type génération de certitude.

1.3 Organisation du rapport

Le chapitre 2 situe l'architecture fréquentielle et fonctionnelle utilisée dans le cadre de Mithra par rapport aux architectures classiques utilisées en robotique mobile. Les différents modules mis en place sont présentés : le contrôleur de véhicule la modélisation la perception et enfin la navigation. Les expériences réalisées sur le robot ROBUTER nous ont mis en évidence le manque de fiabilité du système de navigation orientant notre étude vers la navigation réactive.

Le chapitre 3 permet de définir les notions de comportement et de navigation réactive. Ces notions sont introduites par la différence existant entre le contrôle par plan et le contrôle par un processus en boucle fermée. La plupart des travaux

en robotique comportementale se sont tournés vers la génération de comportements complexes à partir de comportements élémentaires. Nous avons préféré modifier l'architecture de Mithra de manière à intégrer un seul comportement de navigation réactive (naviguer vers un but au milieu d'obstacles) chargé d'exécuter le résultat de la planification. Ce module de navigation fait l'objet de la suite de notre étude.

Le chapitre 4 est consacré aux approches de type champ de potentiels et champ de forces. Les méthodes de champs de potentielsΓprésentées grâce à une étude bibliographiqueΓnécessitent l'utilisation d'un modèle de l'environnement. Nous nous sommes plutôt orientés vers une approche de type champ de forces en adressant en particulier le problème de la gestion heuristique des minima locaux et de la prise en compte de la dynamique du robot. Les résultats expérimentaux obtenus indiquent la présence de plusieurs problèmes inhérents à ce type d'approches.

Les problèmes présentés au cours du chapitre précédent sont une caractéristique de la famille de fonctions utilisées dans les approches de type potentiel. De manière à ne pas nous limiter dans la forme des fonctions utilisées pour la représentation du comportementΓnous nous sommes tournés vers des approches mettant en œuvre un approximateur universel. Le chapitre 5 est consacré aux méthodes basées sur la logique floue. Les principes généraux de cette logique ainsi que l'utilisation dans le domaine du contrôle sont tout d'abord présentés. Notre approche est ensuite détaillée. Elle fait appel à une réduction de l'espace de perception et à la fusion de comportements élémentaires spécialisés dans la gestion de situations particulières. Les résultats obtenus sont similaires sur le plan des problèmes rencontrés aux approches de type champ de potentiels ou champ de forces.

Les approches présentées au cours des deux chapitres précédents ont en commun une implémentation figée de la navigation réactive réalisée par le concepteur du système. La suite du manuscrit est consacrée à la construction automatique du comportement.

Le chapitre 6 a pour objectif de présenter les motivations générales ainsi que les techniques mises en œuvre dans le domaine de l'apprentissage en contrôle.

Le chapitre 7 est consacré à l'apprentissage supervisé de comportements. Les principales approches en robotique sont basées sur l'utilisation de réseaux de neurones particuliers peu adaptés aux contraintes du domaine. Après une étude bibliographique des grandes familles neuronalesΓnous nous sommes tournés vers un réseau de type *Grow and Learn* répondant à nos spécifications. Nous avons tout d'abord réalisé une première série d'expériences mettant en jeu des données capteurs brutes. Nous nous sommes ensuite intéressés à des opérations de pré-traitement ayant pour but de réduire la dimension du problème et de faciliter l'apprentissage. Ces opérations permettent également de mettre en place un processus automatique de construction d'experts adaptés à des situations particulières de l'environnement.

Le chapitre 8 est consacré à l'apprentissage renforcé et ses applications en robotique. Nous nous sommes plus particulièrement intéressés dans ce domaine à l'apprentissage d'une fonction de renforcement représentée par un ensemble de règles floues. Les principaux algorithmes d'apprentissage pour systèmes flous ne répondent pas à nos contraintes. Nous avons donc conçu un algorithme adapté à notre problème : architecture ouverte (pas de limitation dans le type de fonctions pouvant être apprises) et incrémentalité. Cet algorithme ne nécessite pas de phase préalable de transformation du système flou en réseau de neurones. Il est basé sur une manipulation directe des règles.

En conclusion, les méthodes de type potentiel ou logique floue se heurtent à une approche inadaptée du problème. Il peut être reformulé de manière à se baser sur une génération de certitudes et non d'hypothèses. Les objectifs de l'apprentissage automatique en contrôle correspondent à nos besoins en robotique. Mais de nombreux travaux restent à accomplir dans ce domaine.

Chapitre 2

Le projet MITHRA

Nous allons au cours de ce chapitre présenter les objectifs et les différents problèmes attachés à la robotique mobile. Nous décrirons différentes architectures existantes et plus particulièrement celle utilisée au LIFIA et développée dans le cadre du projet Européen Mithra Eureka : EU 110.

2.1 La robotique mobile autonome

Les travaux en robotique ont pour but de concevoir et de construire des machines capables d'évoluer et d'interagir avec un environnement physique de manière à accomplir les différentes tâches pour lesquelles elles ont été créées. Dans le cadre de la robotique mobile, ces tâches peuvent être par exemple des tâches de manutention (robot de transport hospitalier FIRST), de surveillance (robot Mithra) ou de nettoyage (quais du métro, hôtels ...). La robotique mobile peut également avoir un rôle d'exploration et d'intervention en milieu hostile à l'homme (exploration sous-marine, véhicule planétaire).

Un robot évoluant dans un environnement réel est confronté à de multiples problèmes. Citons :

- le monde est vaste et dynamique. Contrairement à un bras manipulateur fixe, un robot mobile est susceptible d'évoluer dans un environnement vaste non contrôlé (présence d'êtres humains ...). Cela signifie que les objets (obstacles) peuvent se déplacer, apparaître ou disparaître. L'ensemble des situations ne peut pas être prévu par avance. Le robot devra être muni de capteurs lui permettant d'acquérir des informations sur son environnement proche (caméras vidéo, télémètres ultrasonique ou infrarouge, etc.)

- le robot a une connaissance imparfaite de son propre fonctionnement. Un système mobile automatique est un système complexe pour lequel il n'existe en général pas de modèle fiable. Le comportement du véhicule est décrit la plupart du temps par un système d'équations différentielles non linéaires

(contrainte de non holonomie)Γ dont les paramètres varient en fonctionΓ par exempleΓ du type de sol sur lequel le robot se déplace. De mêmeΓ les capteurs ultrasons réagissent différemment en fonction de paramètres tels que l'humidité de l'airΓ la températureΓ la forme des objets rencontrés et surtout leur constitution. De plusΓ il faut noter que cette interaction du robot avec son environnement est susceptible de se modifier avec le temps (décalibration des capteurs due à des vibrations ...).

Un robot mobile est donc un système mécaniqueΓ électronique et informatique comportant entre autre :

- des effecteurs (moteurs)Γ
- des capteurs (télémètresΓ etc).

Parmi l'ensemble des robots mobiles existantsΓ nous nous intéresserons au cours de cette étude à une sous-famille appelée *les robots mobiles autonomes*. Nous considérerons qu'un système est autonome si :

- il est capable d'accomplir les objectifs pour lesquels il a été conçu sans intervention humaineΓ
- il est capable de choisir ses actions afin d'accomplir ses objectifs.

Le problème posé est de déterminer à chaque instant quelle commande doit être envoyée aux effecteursΓ connaissant d'une part le but à accomplir et d'autre part les valeurs retournées par les différents capteurs. Il s'agit de déterminer les liens existants entre la perception et l'action connaissant les buts à atteindre. Ces buts peuvent être de haut niveauΓ tels que "nettoie la pièce"Γ "soit présent au point A à 11 h" ou encore "explore l'environnement". Ils peuvent être également de niveau inférieurΓ tels que "va au point (x, y)" ou "suit le mur M". Nous nous intéresserons plus particulièrement au cours de cette étude au choix des actions permettant à un robot d'atteindre un point de l'environnement spécifié par ses coordonnéesΓ tout en évitant les différents obstacles présents.

HistoriquementΓ comme le rappellent MalcolmΓ Smithers et Hallam [119]Γ les premières études ont été basées sur le cycle classique en intelligence artificielle : *perçoit, pense, agit* (voir figure 2.1).

La décomposition du problème en trois sous-problèmes a été à l'origine de nombreux travaux. Elle a été affinée mais également contestée (voir [29] par exemple). La présence de multiples modules attachés chacun à la résolution d'un sous-problème nécessite la mise en place d'une organisation permettant la construction d'un système complexe à partir de ces briques élémentaires. Cette organisation est appelée *architecture de contrôle*.

Les nombreuses architectures de contrôle proposées ont fait l'objet de plusieurs classifications ([48] par exemple). Il faut noter que dans le cadre de problèmes

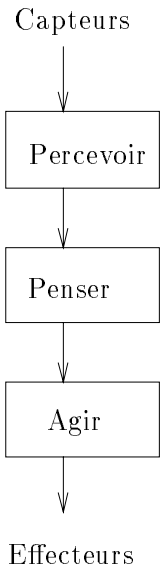


FIG. 2.1 - : *Cycle classique issu de l'intelligence artificielle*

complexes (tels que ceux auxquels nous nous intéressons)Γles architectures utilisées sont des architectures hiérarchiques dont nous allons exposer les principales caractéristiques.

2.2 Architectures hiérarchiques de contrôle

Nous allons au cours de ce paragraphe décrire les principaux types d'architectures hiérarchiques de contrôle. Ces architectures ayant pour but de contrôler un système évoluant dans un environnement réel et dynamiqueΓles contraintes de temps de réponse apparaissent clairement dans la conception de la plupart. Cette classification reprend celle proposée par Schopper [159]. Il faut noter que les systèmes réels ne sont bien souvent pas basés sur une seule catégorie de hiérarchie mais font appel à plusieurs d'entre elles de manière à exploiter les avantages respectifs.

Hiérarchies fréquentielles [49Γ130]. La fréquence d'exécution des modules décroît au fur et à mesure que l'on monte dans la hiérarchie. TraditionnellementΓle niveau le plus bas (et donc le plus rapide) est chargé du contrôle des moteurs. Le niveau le plus haut (et donc le plus lent) est chargé de la planification. Afin d'éviter tout problème d'instabilitéΓla différence de fréquence entre chaque niveau doit être importante (usuellementΓla vitesse est divisée par un facteur compris entre 3 et 10). Ce principe sera illustré au paragraphe 2.3.

Hiérarchie d'abstraction de données (ou fonctionnelle) [40Γ37]. La hiérarchie d'abstraction de données utilise les mêmes concepts que la programmation orientée objet (encapsulation ...). Il s'agit de fournir un ensemble de modules ayant leurs compétences propres. Chaque module n'a besoin de connaître que la fonctionnalité et non le fonctionnement ou l'implémentation d'un autre module pour interagir avec lui. La communication entre les différentes composantes peut être réalisée par une structure client-serveur par exemple.

Hiérarchie d'abstraction de représentations [148Γ71]. La notion d'abstraction de données est une notion fréquemment utilisée en intelligence artificielle. Le principe est d'ignorer pour un niveau supérieurΓune partie des informations provenant d'un niveau inférieur. On obtient ainsi au fil des niveaux une représentation simplifiée du problème que l'on peut alors résoudre plus rapidement.

Hiérarchie de résolution [47Γ38]. Ces hiérarchies mettent en œuvre un ensemble de modules de fonctionnalités identiques travaillant en parallèle sur différents niveaux de représentation d'une même donnée (cela peut être un système de planification de chemins travaillant sur différentes échelles d'une carte représentant un même terrain). Contrairement aux hiérarchies d'abstraction de représentationsΓle passage d'un niveau inférieur à un niveau supérieur n'est pas obtenu en ignorant certaines données mais en les combinant (par un opérateur de moyenne par exemple).

Hiérarchie de compétences [27]. Contrairement aux hiérarchies présentées précédemment et que l'on peut qualifier d'*horizontales*Γla hiérarchie de compétences propose une décomposition *verticale* du contrôle. Un comportement est défini comme un ensemble de réponses observables générées par un système face à un ensemble de stimuli internes ou produits par l'environnement. Un niveau de compétence est une spécification informelle d'une classe voulue de comportements pour un robot (évitement de collisionsΓexploration de l'environnement ...). Le premier niveau envoie des commandes au véhicule. Les niveaux suivants influent sur le fonctionnement des niveaux inférieurs.

Nous avons développé au LIFIA dans le cadre du projet MITHRA un système de contrôle pour un robot mobile de surveillance. Ce système nous a permis de réaliser nos premières expériences et de mettre en évidence certains problèmes liés à la solution proposée. L'étude de ces problèmes a conduit à la deuxième partie de notre travail (la navigation réactive) exposée dans ce manuscrit. Nous allons maintenant revenir plus en détail sur ce système.

2.3 Le système MITHRA

Le projet EUREKA EU 110 : Mithra 110 était un projet de recherche Européen mené en collaboration avec des partenaires de la région alpine FrançaiseΓ Italienne et Suisse. Ce projet s'est déroulé entre 1988 et 1991 et a eu pour but la démonstration d'une famille de robots autonomes de surveillance et de première intervention.

L'architecture proposée (voir figure 2.2) est basée sur une coopération entre la perception et l'action. Elle fait appel à une hiérarchie de type fréquentielle mais aussi fonctionnelle. Elle se décompose en cinq couches [50] :

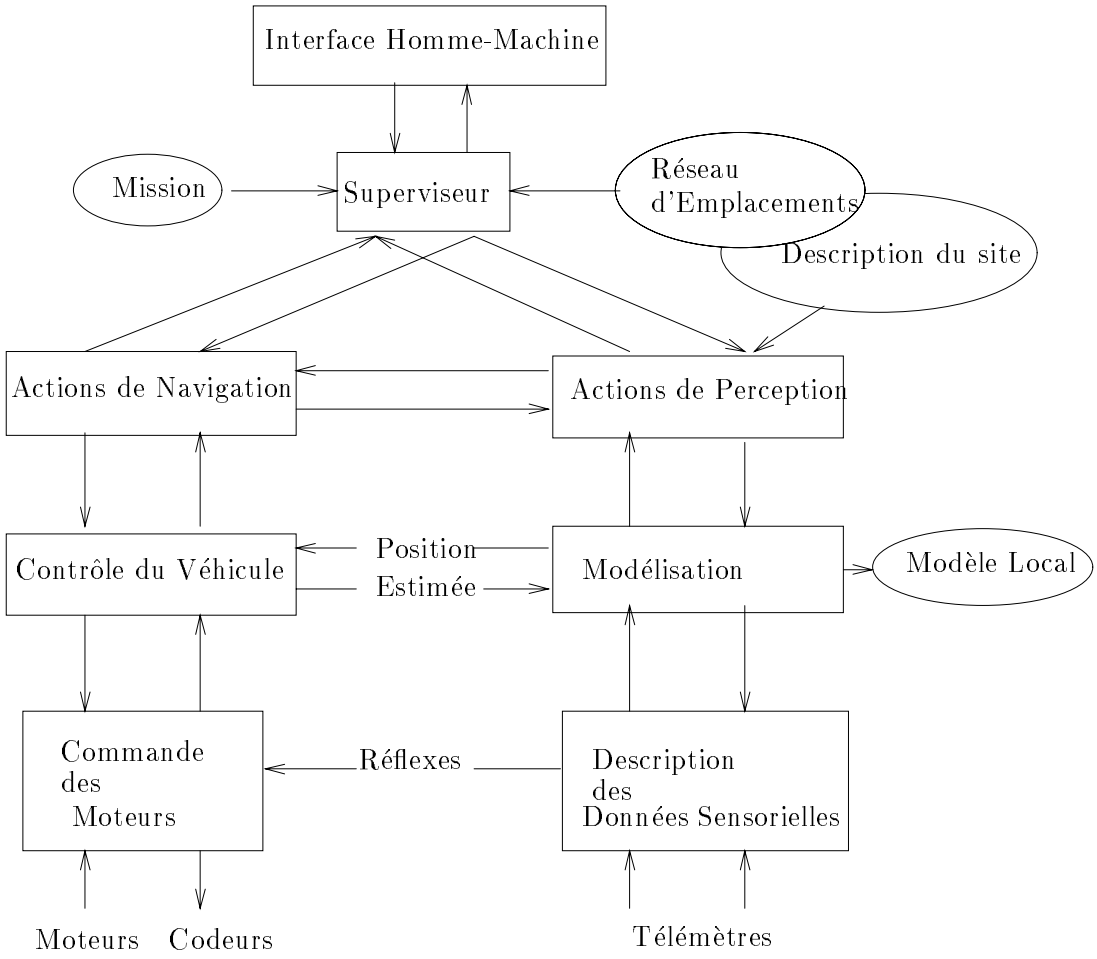


FIG. 2.2 - : *L'architecture du système Mithra*

– le contrôleur des moteurs et la gestion des capteurs pour la couche de plus bas niveauΓ

- les procédures de modélisation de l'environnement et les procédures de contrôle du véhiculeΓ
- les procédures de perception et de navigationΓ
- le superviseur (raisonnement symbolique pour la planification et le contrôle d'exécution)Γ
- l'interface homme-machine.

Cette architecture met en avant une collaboration entre des aptitudes de type algorithmique (les trois premières couches) et des connaissances de type heuristique (le superviseur). L'interface homme-machine ainsi que le système de raisonnement symbolique ont fait l'objet d'un travail de thèse [36Γ35]. Nous allons plus particulièrement nous intéresser aux couches basses du système.

2.3.1 Le contrôleur de véhicule

Le système robotique auquel nous nous sommes intéressés est constitué d'une plate-forme munie de deux roues motrices indépendantes et de deux roues "folles" (voir figure 2.3). Le mouvement de rotation de l'ensemble est obtenu en appliquant une vitesse de rotation différente à chacune des deux roues motrices. Un bras robotique nécessite un contrôleur de bras pour coordonner les différents mouvements des moteurs afin d'obtenir la trajectoire souhaitée. De mêmeΓla plate-forme que nous utilisons nécessite un *contrôleur de mobilité* afin de coordonner les mouvements des deux roues et obtenir ainsi les trajectoires désirées.

Les ordres principaux acceptés par le module sont les suivants :

- Move (distanceΓvitesseΓaccélération)Γ
- Turn (distanceΓvitesseΓaccélération)Γ
- StopΓ
- GetEstPosΓ
- SetEstPos (Position).

Les deux dernières commandes ont pour but de lire ou de fournir une position estimée du robot exprimée dans un repère absolu attaché au site. Les trois premières commandes permettent de contrôler les déplacements linéaires et angulaires en termes de distanceΓvitesse et accélération. Les deux types de mouvements peuvent être combinés afin d'obtenir des trajectoires complexes. Le profil de vitesse utilisé pour les ordres de translation et de rotation est le profil classique en trapèze représenté figure 2.4.

FIG. 2.3 - : *Le robot mobile du LIFIA*

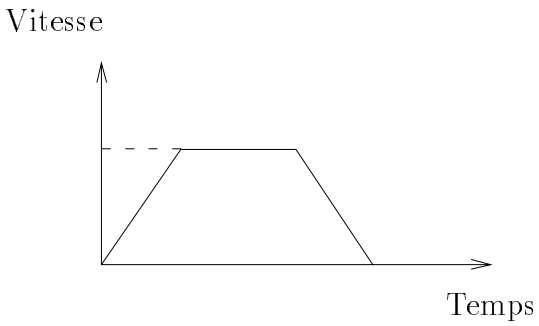


FIG. 2.4 - : *Profil de vitesse pour un ordre de translation ou de rotation.*

Le robot doit se déplacer dans un univers dynamique non maîtrisé. De manière à pouvoir réagir immédiatement à toute situation nouvelleΓtout ordre en cours d'exécution doit pouvoir instantanément être remplacé par une nouvelle directive. A la réception de l'ordreΓle trapèze correspondant à la situation courante et aux consignes fournies est déterminé (exemple figure 2.5). On calcule ensuite la liste $l = (d_1, \ldots, d_n)$ associée avec :

$d_1 = 0$
$d_i =$ distance devant être parcourue au bout du temps $i \times \Delta_t$
$d_n =$ distance totale à parcourir passée en paramètre de l'ordre reçu

Ces distances d_i sont envoyées régulièrement tous les Δ_t à un contrôleur de bas niveau chargé de les exécuter. Cet envoi se poursuit jusqu'à exécution complète de l'ordre associé ou jusqu'à réception d'un nouvel ordreΓprovoquant la génération d'une nouvelle liste remplaçant la liste courante.

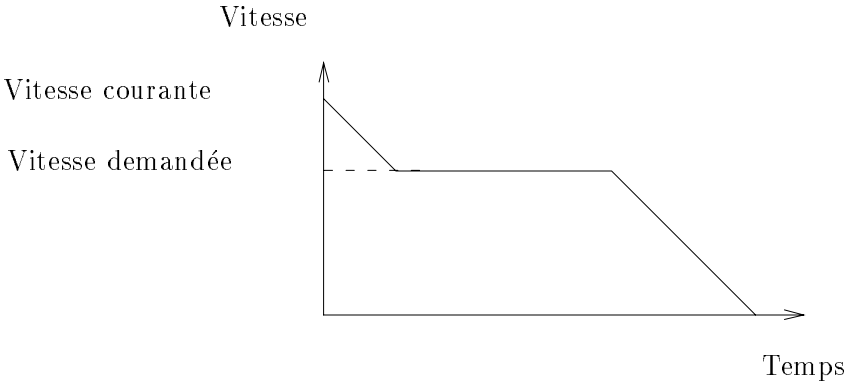


FIG. 2.5 - : *Réception d'un nouvel ordre. Le profil est calculé de manière à ralentir le robot jusqu'à la nouvelle vitesse commandée puis de l'arrêter à la distance souhaitée.*

Le schéma de principe du contrôleur bas niveau est fourni figure 2.6. Les contrôleurs de rotation et de translation sont réalisés à l'aide de deux PID chargés d'asservir respectivement le cap du robot et la distance linéaire parcourue. Ils sont indépendants et peuvent travailler simultanémentΓautorisant des mouvements combinés de translations et de rotations.

Les commandes fournies par ces deux modules sont combinées et transformées en déplacements Δ_{rd} et Δ_{rg} pour la roue droite et gauche respectivement à l'aide du *générateur de commandes moteurs* :

$$\begin{cases} \Delta_{rg} = \Delta_l + r\Delta_\theta \\ \Delta_{rd} = \Delta_l - r\Delta_\theta \end{cases}$$

Δ_l et Δ_θ représentent la commande reçue en terme de déplacement linéaire et angulaire. r représente la distance séparant le centre de rotation du véhicule des

roues. Δ_{rg} et Δ_{rd} sont ensuite fournis aux contrôleurs de moteurs et exécutés grâce à deux autres régulateurs PID fournis avec le robot.

Les déplacements effectivement réalisés par les roues sont mesurés par deux odomètres puis transformés en déplacements relatifs linéaires et angulaires par le module de calcul de l'état cinématique :

$$\begin{cases} \Delta_l = \frac{\Delta_{rd}+\Delta_{rg}}{2} \\ \Delta_\theta = \frac{\Delta_{rd}-\Delta_{rg}}{2r} \end{cases}$$

La sommation au cours du temps de ces déplacements élémentaires permet d'estimer la position et l'orientation du robot dans un repère absolu [53]:

$$\begin{cases} x \leftarrow x + \Delta_l * cos(\theta) \\ y \leftarrow y + \Delta_l * sin(\theta) \end{cases}$$

Le premier module lié à l'action est maintenant en place. Avant de nous tourner vers la perception nous allons tout d'abord rappeler les enjeux de la représentation de l'environnement.

2.3.2 Représentation de l'environnement

La représentation de l'environnement dans le cadre d'un véhicule autonome a un double but :

1. Il s'agit tout d'abord de permettre au robot de maintenir sa position estimée au cours des différents déplacements. Comme nous l'avons vu précédemment cette position est calculée grâce aux encodeurs situés sur les deux roues. L'erreur commise étant additive il est nécessaire de relocaliser périodiquement le véhicule grâce à de multiples observations de l'environnement.

2. Il s'agit ensuite de déterminer les régions libres et les régions occupées de l'espace afin de permettre au robot de se déplacer sans entrer en collision avec les différents obstacles présents.

Nous allons revenir sur ces deux aspects particuliers.

La relocalisation

De manière à se relocaliser un véhicule a besoin de construire et de maintenir une représentation interne de son environnement. Il existe pour ce faire deux grandes familles de modèles :

- Les modèles paramétriques.
- Les modèles à base de grilles.

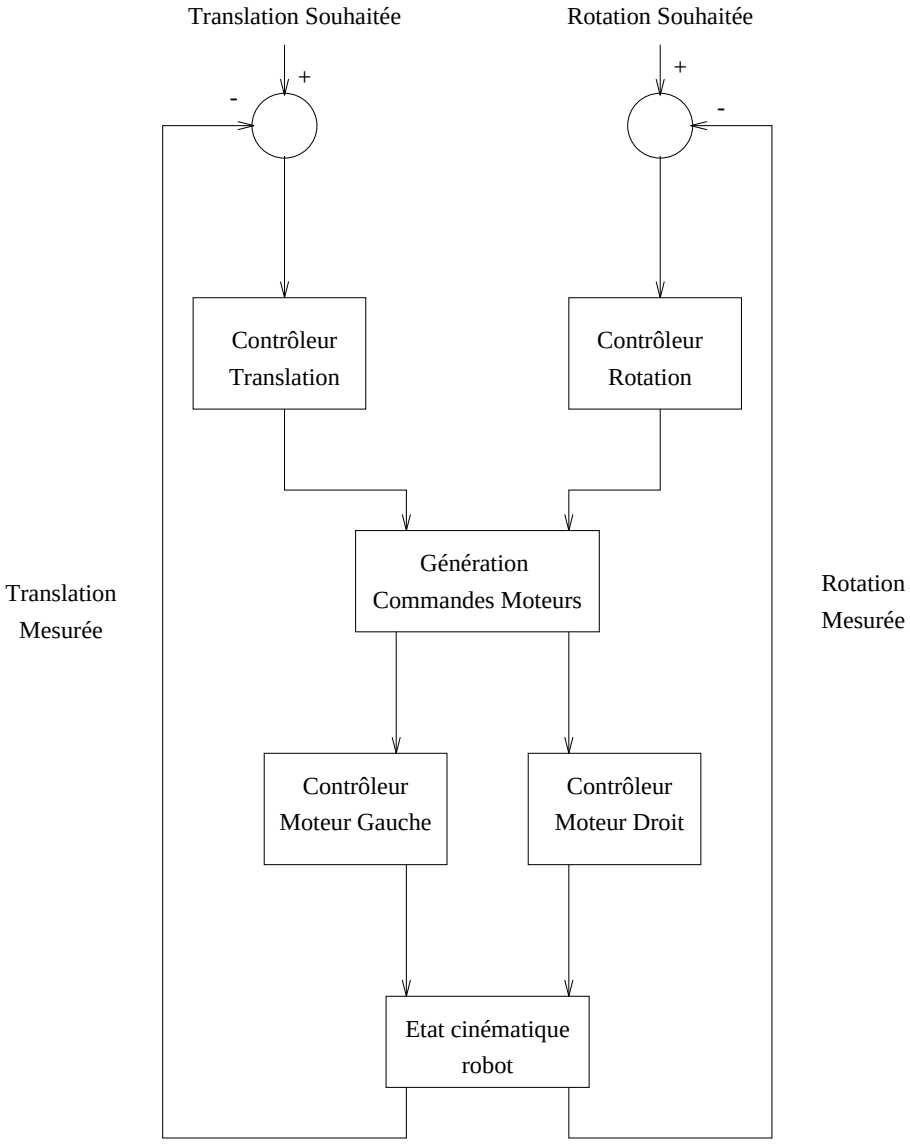


FIG. 2.6 - : *Le contrôleur de véhicule*

La représentation interne de l'environnement sous forme paramétrique a été développée dans le cadre de l'utilisation de capteurs ultrasons [51Γ52]. Nous reviendrons plus en détail sur cette approche au cours du paragraphe 2.3.3.

La représentation interne de l'environnement sous forme de grilles a été initialement conçue par Elfes et Moravec [124Γ56]. L'objectif est de représenter l'environnement à l'aide d'une grille régulière et d'assigner à chacune des cellules une probabilité indiquant si la surface correspondante est vide ou occupée. Elfes a utilisé cette approche afin de résoudre le problème de la localisation (et non de la relocalisation) par détermination du mouvement [1] [56] :

- le robot est à l'arrêt. Il prend une vue complète de l'environnement grâce à l'ensemble de ses capteurs ultrasonsΓ
- le véhicule ensuite se déplace et s'arrête à nouveau. Une nouvelle vue est priseΓ
- l'algorithme cherche la transformation (rotation plus translation) permettant de superposer les deux grilles obtenues. La connaissance de cette transformation permet alors de déduire le mouvement réalisé par le véhicule et de mettre à jour ainsi sa position estimée.

Il faut noter que cette approche est assez coûteuse en temps de calcul (complexité n^5 pour une grille de n cellulesΓpouvant être ramenée à une complexité n par l'utilisation de techniques multi-résolutions). Elle nécessite de plus l'arrêt très fréquent du robot.

De récents travaux [157Γ156Γ158] ont abordé le problème de la relocalisation "en vol". Cette méthode fait appel à l'utilisation d'une grille globale représentant la totalité de l'environnement d'évolutionΓainsi que d'une grille localeΓcentrée autour du robot. La grille globale est mise à jour périodiquement grâce à la grille locale. La relocalisation est réalisée par la mise en correspondance de segments extraits dans chacune des deux grilles à l'aide de transformées de Hough et par l'utilisation d'un filtre de Kalman. Le coût de calcul reste assez élevé mais les résultats sont prometteurs. Il faut noter que ces derniers travaux n'étaient pas réalisés lors du projet Mithra.

De manière à réduire la complexité de calcul lors de la mise à jour de la grilleΓ Borenstein et Koren ont proposé une approche simplifiée appelée "histogram grid" [26]. Cette approche permet de construire une représentation correcte de l'environnement [138] mais n'a jusqu'à présent été utilisée que pour résoudre le problème de l'évitement d'obstacles et non celui de la relocalisation.

Dans le cadre du projet MithraΓla capacité mémoire ainsi que la puissance de calcul sont limitées. Le robot doit de plus avoir la possibilité de se relocaliser sans avoir à s'arrêter. Ceci conduit à l'utilisation d'une représentation paramétrique de l'environnement.

1. En anglais, problème du *motion solving*

Nous allons maintenant aborder le problème de la représentation de l'environnement dans le cadre de la navigation.

La navigation

Nous considérerons dans cette partie le problème de la planification de chemins (différent de la planification de trajectoires). Historiquement, la planification de chemins correspond aux premiers travaux de recherche effectués dans le domaine de la planification de mouvement et se limite à l'aspect géométrique du problème. La planification de trajectoire, quant à elle, introduit la dimension temporelle et permet de prendre en compte des obstacles mobiles ainsi que des contraintes de nature dynamique auxquelles peut être soumis le robot (force, vitesse, accélération).

Avant de poursuivre, nous allons définir la notion d'espace des configurations [178, 114]. Le principe de l'espace des configurations est de reformuler le problème de planification de chemins dans un nouvel espace où le robot est représenté sous forme d'un point. Cet espace est créé en associant à chacun des axes un des paramètres permettant d'identifier de manière unique la situation du robot (par exemple, position (x, y) et orientation θ dans le cas d'un robot mobile).

Le choix d'une méthode de planification de chemins est guidé par deux questions :

1. Quel type d'espace utilisé : l'espace de travail ou l'espace des configurations ?

2. Quel type de méthodes : des méthodes exactes ou des méthodes approchées ?

Les méthodes exactes sont basées sur une exploitation complète de la description de l'environnement. Par opposition, les méthodes approchées réalisent tout d'abord une discrétisation de l'environnement sous forme de grilles régulières ou irrégulières [2]. L'espace libre ainsi représenté est un sous-ensemble de l'espace libre réel. Alors que les méthodes exactes sont susceptibles d'être complètes (si un chemin existe, alors la méthode le trouve), les méthodes approchées ne le sont jamais. Elles ne peuvent être au mieux que complètes pour la résolution choisie (finesse de la discrétisation).

Nous allons tout d'abord aborder le problème de la complexité. Les principaux résultats théoriques sont les suivants :

- Un chemin libre dans un espace des configurations de dimension m, l'espace libre étant un ensemble défini par n polynômes de degré maximal d, peut être calculé par un algorithme dont la complexité en temps est exponentielle par rapport à m et polynomiale par rapport à n (complexité géométrique) et d (complexité algébrique) [160].

2. quadtrees

- La planification du mouvement d'un objet rigide se déplaçant en translation sans effectuer de rotation dans un espace en trois dimensions au milieu d'un nombre arbitraire d'obstacles mobiles pouvant effectuer des translations et des rotations est un problème PEspace dur si la vitesse de l'objet est bornée et NP dur sinon [139].

- La planification du mouvement d'un point dans un plan (la vitesse du point étant bornée) au milieu d'un nombre arbitraire d'obstacles convexes polygonaux se déplaçant à vitesse constante sans effectuer de rotations est un problème NP dur [31].

La complexité du problème est donc une fonction du degré algébrique de la description de l'environnement. De manière à simplifier le problème et obtenir des temps d'exécution raisonnablesΓles chercheurs se sont intéressés à des obstacles polygonaux (dans l'espace des configurations).

Les principales approches existantes peuvent être classées en trois grandes catégories [109] :

- Les méthodes de type *squelette*. Le principe consiste à représenter la connectivité de l'espace libre dans un réseau unidimensionnel de courbes appelées *squelettes*. les différentes méthodes se distinguent par le type de squelettes utilisés (graphe de visibilitéΓdiagramme de VoronoïΓ "freeway net"Γ "road maps" ...).

- La décomposition en cellules. Cette approche est parmi celles les plus étudiées. Il s'agit de décomposer l'environnement du robot en régions élémentaires (appelées *cellules*) de telle sorte qu'un chemin entre deux configurations d'une même région soit trivial à générer (en imposant par exemple la propriété de connexité. Le chemin est alors une simple ligne droite joignant les deux points). On construit et on parcourt ensuite le graphe représentant la relation d'adjacence entre les cellules.

- Les méthodes de potentiels. Le robotΓ représenté par un point dans l'espace des configurationsΓ est considéré comme une particule se déplaçant sous l'influence d'un champ de potentiels artificiels créé par la configuration du but et des obstacles. Ces méthodes furent développées à l'origine pour l'évitement en ligne d'obstacles par un bras de robot découvrant son environnement au fur et à mesure [97]. On parle à ce titre de méthodes *locales* par opposition aux méthodes précédentes qualifiées de *globales*. L'avantage principal des méthodes de type potentiel est leur rapidité de calcul permettant de les utiliser en ligne (nous reviendrons plus en détail sur ces approches au cours du chapitre 4). L'inconvénient principal est que le champ est susceptible de contenir des minima locaux. C'est pour essayer de palier ce problème que les méthodes de potentiels ont évolué vers des méthodes

globales grâce à l'utilisation de la connaissance a priori de la totalité de l'environnement. On peut citer dans ce domaine les travaux de Barraquand et Latombe [17]Γproposant de "sortir" des minima locaux par utilisation de déplacements aléatoires de type BrownienΓainsi que les travaux de Koditschek et Rimon [98Γ99] ou plus récemment Grupen et Connolly [43Γ44] visant à fournir des fonctions de potentiels dont le seul minimum local est le but à atteindre.

Les approches brièvement décrites précédemment utilisent en majorité l'espace des configurations. Dans le cas d'un véhicule mobile rigide (pas de remorque par exemple)Γcet espace est de dimension 3. Il peut être calculé de manière approchée (cas le plus fréquent) en discrétisant l'ensemble des directions possibles pour le robot. Il peut également être calculé de manière exact [15].

Le robot utilisé dans le cadre de Mithra est un véhicule à chenille. Pour des raisons de fiabilité de la localisation et de coût énergétiqueΓles mouvements de translation et de rotation sont séparés. Les trajectoires du robot sont des lignes brisées et sont facilement modélisables dans l'espace de travail. De manière à limiter le coût de calculΓnous avons décidé de réaliser la planification dans cet espace.

Comme nous l'avons indiqué au cours de la section précédenteΓnous avons choisi une représentation paramétrique pour résoudre le problème de la relocalisation. Nous utiliserons cette même représentation pour la navigation.

La représentation de l'environnement étant choisieΓnous allons maintenant détailler les modules de modélisationΓde perception et de navigation.

2.3.3 La modélisation

Nous ne décrirons dans cette section que les principes généraux mis en œuvre. Cette réalisation reprend les travaux de Crowley [51Γ52]. Le principe est donné figure 2.8. Les capteurs utilisés sont des capteurs ultrasons (voir leur répartition figure 2.7). Les données sensorielles brutes sont dans un premier temps traitées de manière à produire une description de l'environnement proche sous forme de primitives géométriques.

Ces capteurs fournissent une mesure de profondeur peu préciseΓdépendant de paramètres tels que le type des matériaux constituant les obstaclesΓl'angle du capteur par rapport à la surface de réflexionΓmais aussi la température de l'air. De manière à fiabiliser les résultatsΓil est nécessaire de rechercher une corrélation entre toutes ces données. Pour ce faireΓnous transformons les mesures capteurs en points dans le repère absolu (grâce à la position estimée du véhicule) et nous recherchons lesquels sont susceptibles d'être alignés pour former un segment (il en faut trois au minimum pour qu'un segment soit créé). Les primitives géométriques retenues pour la constitution du modèle sont donc les segments de droite.

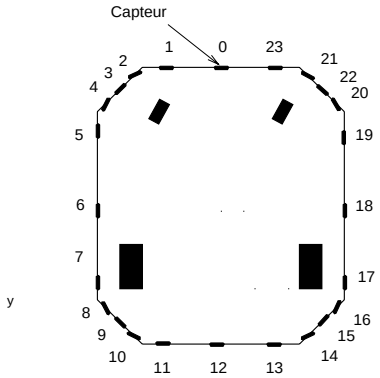


FIG. 2.7 - : *Répartition des capteurs ultrasons sur la ceinture du robot*

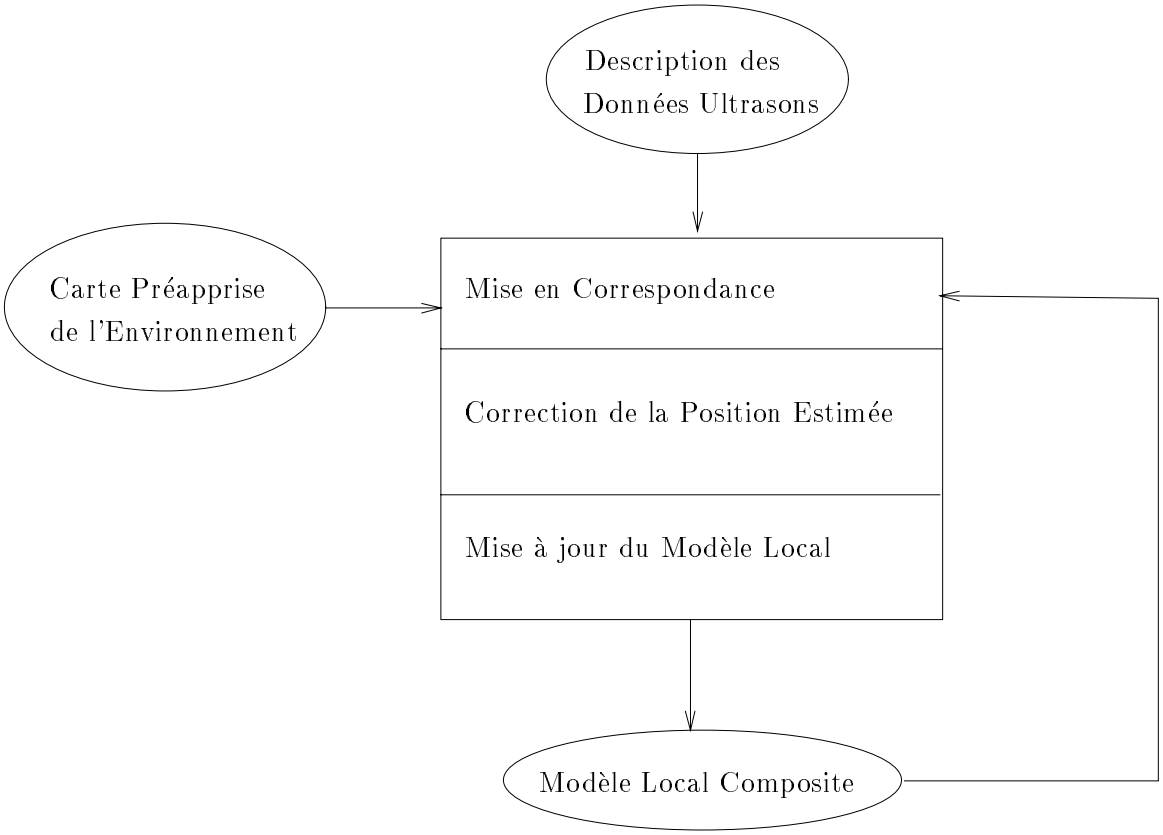


FIG. 2.8 - : *Principe de mise à jour du modèle local de l'environnement libre*

Au fur et à mesure de leur créationΓces primitives sont ensuite envoyées vers le processus de modélisation proprement dit dont la tâche est de maintenir le modèle local composite. Ce modèle est dit local car il n'intègre que les données situées dans un environnement proche du véhicule. Il est dit composite car il est formé d'observations réalisées en différents points de l'espaceΓet pouvant éventuellement provenir de plusieurs sources différentes (dans notre casΓles capteurs ultrasons sont l'unique source). Les segments de droites provenant des capteurs sont mis en correspondance avec ceux déjà présents dans le modèle. Les nouveaux segments sont ajoutésΓceux déjà présents sont mis à jour (par utilisation d'un filtre de Kalman). Cette mise à jour permet d'affiner le modèle mais entraîne également une correction de la position estimée du robot.

Le temps de calcul nécessaire à l'exécution d'un cycle de modélisation dépend du nombre de segments présents dans le modèle (complexité linéaire par exemple dans le cas de la mise en correspondance entre un segment perçu et un segment du modèle). De manière à garder des temps de réponse correctsΓtout segment n'ayant pas été observé depuis un nombre fixé de cycles est jugé obsolète et éliminé. Ce mécanisme permet également de conserver la propriété de localité du modèle.

Pour finirΓle processus de modélisation peut avoir recours s'il le désire à une carte pré-apprise de l'environnement contenant les obstacles principaux. Cette carte est saisie grâce à un outil de CAO [35].

Le véhicule est maintenant capable de maintenir sa position estimée et a une connaissance de l'espace libre l'entourant. Cette connaissance est sous forme d'une liste de segments de droite. Il s'agit maintenant de construire les outils permettant de l'exploiter pour pouvoir naviguer.

2.3.4 La perception

Le module *perception* a pour but d'extraire du modèle local les informations nécessaires au robot :

1. pour détecter les obstacles.
2. pour calculer un chemin permettant de rejoindre le but en évitant les obstacles.

La détection d'obstacles (fonction *freepath*)

Le robot effectuant soit des mouvements de rotationΓsoit des mouvements de translationΓnous avons créé deux fonctions différentes spécialisées dans chaque type de mouvements.

Lors d'un déplacement en translationΓla partie de l'espace de travail couvert par le robot correspond à un rectangle (voir figure 2.9) dont on va tester l'intersection avec chaque segment du modèle local. On peut remarquer que la complexité

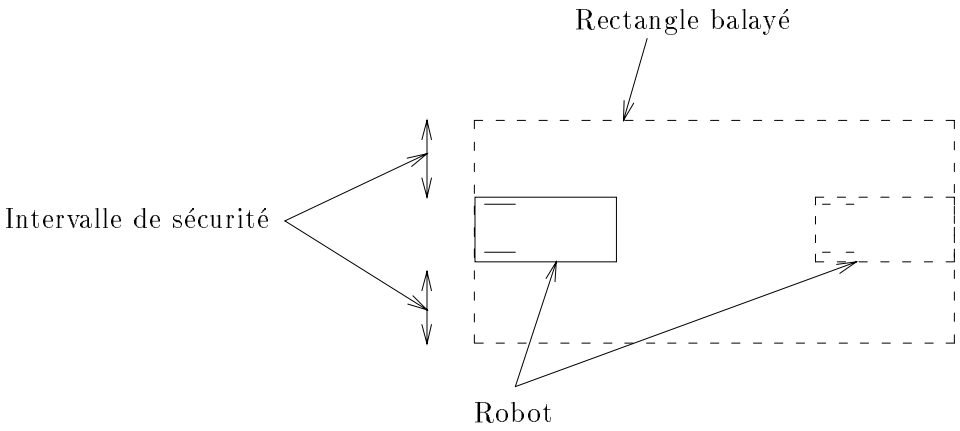


FIG. 2.9 - : *Espace couvert par le robot lors d'une translation*

de l'analyse nécessaire n'est pas la même suivant les cas. En particulierΓsi une des extrémités du segment considéré se situe à l'intérieur du rectangleΓla déduction est immédiate. Nous avons ainsi répertorié les cas où l'intersection est sûreΓceux où il n'y a pas d'intersectionΓet enfin les cas ambigus où une analyse plus poussée est nécessaire. L'espace est découpé en 9 zones distinctes (figure 2.10).

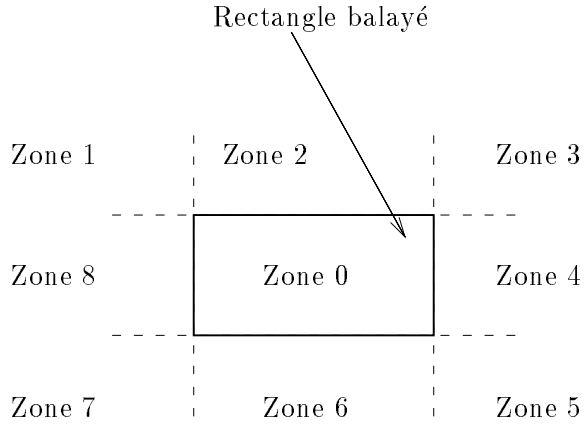


FIG. 2.10 - : *Découpage de l'espace en* 9 *zones*

Les trois cas possibles sont représentés sur la figure 2.11. Les cas d'ambiguïtés sont résolus par un calcul d'intersection entre deux droites. L'algorithme obtenu est linéaire par rapport au nombre de segments présents dans le modèle.

Lors d'un mouvement de rotationΓl'espace balayé par le véhicule n'est pas de forme simple. Nous l'avons approché par deux portions de disques. L'algorithme

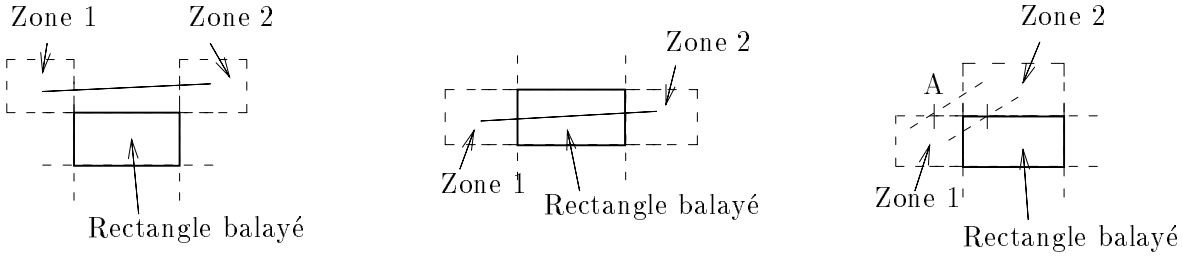


FIG. 2.11 - : *La figure de gauche représente deux zones dans lesquelles il n'y a pas intersection. La figure centrale représente deux zones dans lesquelles il y a toujours intersection. La figure de droite représente un cas d'ambiguïté. Il suffit alors de calculer la position du point A.*

consiste à étudier (voir figure 2.12):

- l'intersection entre un segment et la partie supérieure du disqueΓ
- l'intersection entre un segment et un triangle.

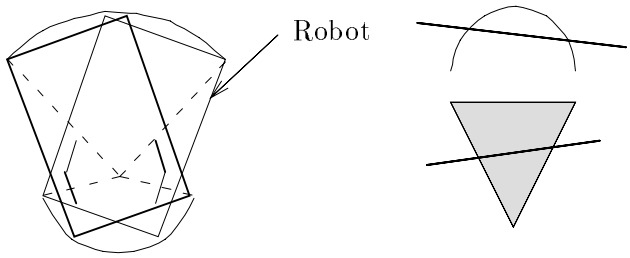


FIG. 2.12 - : *La figure de gauche représente l'approximation de la surface balayée par le robot à deux portions de disques. L'algorithme teste donc l'intersection d'un segment avec un triangle et la partie supérieure du disque (figure de droite)*

Le système est maintenant en mesure de détecter les obstacles. Nous allons nous intéresser au calcul d'un chemin d'évitement.

Recherche d'un chemin libre (fonction *findpath*)

Le but de cette fonction est d'élaborer un chemin libre permettant au robot de contourner les obstacles présents. La recherche d'une route au niveau carte (recherche d'un chemin à travers une série de pièces et couloirs) est résolue par le superviseur grâce à un réseau d'emplacements [36]. Il s'agit de planifier un chemin entre deux de ces emplacementsΓsur une "courte" distance.

Comme nous l'avons précisé au cours de la section 2.3.2Γla procédure que nous avons proposée utilise directement l'espace de travail et donc l'ensemble des segments présents dans le modèle local. Le but peut être en dehors de l'horizon visible. Toute partie située en dehors de celui-ci étant considérée comme libreΓle chemin construit peut être remis en cause lors de son parcours.

L'algorithme mis en œuvre est de type *squelette* (voir section 2.3.2). Le principe général est le suivant : on regarde grâce à la fonction *freepath* si le chemin entre la position courante du véhicule et le but est libre. Si la réponse est positiveΓalors l'algorithme est terminé. Dans le cas contraireΓon construit deux points d'évitement de part et d'autre de l'obstacle et on appelle récursivement l'algorithme sur les quatre segments ainsi créés (voir figure 2.13).

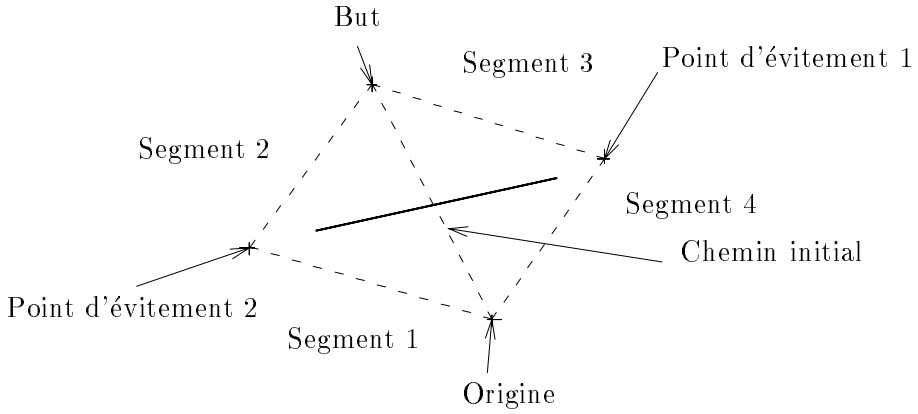


FIG. 2.13 - : *Principe de l'algorithme de recherche d'un chemin libre.* Findpath *est appelé récursivement sur les* 4 *segments créés*

Cette méthode conduit à construire et à parcourir un arbre décrivant toutes les possibilités de contournement des obstacles présents (pour une méthode de construction de points d'évitement donnée). De manière à réduire le coût de cette approcheΓnous avons proposé de rejeter directement certaines solutionsΓcelles-ci pouvant par ailleurs être correctes. Ceci est justifié par la "faible" distance devant être parcourue lors d'une commande de navigation (voir le début du paragraphe) et donc la relative simplicité des obstacles pouvant être présents.

Construction d'un point d'évitement

Il n'existe pas de manière unique de construire un tel point. Nous avons choisi la suivante : un point d'évitement est la projection d'une des extrémités du segment perpendiculairement à la direction origine - cibleΓà une distance de sécurité d de cette extrémité (figure 2.14).

Ceci fournit pour chaque segment du modèle quatre points. Ne souhaitant pas

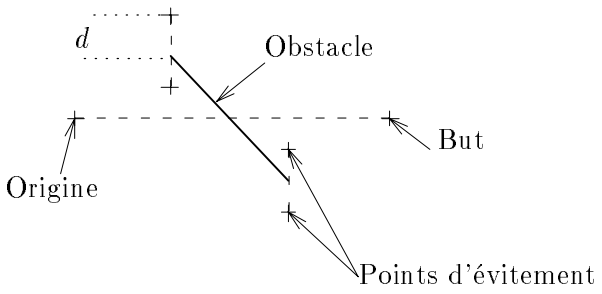


FIG. 2.14 - : *Construction des points d'évitement*

réaliser une exploration exhaustiveΓnous ne les retiendrons pas tous. Le choix s'effectue en deux temps :

- choix du point de contournement. Il faut déterminer par rapport à laquelle de ses extrémités le segment va être contournéΓ
- choix du point de passage. Il s'agit de déterminer lequel des deux points d'évitement choisir pour cette extrémité.

La philosophie générale est la suivante : on fixe avant le début de la recherche un sens de parcours (droite ou gauche) que l'on garde ensuite pour la totalité du chemin. La notion de *sens* pour un point se définie par rapport à deux points de référence (voir figure 2.15). Un point G sera dit à gauche de deux points R et I

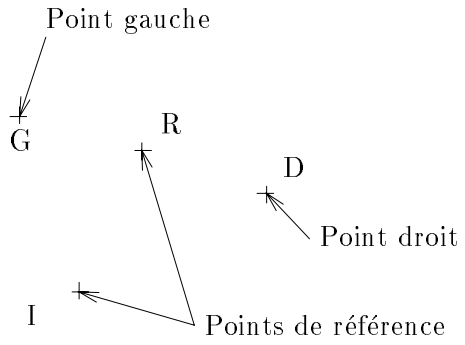


FIG. 2.15 - : *Définition du sens d'un point par rapport à deux autres*

si et seulement si ($\vec{k}$ étant le vecteur de base orthogonal au plan contenant les points) :

$$(\vec{RI} \wedge \vec{GI}).\vec{k} \geq 0$$

De mêmeΓun point D sera dit à droite de R et I si et seulement si :

$$(\vec{RI} \wedge \vec{DI}).\vec{k} < 0$$

Revenons sur les deux choix :

- choix du point de contournement : on choisit le point de contournement conformément au sens défini pour la recherche. La détection du sens est réalisée par rapport au point central du segment et au point de départΓ
- choix du point de passage : On choisit le point vérifiant la relation $\vec{AP}.\vec{AB} \leq 0$ (voir figure 2.16). Ceci permet de sélectionner le point de passage situé au delà du segment et a priori le plus accessible.

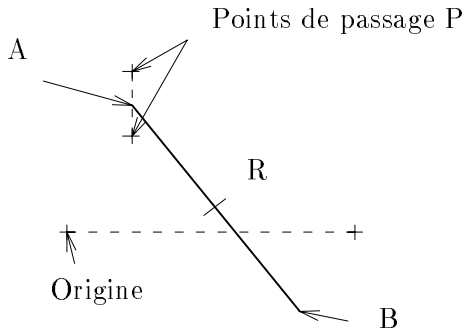


FIG. 2.16 - : *Choix du point de passage*

La méthode de construction et de choix d'un point de passage n'apporte aucune garantie sur la validité du chemin construit. L'une des extrémités du segment contourné peut en effet se trouver à l'intérieur du rectangle d'encombrement défini pour la fonction *freepath*. Il faut alors avoir recours à une analyse de l'environnement immédiat de l'obstacle.

Nous allons dans un premier temps définir deux notions nécessaires pour poursuivre cette étude :

la connexité : un point A est dit vérifier le critère de connexité s'il existe l'extrémité d'un segment obstacle dans le cercle de centre A et dont le rayon est égal à la largeur du véhicule plus une distance de sécurité. Ceci signifie que le robot n'a pas la place de circuler entre le point A et ce segment.

le critère de traversée : le critère de traversée consiste à regarder si l'origine et le but sont situés de part et d'autre de l'obstacle formé par deux segments. Pour ce faireΓon crée à partir des données initiales deux segments jointifs et on teste l'intersection du segment $(but, origine)$ avec chacun des deux autres (voir figure 2.17).

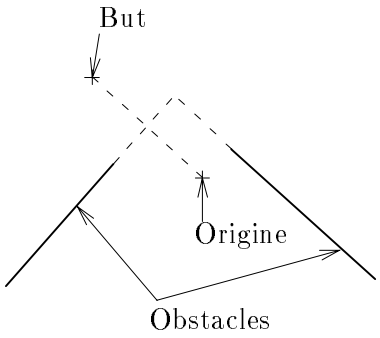


FIG. 2.17 - : *Définition du critère de traversée*

En ajoutant également la notion de *sens* définie précédemmentΓcela fournit trois critères de description de l'environnement géométrique. L'étude des différentes valeurs que peut prendre ce triplet nous a conduit aux trois cas de figures que nous allons maintenant décrire.

Le premier cas correspond à la présence d'un obstacle isolé. L'extrémité du segment présente dans le rectangle de *freepath* ne vérifie pas le critère de connexité. Ce cas correspond au fait que le chemin envisagé passe trop près de l'extrémité (voir figure 2.18). On construit alors un nouveau point d'évitement permettant d'atteindre P (et donc perpendiculairement à la droite $(Origine, P)$).

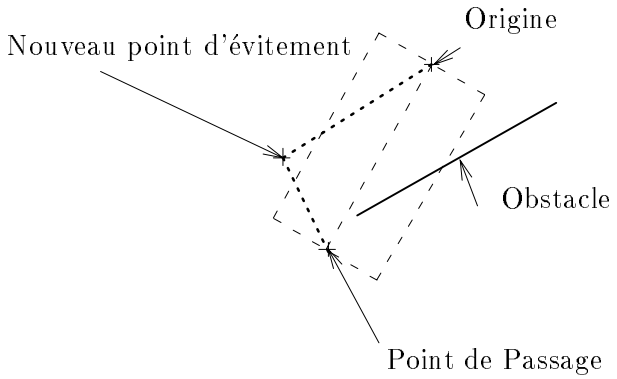


FIG. 2.18 - : *Premier cas : le critère de connexité n'est pas vérifié pour l'extrémité du segment en cause*

Le deuxième cas se produit lorsque l'extrémité de l'obstacle présente dans le rectangle de *freepath* vérifie le critère de sens (droite ou gauche)Γde connexité ainsi que de traversée. Il s'agit alors de franchir la barrière formée par les deux obstacles en considérant le segment connexe comme nouvel obstacle courant pour la détermination du point de passage (voir figure 2.19).

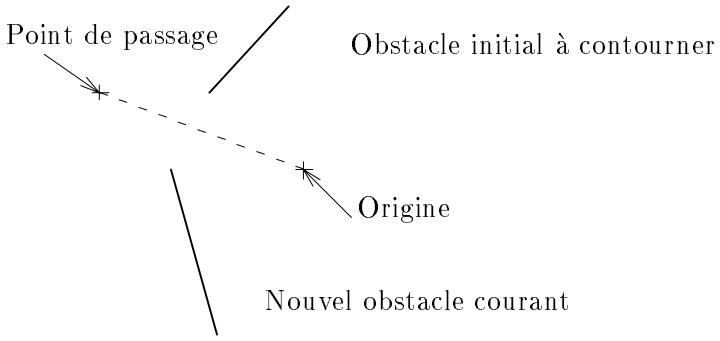


FIG. 2.19 - : *Deuxième cas : il faut contourner la barrière formée par les deux obstacles*

Le troisième cas se produit indépendamment de la valeur prise par le critère de sens lorsque le critère de connexité est vérifié mais pas celui de traversée (voir figure 2.20). Cette situation est identique en cause et en solution au premier cas.

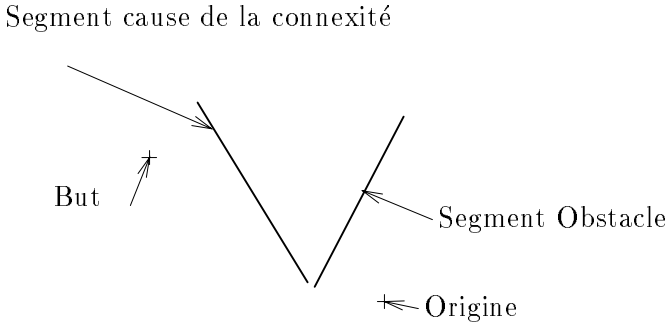


FIG. 2.20 - : *Troisième cas : il faut s'éloigner du segment en cause*

L'algorithme tel qu'il est décrit est susceptible de générer un chemin comportant des arcs inutiles qu'il convient de supprimer (voir figure 2.21). Pour ce faireΓ on parcours la liste des sommets créés dans le sens croissant (de l'origine vers le but) en vérifiant pour un sommet de position n qu'aucun autre de position m tel que $m > (n+1)$ n'est directement atteignable. Dans le cas contraireΓtous les sommets entre $n+1$ et $m-1$ sont supprimés.

Soit n le nombre de segments présents dans le modèle local et m le nombre de segments du chemin construit. La complexité de l'algorithme est :

- de l'ordre de mn^2 pour la construction du cheminΓ
- de l'ordre de m^2n pour la simplification du chemin construit.

La solution proposée n'est pas complète. Elle ne garantit pas de trouver une solution si celle-ci existe. Elle est similaire dans son principe au *graphe de visibilité*

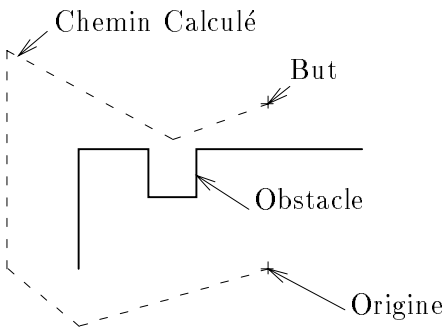


FIG. 2.21 - : *Le chemin créé peut comporter des sommets inutiles*

mais utilise directement l'espace de travail et non l'espace des configurationsΓ coûteux à générer.

La dernière étape consiste à parcourir le chemin calculé. C'est le rôle du module de navigation que nous allons présenter.

2.3.5 La navigation

La fonction principale de ce module est *goto*. Elle admet pour paramètre un point de l'espace d'évolution spécifié par ses coordonnées absolues (x, y). L'orientation finale θ n'est pas imposée. Le but de la fonction n'est pas de permettre au robot de se positionner finement par rapport à un poste fixe en vue d'un accostage par exemple. Il s'agit plutôt d'amener le véhicule dans une zone particulière afin par exemple de réaliser diverses mesures capteurs (détection de fumée etc).

Grâce au module de perceptionΓla fonction *goto* vérifie tout d'abord si le chemin direct est valide ou non. Si ce n'est pas le casΓun chemin d'évitement est calculé sous forme d'une liste l_s de sommets. Ces sommets sont ensuite atteints séquentiellement jusqu'au but.

L'environnement étant dynamique et étant perçu au fur et à mesure du déplacementΓil est nécessaire de vérifier en cours de route la validité de la partie du chemin parcouru jusqu'au prochain sommet. Ceci est réalisé par un appel périodique au module de perception. Si un obstacle est détecté en cours d'exécutionΓ un nouveau chemin est recalculé remplaçant celui présent dans l_s. Si aucun chemin n'est trouvé (porte fermée ou passage trop encombré par exemple)Γle robot envoie alors un message au module de supervision [36] qui replanifie une nouvelle route grâce à son réseau d'emplacements.

2.4 Résultats expérimentaux

Nous allons présenter au cours de cette section quelques exemples d'exécution de notre système.

2.4.1 Le contrôleur de véhicule

La figure 2.22 représente la trace d'exécution d'un ensemble de commandes de type *Move* et *Turn*. Les deux types de mouvements peuvent être combinés de manière à obtenir des trajectoires complexes.

FIG. 2.22 - : *Exemple d'exécution d'une succession d'ordres de translations et de rotations*

2.4.2 La modélisation

Dans l'exemple suivant le robot se déplace le long d'un couloir. Le modèle de l'environnement maintenu est initialement vide et se construit au fur et à mesure du déplacement. Les trois copies d'écran figure 2.23 représentent le modèle obtenu à trois instants successifs. L'épaisseur de chaque segment est associé au nombre de fois que le segment a été observé. A la première observation de l'environnement un premier segment est créé correspondant à l'observation du mur droit. Un second est ensuite détecté correspondant au mur gauche. Ils sont ensuite allongés lors de nouvelles observations au fur et à mesure du déplacement.

2.4.3 La perception et la navigation

La figure 2.24 représente un exemple de chemin calculé par le module de perception dans un environnement simulé. Le robot a pour objectif de pénétrer à l'intérieur du carré.

L'exemple suivant (figure 2.25) illustre l'exécution d'un ordre de navigation sur le robot réel. La figure de gauche représente la perception initiale du véhicule.

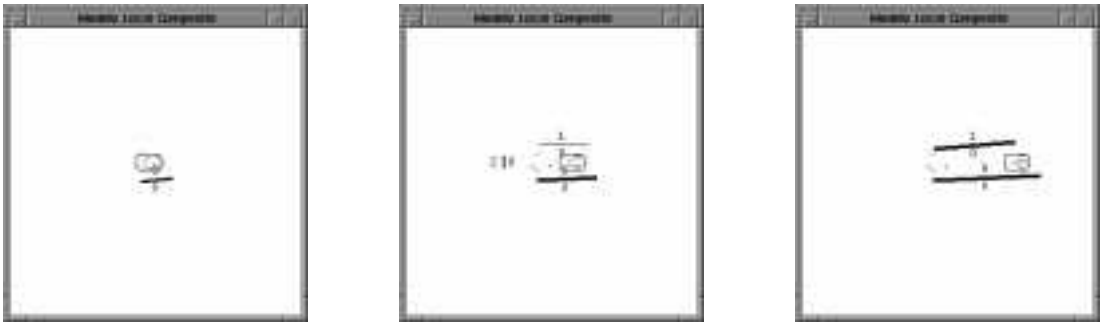


FIG. 2.23 - : *Contenu du modèle local à trois instants différents lors du parcours d'un couloir par le robot.*

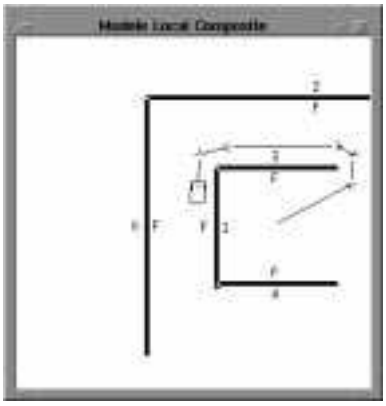


FIG. 2.24 - : *Exemple de recherche d'un chemin à partir d'un modèle de l'environnement*

Les segments 1 et 2 correspondent au coin de la pièce dans laquelle il se trouve. Le segment 3 est un obstacle. Devant rejoindre un point situé de l'autre côté, le module de perception planifie un chemin d'évitement que le module de navigation exécute. Le modèle local se complète au fur et à mesure, pouvant remettre en cause le chemin initial et exiger une replanification.

FIG. 2.25 - : *Exemple d'exécution sur le robot d'un ordre de navigation.*

2.5 Conclusion

L'architecture de perception et d'action que nous avons réalisée dans le cadre du projet Mithra permet la programmation du robot en terme de buts. Les modules mis en place lui permettent de se déplacer vers un objectif tout en évitant les obstacles et en maintenant sa position estimée. Les résultats expérimentaux laissent apparaître néanmoins un certain nombre de problèmes limitant les capacités du système :

- la modélisation de l'environnement en terme de segments de droite est adaptée au problème de relocalisation mais pas de navigation. La plupart des obstacles sont de dimensions trop faibles ou ne présentent pas de surfaces planes suffisantes pour pouvoir être modélisés sous forme de segments de droite et être donc intégrés à la représentation de l'environnement.

- le temps nécessaire entre l'apparition d'un obstacle, son intégration au modèle et sa prise en compte dans le chemin parcouru est trop important. Les réactions du robot sont lentes et inappropriées.

Afin de pouvoir augmenter l'autonomie du robot, nous avons reconsidéré le problème de la navigation. Nous nous sommes plus particulièrement intéressés à des approches plus bas niveau, réduisant le lien entre la perception et l'action. Ces approches, regroupées sous le nom de navigation réactive, font l'objet de la suite de cette étude.

Chapitre 3

La navigation réactive

Nous allons nous intéresser au cours de ce chapitre à la définition de la navigation réactive. Nous resituerons tout d'abord le contrôle vis-à-vis de la planification. Ceci nous amènera à la définition de comportements telle qu'elle nous est fournie en particulier par les éthologistes. Nous détaillerons son utilisation par les roboticiens avant de préciser l'architecture que nous avons retenue et de définir la navigation réactive—cadre de notre étude.

3.1 Motivations

Le chapitre précédent montre que les premières études en robotique se sont appuyées sur une approche du problème fondée sur les acquis de l'intelligence artificielle. Le cycle classique mis en place peut se résumer par les trois mots *percevoir, penser, agir* (voir figure 2.1). Un premier système—généralement de type symbolique—gère les objectifs du robot et détermine un ensemble de buts géométriques devant être atteints (partie *pense* du cycle). Les données provenant des différents capteurs sont filtrées—fusionnées et intégrées dans un modèle de l'environnement (partie *perçoit*). Un chemin est ensuite extrait de ce modèle grâce à un planificateur géométrique (partie *pense* de nouveau—voir [60] par exemple). Ce chemin est finalement traduit en actions de déplacement (partie *agit*). Le robot est contrôlé par une série de planificateurs.

L'état d'un système dynamique peut être décrit par les valeurs prises par un ensemble de variables (appelées variables d'état). Le but de la théorie du contrôle est de concevoir un processus (analogique ou numérique) capable d'amener les valeurs d'une ou plusieurs de ces variables à des valeurs objectifs correspondant au but que l'on souhaite atteindre. Le principe de la planification est le suivant : on réalise un modèle du système que l'on souhaite contrôler. Ce modèle est utilisé comme prédicteur permettant de déterminer l'effet sur les valeurs prises par les variables d'état de n'importe quelle commande envoyée au système. Connaissant l'état initial et l'état final que l'on souhaite atteindre—le planificateur peut grâce

à ce prédicteur générer hors-ligne la succession de commandes devant être exécutées.

Si le modèle utilisé est un modèle parfait décrivant fidèlement le comportement du système, les commandes générées par la planification peuvent être envoyées successivement au système lors de la phase d'exécution et le but est atteint. On parle de commande en boucle ouverte. Si le modèle utilisé comporte des imperfections, il y a dérive. Si on souhaite rester dans une optique de boucle ouverte, la solution consiste à attendre que la totalité des commandes planifiées soient exécutées puis de mesurer l'écart entre l'état obtenu et l'état souhaité. Si celui-ci est trop important, il faut alors replanifier une succession de commandes et ainsi de suite jusqu'à convergence éventuelle vers le but. Dans une application de type robotique, une telle approche est dangereuse car la dérive du système en cours d'exécution peut conduire à une collision avec un obstacle par exemple. Il est alors nécessaire de mettre en place un processus de suivi d'exécution permettant de détecter d'éventuels problèmes et de replanifier si besoin est avant que la totalité des actions précédemment prévues ne soit complètement exécutée.

Les problèmes sont doubles dans le cadre de la planification de chemins en robotique :

1. comme nous l'avons déjà indiqué lors de la section 2.3, les modèles utilisés pour la représentation de l'environnement et acquis automatiquement par le robot sont très souvent incomplets et contiennent de nombreuses erreurs. Cela provient d'une part de la faible qualité des capteurs utilisés, entraînant des erreurs de forme et de position des obstacles modélisés. Cela provient également du coût en temps souvent important pour l'acquisition et le traitement des données, entraînant un délai non négligeable entre l'apparition d'un nouvel obstacle dans le champ visuel du robot et son apparition effective dans le modèle de l'environnement.

2. si quelques planificateurs intègrent maintenant les capacités cinématiques du robot (contraintes de non-holonomie, rayon de courbure borné, etc [85, 167]), ils ne prennent en compte en général que quelques contraintes dynamiques simples telles que des vitesses ou des accélérations bornées mais ne font pas apparaître les aspects liés par exemple à l'inertie du système ou au temps de réponse des asservissements bas niveaux.

Dans le cas du second problème (aspect dynamique du robot), à l'exception de la méthode originale des potentiels, telle qu'elle a été présentée par Khatib [97] dans le cadre d'un bras manipulateur, ou de l'approche à base de fonctions harmoniques proposée par Connolly [43], les planificateurs géométriques ne génèrent pas directement une succession de commandes (non fiables) mais une suite de configurations décrivant un chemin à suivre. Un processus de contrôle en boucle fermée de type "poursuite pure" par exemple [179] se charge alors de maintenir le véhicule le long de ce chemin. Rappelons qu'un processus de contrôle en boucle

fermée est un processus qui n'exécute pas une série de commandes avant de mesurer l'erreur et de replanifier mais qui recalcule en permanence les commandes à envoyer de manière à contraindre le système vers son but. Il est donc plus à même de part sa nature à corriger très rapidement les dérives survenant.

Les algorithmes actuels de modélisation de l'environnement sont la plupart très coûteux en temps de calcul. Le robot de plus a souvent qu'une vue très partielle de son environnement. Ceci implique que si le robot ne remettra pas en cause le plan proposé pour des raisons de dynamique (grâce à la poursuite pure par exemple)Γil devra vraisemblablement le remettre en cause pour des raisons de présence imprévue d'obstaclesΓentraînant une nouvelle modélisation coûteuse de l'environnement et une regénération de chemin. De plusΓle délai entre l'apparition d'un nouvel obstacleΓsa prise en compte dans le modèle et la regénération d'un nouveau plan pouvant être importantΓil existe un risque de collision pour le robot si une réponse immédiate est nécessaire.

Les roboticiens ont cherché à remplacer ou compléter le contrôle à base de plans par des processus de contrôle rapide en boucle fermée de manière :

- à éviter le calcul inutile d'une succession d'actions qui seront très vraisemblablement remises en causes avant d'être exécutéesΓ
- à vaincre l'inertie de ces systèmes à base de planificateursΓdû à la complexité des calculs nécessaires pour maintenir un modèle et à la remise en cause fréquente des plans à cause de l'inadéquation de ces modèles avec la réalité (essentiellement des modèles perceptifs comme nous venons de l'indiquer).

Ces processus ne calculent qu'une commande à la fois et cherchent à la relier le plus directement possible aux données "brutes" provenant de la perception. Le lecteur intéressé peut se reporter à la thèse d'état de Zapata [192] pour une étude approfondie du contrôle d'un robot mobile.

3.2 Approches comportementales

Les approches dites comportementales ont pour but de concevoir et de réaliser un robot "réactif"Γcapable d'apporter une réponse immédiate à toute nouvelle modification de l'environnement le concernant. Ces approches sont initialement fondées sur l'étude de l'interaction de l'animal avec son environnement naturel de manière à essayer de comprendre les mécanismes mis en œuvre et à essayer de les reproduire.

3.2.1 Le comportement animal

L'étude du comportement animal peut être réalisée selon une approche montante ou descendante [8]. L'approche montanteΓpréconisée par les physiologistesΓ

tente d'expliquer le comportement par les interactions existant entre plusieurs sous-composants physiologiques. L'approche descendante (éthologistesΓ psychologistes du comportement) cherche au contraire à comprendre le comportement de l'animal par rapport aux différents stimuli auxquels il est soumisΓsans chercher à analyser les processus physiologiques mis en œuvre.

Les comportements que l'on retrouve chez les animaux présentent un certain nombre de propriétés que nous allons reprendre ici (voir [19Γ8] par exemple) :

- le comportement d'un animal est en constante adaptation pour faire face aux modifications de l'environnement ou à ses propres modifications.
- les comportements peuvent être classés en plusieurs catégories (les termes utilisés sont les termes originaux anglais) :
  - *reflexive* : le comportement réflexe est une réponse rapide stéréotypée à un stimulus. Cette réponse est fonction de l'intensité et de la durée du signal. Le réflexe permet à l'animal de s'adapter aux changements soudains de l'environnement.
  - *taxes* ou *orientation responses* : ce sont des comportements simples imposant à un animal de s'éloigner ou de se rapprocher d'un agent de l'environnement comme la lumière ou un signal chimique.
  - *fixed-action patterns* : ce sont des comportements complexes fournissant un ensemble ordonné d'actions en réponse à un stimulus (stratégie d'évitement d'un prédateur par exemple après sa détection).
- les comportements ne sont pas tous uniquement réflexifs (dépendant de stimuli fournis par l'environnement). Ils peuvent être également motivés par un état interne de l'animal (faimΓpeurΓcuriosité ...). Les comportements motivés sont principalement caractérisés par leur orientation vers un butΓ leur spontanéité (aucun stimulus déclencheur) et leur persistance.
- les comportements futurs peuvent être affectés par les comportements passés (accoutumanceΓsensibilisationΓapprentissage associatif ...).
- les animaux possèdent une hiérarchie comportementale : les comportements les plus complexes sont décomposables en comportements plus simples.

Les aspects de réflexesΓd'adaptation ou de motivation sont des propriétés que l'on souhaiterait retrouver pour un robot mobile. Les approches comportementales ont essayé de s'inspirer plus ou moins fidèlement de ces principes afin de créer des systèmes robotiques. Des comportements de type réflexe assurant la sécurité du robot sont tout d'abord créés. Ils servent de "briques" élémentaires pour la conception de comportements plus complexes. On peut distinguer deux grandes familles d'approches selon le type d'architecture mis en place :

- les approches purement comportementalesΓ

– les approches hybrides

Nous allons revenir sur ces deux catégories d'architectures au travers de quelques systèmes.

3.2.2 Les architectures purement comportementales

Dans les architectures purement comportementalesΓles réactions du robot ne sont déterminées que par la combinaison de comportements. Il n'y a pas de phase de planification ou de modélisation de l'environnement. La première et la plus célèbre des architectures reprenant ce principe est celle proposée par Brooks.

l'architecture subsomption

L'architecture subsomption proposée par Brooks [29Γ28] est une hiérarchie de compétences où chaque niveau supérieur peut subsumer les niveaux inférieurs grâce à des mécanismes d'inhibition et de suppression.

Un niveau de compétence est une spécification informelle d'une classe voulue de comportements pour un robot dans tous les environnements qu'il sera susceptible de rencontrer. Les huit niveaux proposés par Brooks sont regroupés dans leur ordre hiérarchique figure 3.1.

Les niveaux 0Γ1 et 2 correspondent à des niveaux de haute priorité regroupant des actions de type réflexe essentielles à la survie du robot. Ce sont l'évitement d'obstacles mobilesΓl'évitement d'objets se rapprochant dangereusement ainsi que la poursuite d'une cible. Les couches supérieures contiennent la partie décisionnelle du système de navigation. Elles sont de faible priorité. Elles regroupent la reconnaissance des formesΓdes objetsΓdes lieux mais aussi la cartographie et le choix des buts. Elles ont pour but de permettre au robot de répondre à un critère prédéfini et maîtrisable.

Chaque niveau de compétence correspond à une couche de contrôle. Cette couche est composée d'un ensemble de modules fonctionnant de manière asynchrone et pouvant communiquer à l'aide de messages. En particulierΓil n'existe pas de mémoire globale partagée ni de système de contrôle centralisé. Chaque module est implémenté sous forme d'une *Machine d'Etat Fini Augmentée* (machine d'état finie conventionnelle connectée à un ensemble de registres et d'horloges [30]Γ voir figure 3.2). Ces registres permettent la réception ou l'envoi de messages déclenchant alors un changement d'état. L'horloge permet de gérer des mécanismes de type "timeout". La machine d'état fini possède de plus une entrée par défaut contenant un message utilisé si aucune donnée ne provient des autres modules.

La résolution de conflits (envoi par exemple de deux commandes contradictoires aux moteurs par deux modules différents) est réglée par deux mécanismes distincts (voir figure 3.2) :

1. un mécanisme d'inhibition. L'arrivée d'un message d'inhibition a pour effet

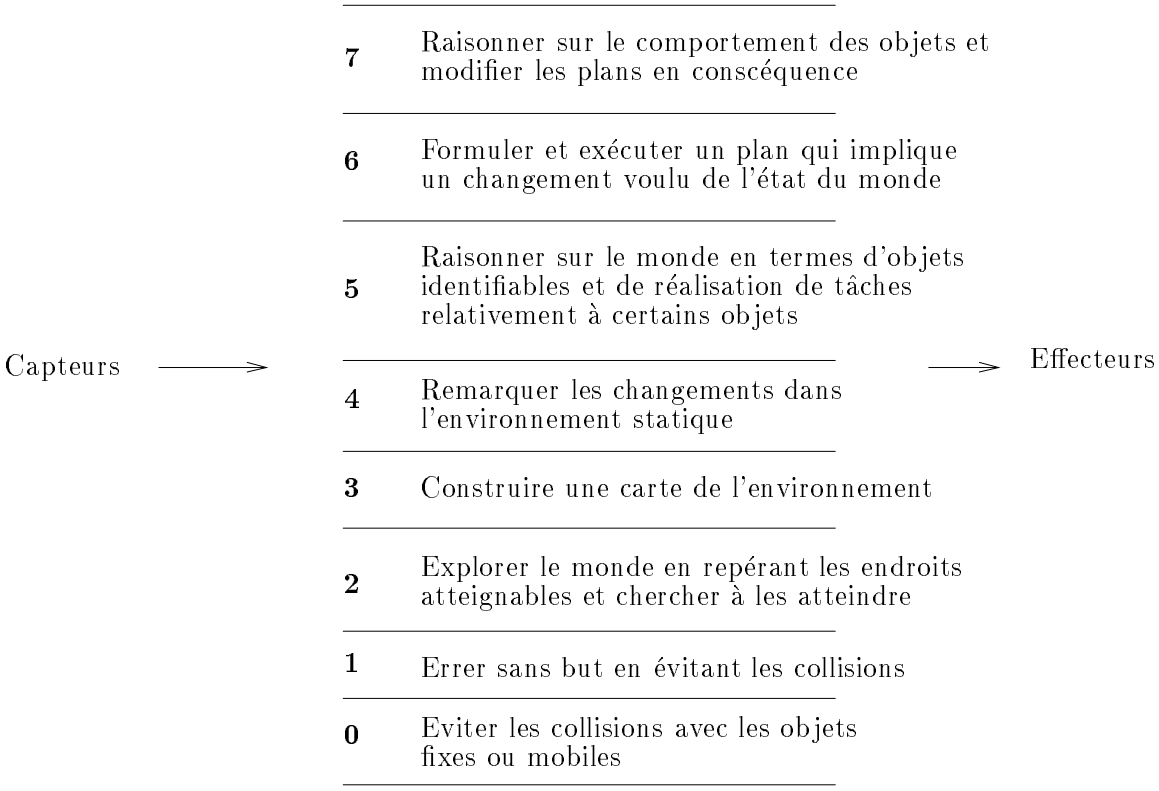


FIG. 3.1 - : *Décomposition d'un système de contrôle en comportements*

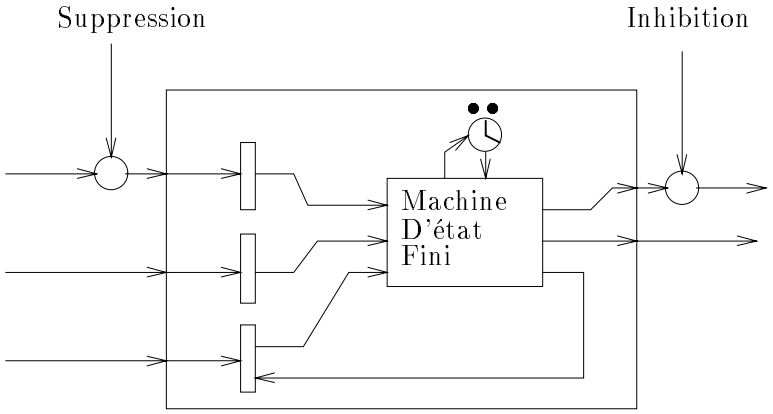


FIG. 3.2 - : *Description d'une machine d'état fini augmentée*

de bloquer la sortie correspondante du module pendant une courte période.

2. un mécanisme de suppression. Lorsqu'un message de suppression se présente sur l'aiguillage du mécanismeΓles messages provenant de la source normale sont remplacés pour un court laps de temps par le message de suppression.

Cette architecture a connu très rapidement un grand succès de part sa facilité de mise en œuvre (tout au moins pour les trois premiers niveaux) et par la robustesse des comportements obtenus. Elle a été à l'origine de nombreux systèmes aussi bien en robotique mobile qu'en robotique de manipulation.

Autres approches

Il existe d'autres approches purement comportementales. Les lecteurs intéressés peuvent se reporter par exemple aux travaux de Anderson et Donath [7Γ6Γ9Γ8] dont l'architecture se décompose en comportements réflexifsΓréalisant un lien direct entre la perception et l'actionΓet en comportements réactifsΓchoisissant à tout instant les comportements réflexifs devant être activés grâce à d'autres facteurs tels que la motivation. BeerΓChiel et Sterling [19] se sont intéressés à la réalisation de systèmes de type "insecte artificiel" en reproduisant un système nerveux formé de neurones interconnectés ayant chacun un rôle d'action ou de détection d'un phénomène extérieur ou d'un état intérieur permettant de motiver le système (appétit par exemple si le niveau d'énergie est faible). Citons enfin les travaux de P. Maes [115Γ116Γ118] dans le domaine de la sélection dynamique d'actions. Le problème qu'elle cherche à résoudre est la détermination d'une succession d'actions permettant de conduire le robot vers le but en exploitant les opportunités et en s'adaptant aux changements imprévisibles de situations. Il s'agit en fait d'une tâche de planification dynamique dont la résolution ne fait pas appel à une structure classique centralisée basée sur des mécanismes de chaînage arrière par exemple. Elle fait au contraire appel à un réseau de modules de compétences (comportements) agencés en fonction des dépendances respectives les uns des autres (saisir un objet impose que la main soit vide). Chaque module possède un niveau d'énergie. Le module dont le niveau est le plus élevé devient actif. L'algorithme proposé par Maes consiste à répartir l'énergie disponible à travers le réseauΓcette répartition étant guidée par l'état de l'environnement et les buts à atteindre.

Comme nous l'avons indiqué auparavantΓtoutes ces approches ont obtenu un très grand succès de par la robustesse des comportements obtenus face aux variations de l'environnement. Il semble néanmoins que tout comme les systèmes purement délibératifs (basés sur des planificateurs et des systèmes complexes de contrôle)Γles systèmes purement réactifs ne soient pas en mesure de fournir un comportement global pour un robot mobile satisfaisant nos attentes (voir Ferguson [59] par exemple). En particulierΓcomme le souligne P. Maes [117]Γla spécification d'un agent ne suffit pas seule à expliquer les fonctionnalités affichées

lors de l'interaction avec l'environnement. L'environnement n'est pas simplement pris en compte mais ses caractéristiques sont exploitées de manière à servir l'objectif. En conséquence il n'est pas possible d'indiquer simplement le but à atteindre au robot. Il faut plutôt trouver une boucle d'interaction entre le système et l'environnement convergeant vers les objectifs (ses travaux [115 116 118] sur la dynamique de sélection d'une action sont une tentative de réponse à ces critiques). Il faut noter que la possibilité de fournir un but précis au robot a été explicitement prévue par Brooks dans son architecture au niveau des couches supérieures. On constate néanmoins qu'il semble impossible sur le plan pratique de concevoir et de réaliser un système robotique basé sur la subsomption allant au-delà de la troisième couche.

Un deuxième courant de recherche s'est donc développé cherchant à combiner d'une part la robustesse des comportements de bas niveau et d'autre part les capacités bien adaptées à notre problème qu'offrent les planificateurs. Ces recherches ont débouché sur les architectures hybrides.

3.2.3 Les architectures hybrides

Ces systèmes se caractérisent par l'utilisation de phases de planification et de modélisation de l'environnement permettant de choisir les comportements devant coopérer. Nous allons en particulier décrire les systèmes proposés par Arkin et Payton.

Le système AuRA

Arkin [11 12 13] propose de décrire l'interaction entre la perception et l'action sous forme de *schémas*. Le concept de schéma provient de la psychologie et de la neurobiologie : "un schéma est une codification mentale d'une expérience qui inclut une façon organisée de percevoir et de répondre à une situation complexe ou à un ensemble de stimuli (*Webster's Ninth New Collegiate Dictionary, Merraim-Webster 1984*)". On distingue deux types différents :

1. les schémas moteurs
2. les schémas perceptifs.

Les *schémas moteurs* sont des spécifications de comportements génériques. Leurs instanciations permettent de déterminer les actions envoyées au robot. La sortie de chaque schéma se compose d'un vecteur représentant la direction et la vitesse du prochain déplacement devant être effectué par le robot. La sortie finale est obtenue par sommation vectorielle de toutes les propositions.

Les *schémas perceptifs* ont pour rôle de traiter les données provenant des différents capteurs afin de ne fournir aux schémas moteurs que les informations pertinentes à la réalisation de leur tâche.

Ces schémas de perception et d'action représentent les éléments de base permettant la construction du système. Il s'agit maintenant d'être en mesure de déterminer en fonction du but et de l'évolution de l'environnement lesquels doivent être activés. Ceci est réalisé grâce à l'architecture AuRA [1] représentée figure 3.3.

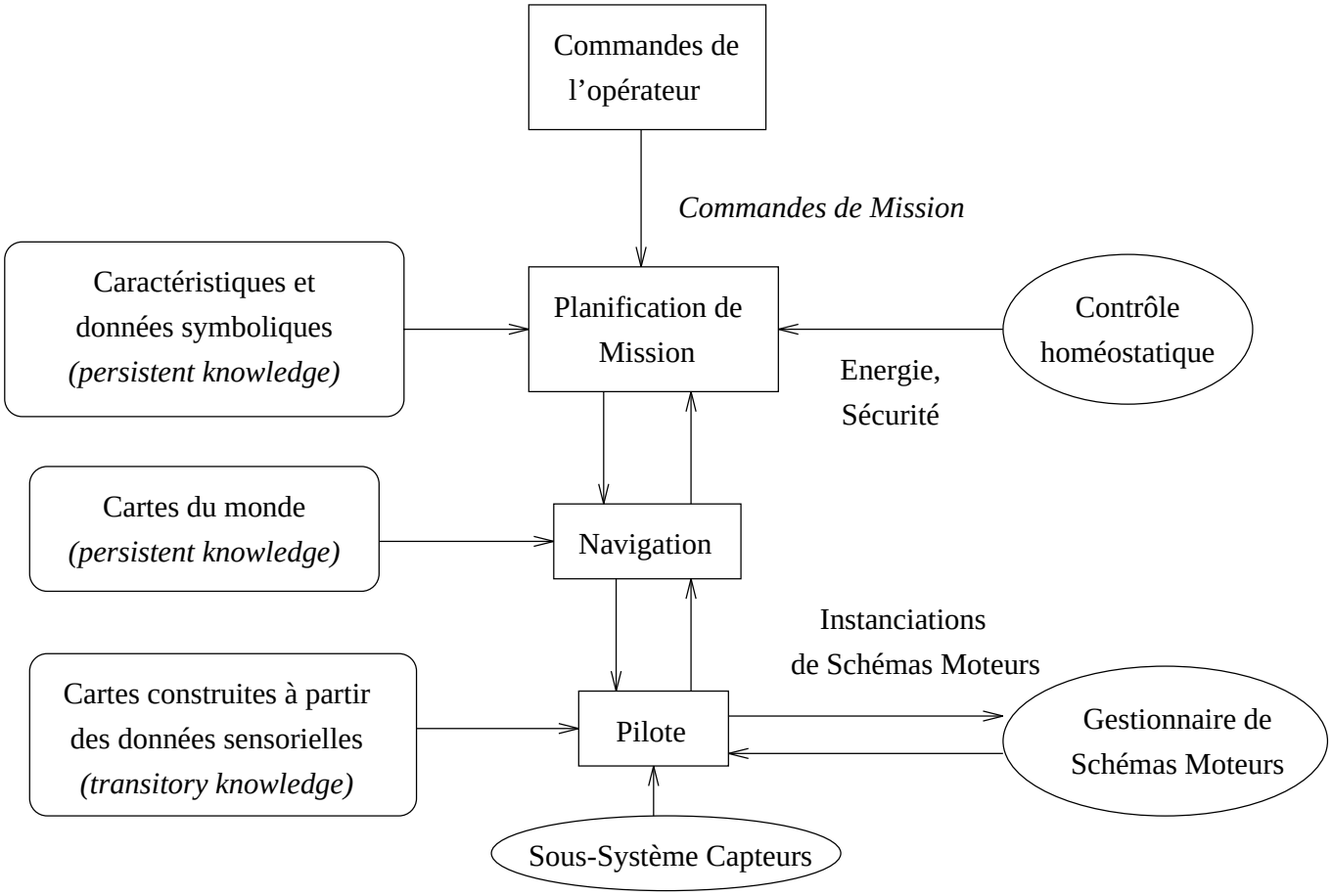


FIG. 3.3 - : *Description de l'architecture AuRA*

Le module de planification de mission a pour tâche d'interpréter les commandes de haut niveau fournies par l'opérateur et de les traduire en un ensemble de buts à rejoindre. Ces buts ainsi qu'une carte des lieux sont ensuite utilisés par le module de pilotage permettant de générer un chemin repris par le module pilote afin d'instancier les différents schémas perceptifs et moteurs chargés de le suivre et de contrôler le bon déroulement (détection d'obstacles imprévus par exemple).

1. Autonomous Robot Architecture

La librairie de stratégies réflexives

L'architecture proposée par Payton [130] a été conçue afin de concilier :

- les exigences d'une représentation de l'environnement et d'une prise de décision de haut niveauΓcoûteuse en terme de temps de calcul mais nécessaire pour la réalisation d'activités complexes pour un robotΓ
- la nécessité de coupler rapidement au niveau du contrôle du véhicule l'action à la perception afin de pouvoir réagir face à une situation imprévue.

L'architecture est composée de quatre niveaux opérant en parallèle (voir figure 3.4). Plus un niveau est élevé et plus les données sensorielles utilisées sont abstraites.

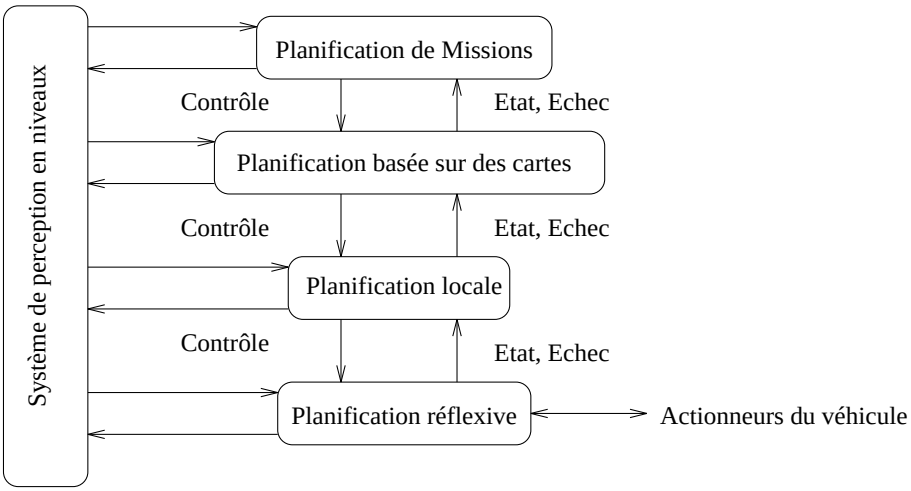


FIG. 3.4 - : *Architecture proposée par Payton*

Nous allons revenir sur chacun de ces niveaux :

- la planification de mission a pour objectif de transformer une série de buts abstraits en buts géographiques. Le temps d'exécution peut être de plusieurs minutesΓ
- le planificateur basé sur cartes permet d'établir les routes vers les différents objectifs. Le temps d'exécution peut être de plusieurs minutesΓ
- le planificateur local a pour tâche de sélectionner les différentes actions devant être mises en jeu afin de suivre les routes. Il doit également veiller au bon déroulement des opérations. Le temps de cycle doit être de l'ordre de quelques secondesΓ
- le planificateur réactif a pour tâche de réaliser le contrôle effectif du véhicule. Son temps de cycle doit être inférieur à la seconde.

Nous allons revenir sur le planificateur réflexif. Ce niveau contient une importante collection de modules experts appelés *comportements réflexifs*. Ces modules experts ont pour rôle de relier la perception à l'action au travers de *capteurs virtuels*. Un capteur virtuel est un module de traitement permettant d'extraire une caractéristique particulière des données sensorielles (cf schémas perceptifs définis ultérieurement par Arkin). Un exemple est donné figure 3.5. L'exécution d'une

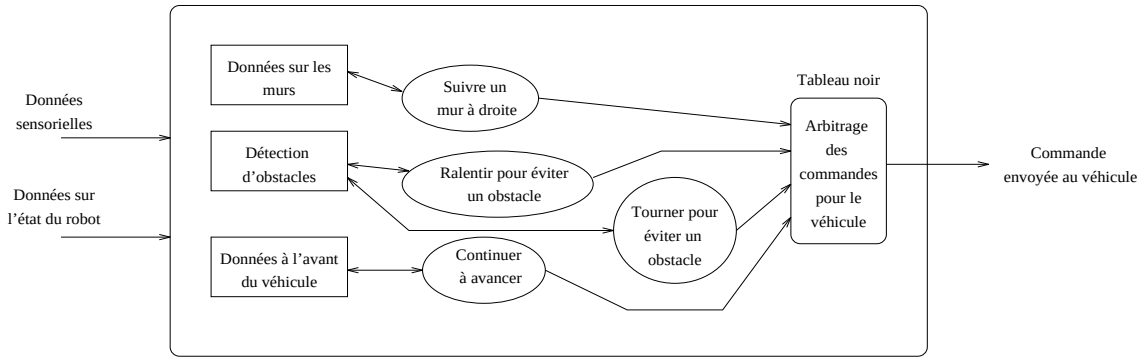


FIG. 3.5 - : *Exemples de modules experts*

tâche de navigation nécessite l'utilisation de plusieurs comportements réflexifs (par exemple l'explorer peut être réalisé en suivant le mur situé sur la droite tout en évitant les obstacles frontaux). A chaque tâche est associé un ensemble de comportements réflexifs appelé *activité*. Le choix de l'activité devant être exécutée est déterminé par le planificateur local. Au sein d'une activité les comportements sont exécutés en parallèle de manière asynchrone. Le choix de la commande finale parmi celles proposées est réalisé grâce à un mécanisme de priorité implémenté sur un tableau noir.

Planification adaptée capteurs

Le système proposé par Collin et al [42] est composé d'un planificateur de haut niveau chargé de déterminer un chemin pour le robot en termes de configurations à atteindre. Ce chemin est ensuite exécuté par un module de bas niveau. L'originalité de cette approche provient de la prise en compte dès la planification des capacités sensorielles du robot.

A chaque configuration libre de l'espace d'évolution est associé un nombre réel positif (appelé CUP [2]) indiquant le degré de précision que pourra atteindre le robot en se localisant à cet endroit grâce à ses capteurs. Collin détermine également l'ensemble minimum de murs permettant cette localisation. L'algorithme de planification retenu est un algorithme de type $*$ dont la fonction coût est une combinaison pondérée de la distance parcourue, de la facilité de localisation et du

2. Configuration Uncertainty Potential

nombre de murs différents nécessaires pour cette localisation entre deux configurations successives. Ce dernier critère permet d'assurer une continuité des cartes utilisées pendant le déplacement.

Une fois le chemin calculéΓles différents ensembles de murs associés à chacune des configurations déterminées sont regroupés en un nombre réduit de cartes locales par un algorithme de classification. La commande du véhicule est ensuite réalisée par un algorithme d'asservissement bas niveau appelé fonction de tâche associé à chacune des cartes précédentes et chargé d'asservir le robot sur la portion de chemin correspondant. Le changement de carte et donc de fonction de tâche est réalisé lorsque la carte courante ne fournit plus la meilleure localisation.

Les grandes approches comportementales que nous venons de décrire présentent un ensemble de caractéristiques les rendant plus ou moins bien adaptées aux contraintes imposées par le type d'application que nous souhaitons pouvoir réaliser avec un robot mobile.

3.3 Cadre de notre étude

Nous allons tout d'abord préciser l'architecture retenue dans le cadre de notre travail. Nous définirons ensuite ce que nous appellerons un comportement de navigation.

3.3.1 Architecture retenue

L'objectif de notre étude en robotique mobile est de concevoir et de réaliser un système autonome capable d'accomplir un ensemble de tâches pour lesquelles il a été programmé. Le domaine auquel nous nous sommes particulièrement intéressés dans le cadre du robot Mithra est celui de l'agent de surveillance dans un environnement industriel. Dans le cadre de sa missionΓun agent de ce type peut être confronté à des ordres comme :
patrouiller dans le couloir A jusqu'à $11h$*, puis se rendre dans la pièce* $1c$ *afin de détecter d'éventuels intrus*
Cet exemple permet d'illustrer le caractère explicite des activités devant être supportées par le robot.

Nous avons mis en évidence au cours de la section précédente les deux grandes familles d'architectures d'approches basées sur les comportements :

- les architectures purement comportementalesΓ
- les architectures hybrides.

Comme nous l'avons déjà préciséΓles architectures de type purement comportementales ne semblent actuellement pas être en mesure de répondre aux exigences imposées par le type de tâches envisagées pour le robot. Nous nous sommes donc tournés vers les architectures hybrides mieux adaptées à nos contraintes.

Les architectures proposées par Arkin et Payton sont très similaires dans leurs principes et dans leurs buts à celle proposée par Crowley [49] et que nous avons reprise dans le cadre du projet Mithra (voir chapitre 2). Nous allons donc repartir de cette architecture et séparer le niveau navigation et perception en deux niveaux (voir figure 3.6) :

- le niveau le plus élevé reste attaché à la planification de chemins côté perception et à la gestion des sous-buts côté mobilitéΓ

- le niveau inférieur implémente côté mobilité un comportement réactif de navigation vers un but en évitant les obstacles à partir des données fournies par la perception (capteurs virtuels). Ces notions seront reprécisées au cours de la prochaine sous-section.

Il est important de remarquer que le système de navigation réactive est situé au-dessus du contrôleur de véhicule et de la modélisation. Cela signifie que pour rejoindre son butΓce système pourra utiliser la position estimée du robot comme une donnée fiable. Il n'aura pas à prendre en compte les dérives provenant inévitablement d'un système odométriqueΓcelles-ci étant corrigées.

Si l'on reconsidère les explications fournies lors de la section 3.1Γl'objectif de notre travail est d'augmenter la sécurité et l'efficacité du robot en reportant la génération de commande de la planification vers un processus de contrôle plus adapté en boucle fermée. Nous utiliserons ici les termes employés par Payton [131] ou Agre [2] et décrivant parfaitement le problème : il s'agit de remplacer *le plan comme programme* par *le plan comme ressource pour l'action.* Nous tenons à faire remarquer à ce stade que la transition du premier type de plan vers le second nécessite deux étapes :

1. la mise en place du processus de contrôle en boucle ferméeΓthème de notre étudeΓ

2. la gestion de la coopération entre ce processus bas niveau et la planification.

Le second point dépasse le cadre de notre travail et constitue un problème très important nécessitant une étude approfondie. Nous avons choisi d'utiliser la solution employée par Krogh [106] consistant à fournir au processus de contrôle un ensemble de sous-buts générés par le planificateur géométrique. Cette approche présente des limitations [131]. Un dialogue de plus "haut niveau" semble être nécessaire entre les deux modules (utilisation de *plans comme communication* [131Γ2]).

Nous allons maintenant définir le comportement de navigation que nous allons étudier ainsi que les hypothèses que nous avons retenues.

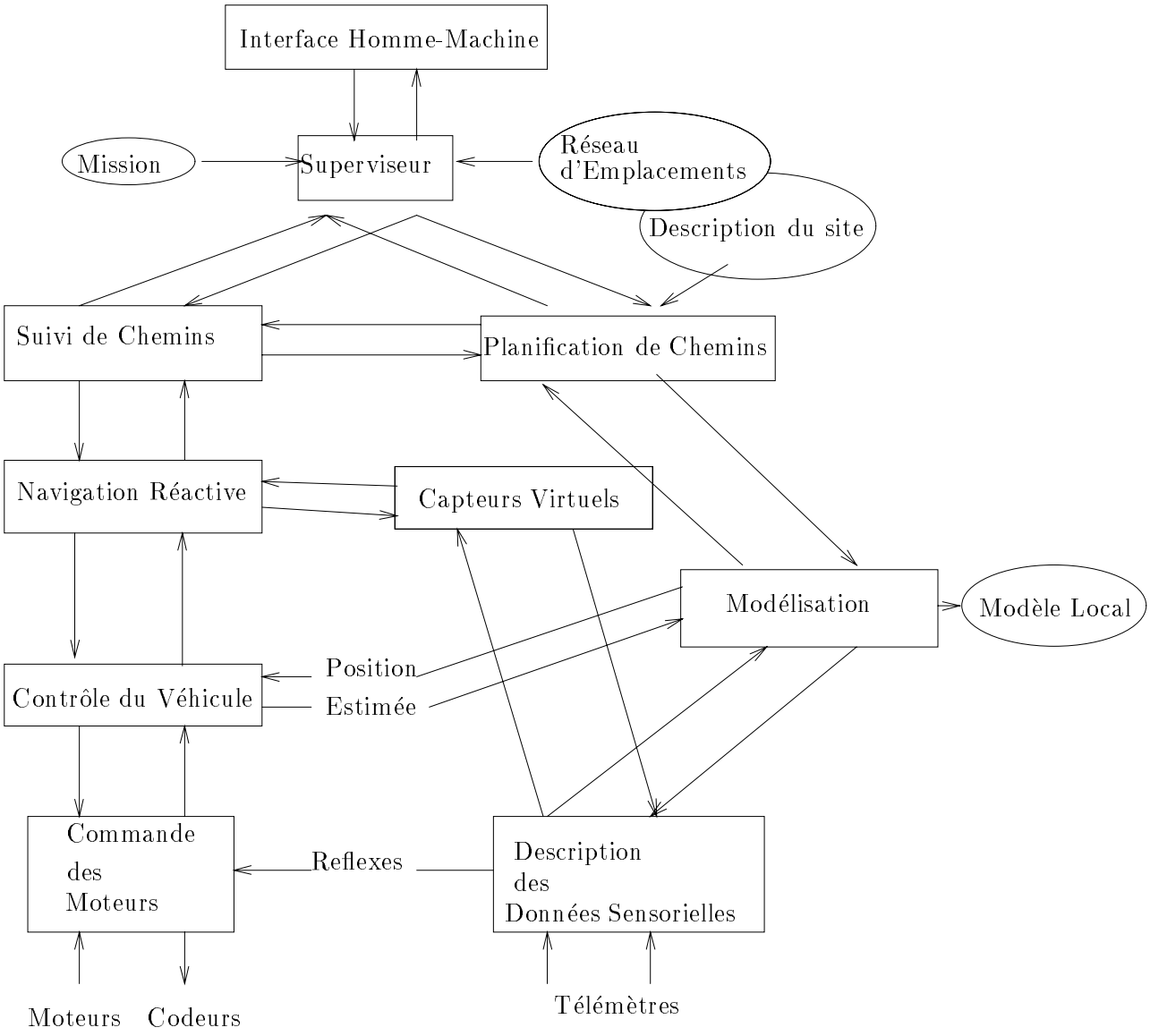


FIG. 3.6 - : *Architecture hybride dérivée de l'architecture proposée par Crowley*

3.3.2 Définition d'un comportement de navigation

Le comportement auquel nous allons nous intéresser au cours de cette étude est celui de l'évitement d'obstacles en essayant de rejoindre un but (comportement motivé). Le robot dispose pour cela d'un ensemble de capteurs extéroceptifs (gestion des obstacles) et proprioceptifs (gestion du but). Il dispose également d'effecteurs permettant son déplacement.

Nous appellerons classiquement *Espace perceptif* l'espace de dimension n dont chaque axe est associé à un capteur et représente les valeurs possibles. Ce capteur peut être un capteur physique (télémètre ultrasons par exemple) ou ce que Payton défini comme un capteur virtuel [130]. Un capteur virtuel est un module de traitement acceptant comme entrée les valeurs fournies par un ou plusieurs capteurs physiques et fournissant une valeur en sortie. Le rôle d'un capteur virtuel peut être multiple :

- filtrage simple des mesures fournies par un seul capteurΓ
- fusion multi-sensorielleΓ
- construction d'un modèle et extraction d'indices particuliers (détection de portesΓde couloirs etc). La construction d'un modèle permet de réaliser une intégration dans le temps des différentes données. Il faut néanmoins noter que le but de cette étude est d'essayer d'éviter d'avoir recours à des processus de modélisation trop complexes et peu fiables.

Nous appellerons de même *Espace d'action* l'espace de dimension m dans lequel chaque axe est associé à un effecteur. De même que pour l'espace perceptifΓ ces effecteurs peuvent être directement des effecteurs physiques (commande des moteurs de chacune des roues) ou peuvent correspondre à des mécanismes plus complexes. Dans le cadre de notre problèmeΓnous avons choisi de commander le véhicule au travers de son contrôleur (voir chapitre précédent) en termes de vitesses ou de déplacements linéaires et angulaires.

Nous appellerons *comportement de navigation* toute fonction f réalisant une transformation de l'espace perceptif vers l'espace d'action (voir figure 3.7). Les sous-buts devant être rejoints par le véhicule proviennent d'un planificateur et ne doivent donc pas être très éloignés les uns des autres. La tâche du comportement de navigation est l'évitement d'obstacles imprévus mais n'est certainement pas de conduire seul le véhicule à l'autre bout d'un bâtiment à travers plusieurs pièces et couloirs. Nous avons choisi pour cette raison de représenter l'association perception action par l'intermédiaire d'une fonction et non d'un automate. La commande fournie dépend uniquement de la perception réalisée à l'instant courant et pas d'un historique. Il faut néanmoins remarquer que si aucune mémorisation n'est possible au niveau de la transformation des espacesΓelle est néanmoins possible au niveau des espaces eux-mêmes (en maintenant un modèle de l'environnement par exemple comme nous l'avons signalé précédemment).

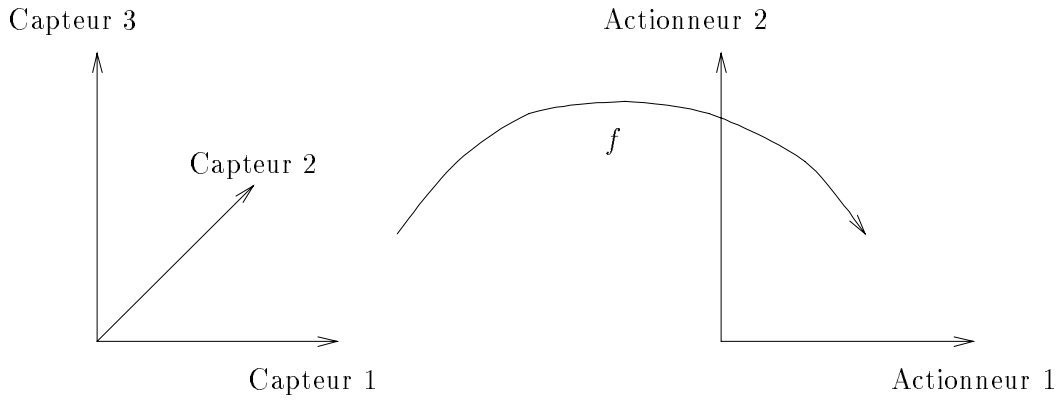


FIG. 3.7 - : *Définition d'un comportement f*

Soit M une fonction inconnue de transformation perception $\mapsto$ action permettant de guider le robot vers son but au milieu des obstacles et donc solution possible de notre problème. Lorsque le concepteur d'un système implémente un comportement de navigation il essaye de créer une fonction f approchant la fonction M qu'il ne connaît pas. Cette approximation doit tout d'abord être correcte localement. Cela signifie que le robot doit pour une situation donnée avoir une réponse correcte compte tenu des capacités du véhicule et des objectifs de navigation. Face à un mur par exemple le robot doit ralentir et changer de direction. La fonction f ensuite doit être définie sur la totalité de l'espace perceptif. Le robot doit être en mesure de faire face de manière appropriée à l'ensemble des situations qu'il sera en mesure de détecter. La dimension de l'espace d'entrée étant généralement importante le nombre de situations différentes est très élevé rendant le problème difficile.

Nous allons au cours des prochains chapitres nous intéresser à plusieurs grandes approches permettant de construire cette transformation f en commençant par les approches basées sur les potentiels.

Chapitre 4

Les champs de potentiels

Les premières grandes approches de la navigation réactive que nous allons considérer au cours de ce chapitre sont classiquement regroupées sous l'appellation "approches à base de potentiels". Le principe général est de réaliser l'association *action* $\mapsto$ *perception* à l'aide d'une fonction générant des commandes cherchant à minimiser un critère donné (la fonction potentiel). Nous nous intéresserons dans un premier temps à l'approche originale proposée par Kathib dans le cadre du contrôle réactif d'un bras manipulateur. Elle a donné lieu par la suite à d'autres approches ne visant plus cette fois à générer des commandes mais un chemin au milieu des obstacles. Bien que ces méthodes soient plus reliées à la planification qu'au contrôle d'exécution nous les décrirons brièvement de part l'importance qu'elles ont eue et qu'elles ont encore dans le domaine de la robotique. Les approches précédentes ont pour hypothèse forte la connaissance d'un modèle de l'environnement et des différents obstacles présents. Nous relâcherons cette hypothèse pour nous intéresser à des méthodes souvent qualifiées de potentiels mais dont la dénomination plus exacte est *méthodes à base de champs de forces*. Nous décrirons finalement l'approche que nous avons proposée.

4.1 Génération de commandes par potentiels

Le principe général de la commande par potentiels a été formulé à l'origine par Khatib (voir [97] par exemple) pour le contrôle d'un bras manipulateur. Plusieurs méthodes ont ensuite vu le jour afin d'essayer d'apporter une solution aux différents problèmes apparus.

4.1.1 Evitement en ligne d'obstacles imprévus

Dans le cadre de son travail de thèse [96] Khatib s'est intéressé au problème du contrôle d'un bras manipulateur dans un univers encombré.

Le contrôle du robot est effectué par une force F appliquée sur l'outil terminal

qu'il s'agit de déterminer en fonction des contraintes mécaniques du système et de l'environnement. Remarque : le lien entre la force F et les différents couples $\Gamma(q)$ moteurs à appliquer sur chacune des articulations de manière à produire F est fourni grâce au Jacobien du système :

$$\Gamma = J^t(q)F$$

Le problème est formulé dans l'espace opérationnel. Cet espace correspond à l'ensemble x des paramètres indépendants permettant de décrire la position et l'orientation de l'outil terminal du robot. L'énergie totale L du système mécanique est composée de l'énergie cinétique et de l'énergie potentielle provenant de la gravitation. Le formalisme de Lagrange permet au travers de cette énergie de mettre en relation la force F avec le déplacement de l'outil :

$$\Lambda(x)\ddot{x} + \mu(x,\dot{x}) + p(x) = F \tag{4.1}$$

où Λ est la matrice d'inertie$\Gamma\mu(x,\dot{x})$ correspond à la force de Coriolis et $p(x)$ à la force de gravité.

La relation 4.1 permet de calculer la commande F assurant au robot une trajectoire particulière. Aucune référence n'est faîte aux obstacles ou à la position du but. Le principe de l'évitement en ligne d'obstacles est le suivant : une particule soumise à un champ attractif de la part d'un point G et à un ensemble de forces répulsives générées par des régions O de l'espace se déplacera vers G sans pénétrer dans O. Il s'agit donc de calculer la force F à appliquer à l'outil terminal de telle sorte qu'il se comporte de manière similaire à cette particule (G correspond au but et O aux différents obstacles). Cette force F est obtenue grâce à l'équation 4.1 en intégrant dans le calcul de l'énergie celle résultant des nouvelles forces virtuelles. L'énergie potentielle ne dépend plus que de la gravité mais également du potentiel U_{att} de la force attractive et de la somme des potentiels U^i_{rep} des forces répulsives associées à chaque obstacle i. EnfinΓde manière à stabiliser son système autour du butΓKhatib propose également d'ajouter une force F_v de dissipation (équivalente à un frottement) fonction de la vitesse.

$$U_{att} = \tfrac{1}{2}k_p(x - x_d)^2 \qquad x_d \text{ étant la position du but}$$

$$U^i_{rep} = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{f_i(x)} - \frac{1}{f_i(x_0)}\right)^2 & \text{si } f(x) \leq f(x_0)\Gamma\text{l'obstacle } i \text{ étant décrit par } f(x) = 0 \\ 0 & \text{sinon} \end{cases}$$

$$F_v = -k_v\dot{x}$$

Le problème principal d'une telle approche réside dans l'existence de minima locaux. La résolution de ce problème peut être abordée de deux manières différentes :

1. en construisant une fonction de potentiel dont le seul minimum soit le but à atteindre.

2. en utilisant une fonction potentiel possédant des minima locaux mais en mettant en place un ensemble de mécanismes permettant au robot de les éviter ou de repartir s'il en est prisonnier.

4.1.2 Les fonctions de navigation

Une fonction de potentiel possédant comme seul minimum le but à atteindre est appelée une *fonction de navigation globale*. Koditschek [98] a montré qu'une telle fonction n'existe pas en général. En particulierΓsi l'espace des configurations est de dimension 2 et qu'il existe q obstacles disjoints homéomorphes au disque unitéΓune fonction de potentiel U possède au moins q points singuliers en forme de cols (voir figure 4.1). La présence de tels points n'est en revanche pas un

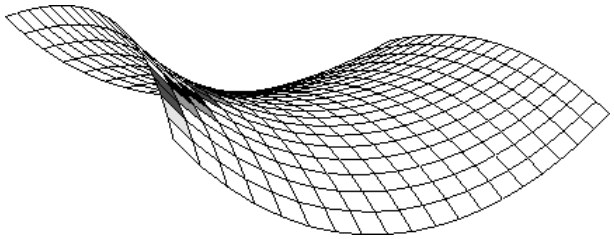


FIG. 4.1 - : *Point singulier en forme de col. Le gradient de la fonction est nul en ce point*

problème car ils représentent un point d'équilibre instable. Une petite perturbation suffit à faire basculer la particule d'un côté ou de l'autre. Une telle fonction de potentielΓque l'on peut expliciter dans ce monde sphérique particulierΓest appelée *fonction de navigation*. Le problème maintenant est de pouvoir calculer cette fonction dans un environnement quelconque. Koditschek a montré que s'il existe un difféomorphisme permettant de transformer le monde sphérique en un environnement AΓce difféomorphisme peut également transformer la fonction de navigation du monde sphérique en une fonction de navigation de A. Rimon et Koditschek ont explicité un tel difféomorphisme dans le cas d'obstacles dont la forme est un convexe étoilé [99]. L'utilisation d'une telle approche dans un cas général semble néanmoins délicate.

4.1.3 Les potentiels généralisés

L'approche proposée par Krogh et Thorpe [106] ne nécessite pas l'utilisation d'une fonction de navigationΓdifficile à fournir dans un cas général comme nous l'avons vu précédemment. Elle se base plutôt sur la constatation qu'un environnement simple est moins susceptible de produire des minima locaux qu'un environnement complexe. Le principe consiste à faciliter le travail du système réactif en lui décomposant le chemin total en un ensemble de sous-buts proches

à rejoindre. Ces sous-buts sont calculés grâce à un planificateur global [170] à partir d'un modèle de l'environnement. Ce modèle peut être incompletΓle rôle du système réactif étant bien entendu de gérer ces imperfections.

L'originalité de l'approche de Krogh réside dans la fonction de potentiel choisieΓappelée *potentiel généralisé*. Cette fonctionΓafin de prendre en compte la dynamique du systèmeΓdépend non seulement de la configuration du robot mais également de sa vitesse:

$$\begin{cases} P(q,v) = P_g(q,v) + P_o(q,v) \\ u = -\alpha \|\nabla_v P\|^{-1} \nabla_v P \end{cases} \tag{4.2}$$

u représente l'accélération commandée au robot et α l'accélération maximum pouvant être appliquée. $P_g(q,v)$ est le potentiel associé au but. Il correspond au temps minimum que le robot mettrait pour atteindre le but à partir de la configuration et de la vitesse courante si aucun obstacle n'était présent et si l'accélération utilisée était l'accélération limite. $P_o(q,v)$ est le potentiel associé aux obstacles. Seuls ceux situés dans la direction prise par le robot sont utilisés pour le calcul. Cela signifie que le robot ne sera pas perturbé par un obstacle proche parallèle à lui. $P_o(q,v)$ est défini comme l'inverse du temps de réserve avant collision $t_M - t_m$ où t_m est le temps minimum dont le robot à besoin pour s'arrêter en utilisant la décélération maximum et t_M est le temps maximum dont il dispose en commençant à freiner maintenant de manière à s'arrêter juste avant l'obstacle. Plus la différence entre ces deux temps est faibleΓplus le freinage devient urgent et plus le potentiel devient élevé.

4.1.4 La méthode des schémas

La méthode des schémas a été proposée par Arkin [11Γ13] afin de permettre à un robot mobile d'effectuer des tâches complexes dans un environnement complexe. Elle trouve son origine en psychologie. Le principe est de créer un ensemble de "comportements" qui sont autant de manières très spécialisées de percevoir une situation particulière et de réagir en conséquence. Cette réaction peut prendre la forme d'une commande envoyée aux effecteurs. Cela peut être également par exemple la génération d'un stimulus interne permettant le déclenchement d'autres schémas. Le point important est que la perception d'un schéma n'est pas générale mais est guidée par le rôle qu'il doit tenir. Les schémas de type *perception action* sont réalisés sous formes de fonctions de potentiels. Lorsque plusieurs schémas sont actifs simultanémentΓla commande finale est la somme des différentes commandes. Lorsque plusieurs schémas sont en conflitΓla somme des commandes peut s'annuler provoquant ainsi une situation de minimum local. La solution proposée consiste alors en un échange de messages entre les différents protagonistes de manière à résoudre le problème.

Considérons par exemple que le robot suit une trajectoire prédéfinie. Un obstacle est présent sur le chemin. Le schéma d'évitement d'obstacles va s'activer

et va entrer en conflit avec le schéma de suivi de chemin toujours actif. Le robot va alors se bloquer dans un minimum local de la fonction potentiel totale somme des deux autres. Après envoi d'un message le schéma de suivi de chemin va temporairement se désactiver modifiant ainsi la carte du potentiel total et permettant au robot de repartir.

Au lieu d'avoir une seule fonction de potentiel résolvant tous les problèmes simultanément Arkin propose d'avoir recours à plusieurs fonctions spécialisées qu'il peut combiner créant ainsi un vaste ensemble de fonctions différentes. Cela lui donne un recours en cas de problèmes où il peut transformer sa fonction en influant sur les différentes composantes.

4.2 Génération de chemins par potentiels

Les approches à base de potentiels telles qu'elles ont été introduites à l'origine et telles que nous les avons présentées au cours de la section précédente considèrent que le robot est une particule élémentaire se déplaçant dans l'espace des configurations sous l'action de forces provenant des obstacles et du but à atteindre. Pour chaque configuration q la particule est soumise à une force $\vec{F}(q)$ déterminant son accélération. Connaissant les équations dynamiques du robot il est alors possible de déterminer le couple à appliquer à chacune de ces articulations pour que le comportement du point contrôlé soit effectivement identique à celui de la particule considérée.

Si l'on dispose par avance d'un modèle de l'environnement et des différents obstacles il est possible de simuler le comportement de la particule afin de planifier un chemin libre permettant de rejoindre le but. On obtient ainsi une méthode de planification de chemin basée sur les potentiels. Il est très important de remarquer que la philosophie utilisée par ces planificateurs est identique à celle utilisée par les approches d'évitement en ligne d'obstacles. La différence fondamentale est que dans un cas le résultat est directement une commande envoyée au robot. Dans le second cas le résultat est un chemin qu'il convient ensuite de soumettre à un système de contrôle afin que le robot puisse le parcourir. Les méthodes de planification par potentiel sont des méthodes de haut niveau. Elles connaissent de part leur facilité de mise en œuvre un très grand succès en robotique. Nous allons pour cela en donner très brièvement une description bien que ne faisant pas directement partie du cadre de notre travail.

4.2.1 Planification en Profondeur

Dans l'approche "Planification en Profondeur[1]" un chemin est représenté sous forme d'un ensemble de segments de droite partant de la configuration initiale q_{init}. Soit q_i la configuration atteinte par le segment $i-1$. Le sommet q_{i+1}

1. En anglais: Depth-First Planning

du segment i est alors choisi de manière à ce que le segment $q_i q_{i+1}$ soit dans le sens de l'opposé du gradient de la fonction potentiel en q_i :

$$\begin{cases} x(q_{i+1}) = x(q_i) - \delta_i \frac{\partial U}{\partial x}(x,y,\theta) \\ y(q_{i+1}) = y(q_i) - \delta_i \frac{\partial U}{\partial y}(x,y,\theta) \\ \theta(q_{i+1}) = x(q_i) - \delta_i \frac{\partial U}{\partial \theta}(x,y,\theta)[2\pi] \end{cases}$$

Le chemin ainsi généré suit la pente la plus forte afin de diminuer la fonction potentiel de la configuration initiale jusqu'au but à atteindre. Dans le cas d'environnements simplesΓcette méthode permet de générer un chemin très rapidement. Dans le cadre d'environnements plus complexesΓelle est susceptible d'être mise en échec par la présence de minima locaux.

Comme nous l'avons indiqué au cours de la section précédenteΓces minima locaux peuvent être combattus de deux manières différentes :

1. par la mise en place d'un algorithme de recherche permettant de sortir de ces points singuliers.

2. par l'utilisation de fonctions de potentiels ne possédant que le but à atteindre comme minimum.

Le premier algorithme présenté ci-dessous s'appuie sur la première approcheΓ les suivants sur la seconde.

4.2.2 Planification par le Meilleur d'Abord

La planification par "le Meilleur d'abord[2]" consiste à construire itérativement un arbre T des configurations visitées. Le chemin est ensuite extrait de cet arbre. A l'initialisationΠl'arbre T contient la configuration initiale q_{init}. A chaque itérationΓl'algorithme examine les voisins de la feuille de T ayant la valeur de potentiel minimum. Ceux non visités sont ajoutés à l'arbre comme successeur de cette feuille. La construction se termine lorsque le but est atteint. Le chemin est alors extrait en remontant du but vers la racine.

4.2.3 Planification par fonction de navigation numérique

Cette approche a été proposée par Barraquand et Latombe (voir [109]). Elle consiste tout d'abord à construire les différentes valeurs de la fonction de navigation sur l'espace des configurations puis à exploiter cette fonction avec un algorithme de type Planification par le Meilleur d'Abord par exemple.

Afin de déterminer les différentes valeurs prisent par la fonctionΠl'espace des configurations est tout d'abord discrétisé sous forme de grille. A l'initialisationΓ la cellule contenant le but reçoit la valeur 0. Ces voisines directes faisant partie

2. En anglais: Best-First Planning

de l'espace libre et non encore affectées reçoivent la valeur 1. Les voisines de ces nouvelles cellules reçoivent ensuite la valeur 2 et ainsi de suite jusqu'à atteindre la position initiale du robot. La propagation de ces valeurs (correspondant pour chaque cellule à la distance la séparant du but) est réalisée par un algorithme de type "wavefront expansion".

4.2.4 Utilisation de fonctions harmoniques

Nous allons examiner cette approche récente un peu plus en détails car elle met en évidence un schéma à notre connaissance unique où les notions de réactivité et de planification sont intimement liées au sein du même algorithme.

Connolly et Grupen [43Γ44] se sont intéressés à l'utilisation d'une nouvelle famille de fonctions de potentiel Φ vérifiant l'équation de Laplace (équation 4.3). Les fonctions solutions de cette équation sont appelées *fonctions harmoniques.*

$$\nabla^2\Phi = \sum_{i=1}^{n} \frac{\partial^2 \Phi}{\partial {x_i}^2} = 0 \tag{4.3}$$

Le principe du planificateur est le suivant : l'espace des configurations est tout d'abord discrétisé par l'utilisation d'une grille. La valeur du potentiel Φ est ensuite fixée à une valeur faible pour le but à atteindre. Le potentiel en tout point vérifiant l'équation 4.3 est alors calculé grâce à la méthode itérative de JacobiΓ remplaçant simultanément à chaque pas la valeur de toutes les cellules autres que le but et les obstacles par la moyenne de ses voisines directes. Il est possible de fixer des conditions sur la forme de la fonction résultante en bordure d'obstacles ($\partial\Omega$). Les deux conditions retenues sont :

1. la condition de Dirichlet : $\Phi|_{\partial\Omega} = c$. Le potentiel de la bordure d'un obstacle est fixé à une valeur constante importante. Les lignes de courant de la fonction résultat Φ_D s'écoule alors perpendiculairement à la surface de l'obstacleΓtendant à s'en éloigner.

2. la condition de Neumann : $\left.\frac{\partial \Phi}{\partial n}\right|_{\partial\Omega} = 0$ où n représente la normale à la surface de l'obstacle. La condition de Neumann contraint le gradient de Φ à être tangentiel à l'obstacle. Les lignes de courant s'écoulent donc le long de la surface. La fonction résultat est notée Φ_N.

La fonction Φ_D fournit un chemin s'éloignant le plus possible des obstaclesΓ privilégiant ainsi un chemin sûrΓalors que la fonction Φ_N au contraire se déplace le long de leur surfaceΓ privilégiant un chemin plus court. Un comportement intermédiaire peut alors être obtenu par combinaison :

$$\Phi = k\Phi_D + (1-k)\Phi_N$$

La fonction Φ ainsi obtenue est également trivialement une fonction harmonique.

Le choix d'une fonction harmonique comme fonction de potentiel est rendu particulièrement intéressant par les deux propriétés suivantes :

- une fonction harmonique ne possède pas de minimum local en dehors du but à atteindre. Ceci peut être démontré par l'utilisation du principe *min-max* stipulant que quel que soit la région choisie de l'espace libre n'englobant pas le but, la fonction potentiel prend sa valeur minimale et sa valeur maximale locale sur la frontière de cette région. En fait, les seuls points singuliers (gradient nul) que la fonction potentiel est susceptible de rencontrer sont des points de type "col" dont il est facile de s'échapper en basculant d'un côté ou de l'autre.

- la méthode proposée est complète vis-à-vis de la taille des cellules de la grille. Cela signifie que pour une discrétisation donnée de l'environnement, si un chemin existe, il sera trouvé. On peut démontrer que les zones de l'espace des configurations non connectées au but sont des zones où le potentiel est constant et où donc le gradient est nul.

L'algorithme précédemment décrit permet de déterminer une fonction potentiel adaptée à un modèle de l'environnement. Cette fonction peut ensuite être utilisée pour construire un chemin mais également directement pour commander un robot manipulateur par exemple. La vitesse de l'effecteur est alors déterminée par le gradient de la fonction harmonique :

$$\dot{\vec{q}} = K \nabla \phi$$

Si le modèle de l'environnement n'est pas complet, le robot risque de rentrer en contact avec un obstacle imprévu lors de son déplacement. Lors du contact, la stratégie suivie consiste alors à se déplacer le long des équipotentielles de manière à changer de ligne de champs et à en sélectionner une ne coupant pas l'obstacle. La loi de commande suivie est donc :

$$\dot{\vec{q}} = K_{\nabla} \nabla \Phi + K_{eq} A(\vec{q}) \vec{w} \tag{4.4}$$

$\vec{w}$ représentant la force de contact et A la matrice permettant de transformer une force de contact en un vecteur vitesse permettant de se déplacer le long d'une équipotentielle (perpendiculairement aux lignes de champs).

L'étape suivante a ensuite consisté à intégrer la phase planification et la phase évitement réactif en une seule et même phase. L'idée est la suivante : le robot dispose à l'origine d'une description grossière de l'environnement. Une première fonction harmonique est calculée permettant au robot de se déplacer vers son but. Lors de la détection d'un obstacle imprévu, la deuxième partie de l'équation 4.4 permet d'explorer l'obstacle. A chaque pas effectué, une nouvelle partie de l'obstacle est découverte et entrée dans le modèle. Un ou plusieurs cycles de l'itération de Jacobi sont effectués, entraînant une modification de la fonction harmonique.

Au fur et à mesure que l'environnement est découvert grâce à l'exploration réactive la fonction harmonique et donc les chemins empruntés par le robot ainsi que les déplacements effectués pour l'exploration sont modifiés pour tenir compte des nouvelles données. Les deux notions de comportement réactif (exploration de l'obstacle) et de détermination d'un chemin vers le but sont ici réunies dans un même algorithme et dans un même formalisme faisant disparaître la séparation classique planification - action.

L'approche que nous venons de décrire semble être une approche très intéressante de part la fusion de la planification et de l'action en une seule étape et de part les propriétés mathématiques des fonctions harmoniques garantissant la complétude de l'approche tout en permettant un choix de comportements allant du trajet le plus rapide au trajet le plus sûr.

4.2.5 Planification variationnelle

Cette approche de la planification n'est pas basée sur le suivi d'une particule cherchant à minimiser la valeur d'une fonction. Cette méthode nécessite la définition d'une fonctionnelle J associant un nombre à un chemin. Le principe de la planification est de déterminer le chemin τ minimisant J.

Par exemple soit U une fonction de potentiel classique. J peut être définie comme [109] :

$$J(\tau) = \int_0^1 U(\tau(s))ds + \int_0^1 \|\frac{\vec{d\tau}}{ds}\|ds$$

Le premier terme a pour but de fournir un chemin évitant les forts potentiels et donc les obstacles. Le second terme permet de produire des chemins courts.

La fonctionnelle J étant également minimisée par une approche de type descente de gradient l'optimisation peut être bloquée par un minimum local ne correspondant pas à un chemin libre. L'avantage d'une telle approche réside néanmoins dans la possibilité de coder facilement grâce à J un ensemble de critères que l'on souhaite retrouver dans le chemin généré.

4.2.6 Planification par déformation d'un chemin

La dernière possibilité que nous indiquerons dans cette section afin de contourner le problème du minimum local est la déformation de chemins existants telle qu'elle a été proposée par Warren [183]. Le principe consiste à approcher le chemin que l'on cherche par un ensemble de segments de droite. A l'initialisation le chemin relie le but à l'origine en traversant éventuellement un certain nombre d'obstacles. Le principe des potentiels ici est de chasser les points de l'intérieur vers l'extérieur des obstacles et de déformer en conséquence le chemin.

Toutes les méthodes précédemment décrites font appel à une modélisation de l'environnement et plus particulièrement des différents obstacles (sous formes

de primitives géométriques telles que les ellipsoïdes dans le cas de Kathib [97]). Cette hypothèse de la connaissance ou de l'acquisition d'un modèle des obstacles est une hypothèse forte que nous allons relâcher maintenant.

4.3 Utilisation de champs de forces

Le but des approches de type potentiel est de coder les objectifs de navigation dans une fonction :

- atteindre le but.
- éviter les obstacles.

Cette fonction Φ dont la valeur dépend de la configuration du robot doit décroître lorsque l'on se rapproche de l'objectif et croître si le comportement du robot est contraire aux spécifications. Le principe de la navigation par potentiel consiste à créer un contrôleur dont les commandes auront pour but d'amener le robot dans une configuration minimisant Φ la minimisation de la fonction assurant la réussite de la tâche à accomplir. Le problème est donc un problème d'optimisation de fonction.

La méthode retenue dans les approches décrites précédemment est la méthode classique de la descente de gradient. Pour toute configuration q $\Phi(q)$ indique la valeur du potentiel en ce point et $-\nabla\vec{\Phi}(q)$ est un vecteur indiquant dans quelle direction il faut se déplacer pour réduire cette valeur. Une fonction de potentiel est classiquement la somme d'un potentiel attractif Φ_g associé au but à atteindre et d'un potentiel répulsif Φ_o associé aux différents obstacles. Dans le cas du but on sait trivialement qu'il faut se diriger vers celui-ci afin de réduire la valeur de Φ_g. Dans le cas des obstacles il est nécessaire de connaître leurs formes afin de déduire le gradient de Φ_o et donc la direction à suivre.

Si on se place maintenant dans une hypothèse de lien direct perception-action on ne dispose plus de description de la forme des différents obstacles. La fonction potentiel réalisant en quelque sorte un codage de ceux-ci il n'est plus possible de la calculer directement. On utilisera plutôt une approche déterminant directement le vecteur force provenant de chaque mesure capteur (vecteur destiné à éloigner le robot de l'obstacle potentiel). Le vecteur force final est alors la somme de tous les vecteurs forces élémentaires (utilisation de plusieurs capteurs par exemple). Les approches répondant à ce principe seront appelées *méthodes à base de champs de forces* et non pas méthodes à base de potentiels la notion de potentiel étant liée à la notion de gradient.

Ecartons temporairement l'analogie physique en terme de force d'attraction et de forces de répulsions pour reconsidérer le problème de minimisation d'une fonction Φ codant l'objectif : atteindre le but en évitant les obstacles. D'un point de vue strict l'absence de connaissance de la forme des obstacles interdit le

calcul d'un gradient (sauf dans le cas particulier des potentiels généralisés comme nous le verrons ci-dessous). On dispose pour chaque configuration du robot d'une mesure de la valeur de $\Phi = \Phi_g + \Phi_o$. C'est une donnée absolue ne fournissant pas d'indication de direction à suivre afin de la réduire. Dans le problème présentΓ on sait néanmoins trivialement que pour réduire Φ_g il suffit de se déplacer vers le but et que pour réduire Φ_o Γil suffit de se déplacer dans la direction inverse de la mesure indiquée par un capteur. Cette opposition entre donnée absolue et donnée absolue + direction est similaire à celle que l'on retrouve en apprentissage en contrôle entre l'apprentissage renforcé ou distant d'une part et l'apprentissage supervisé d'autre part (voir chapitre 6).

Il existe un cas particulier par rapport à ce que nous venons de dire. Dans le cas de la méthode des potentiels généralisés [106]ΓΦ$_o$ dépend en fait de la vitesse du robotΓde ces caractéristiques dynamiques et de la distance à l'obstacle frontal et non de sa forme. Il est possible de calculer analytiquement le vecteur gradient en fonction de la configuration du robot et de la mesure retournée par les capteurs frontaux.

4.3.1 La méthode des champs de forces virtuels

La méthode des champs de forces virtuels[3] a été développée par Borenstein et Koren [25]. Son but est de permettre à un robot mobile "rapide" de contourner un obstacle imprévu détecté par ses capteurs. Les mesures des différents capteurs sont intégrées dans une grille de dimension 2 appelée *grille histogramme*. Cette grille est inspirée des travaux de Moravec et Elfes [124] mais seule la cellule correspondant à la mesure d'un capteur a son degré d'activation incrémenté de manière à obtenir une mise à jour plus rapide.

La grille ainsi calculée représente une carte globale dans un repère absolu de l'environnement. Lorsque le robot se déplaceΓde manière à limiter son champ de vueΓseuls les échos capteurs situés dans une fenêtre de dimension $w_s \times w_s$ sont considérés (voir figure 4.2). Chaque cellule (l, m) de la grille exerce sur le véhicule une force $\vec{F}_{lm}$ dirigée vers le robot et dont l'amplitude est fonction du degré d'activation de la cellule ainsi que de la distance la séparant du véhicule :

$$\vec{F}_{lm} = \frac{F_{cr} W^n C_{lm}}{d^n(l,m)} \left(\frac{x_l - x_0}{d(l,m)} \vec{i} + \frac{y_m - y_0}{d(l,m)} \vec{j} \right)$$
$$\vec{F}_r = \sum \vec{F}_{lm}$$

où F_{cr} représente une force constanteΓW la largeur du véhiculeΓC_{lm} le degré d'activation de la celluleΓ(x_0, y_0) les coordonnées du robotΓ(x_l, y_m) les coordonnées de la cellule et $d(l, m)$ la distance séparant la cellule du robot.

Simultanément à cette force de répulsion $\vec{F}_r$Γle robot est également soumis à

3. VFF ou Virtual Force Field

une force d'attraction provenant du but $\vec{F_t}$:

$$\vec{F_t} = F_{ct}\left(\frac{x_t - x_0}{d_t}\vec{i} + \frac{y_t - y_0}{d_t}\vec{j}\right)$$

où F_{ct} représente une force d'attraction constante$\Gamma(x_t, y_t)$ les coordonnées du but et d_t la distance séparant le robot du but. La direction du vecteur total $\vec{R} = \vec{F_t} + \vec{F_r}$ indique la direction devant être suivie par le robot. La vitesse linéaire et quant à elle fonction de l'angle θ entre la vecteur vitesse courant et F_r:

$$\begin{cases} V = V_{max} & \text{si } \|F_r\| = 0 \\ V = V_{max}(1 - cos(\theta)) & \text{sinon} \end{cases}$$

La vitesse est maximum si aucun obstacle n'est présent ou s'ils sont tous latéraux ($cos(\theta) = 0$).

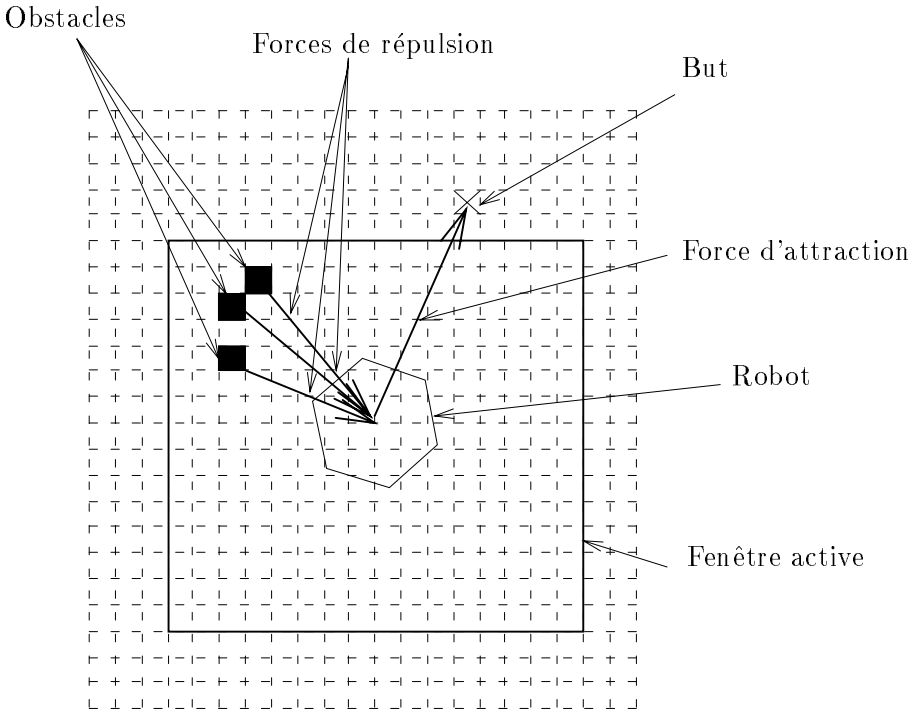


FIG. 4.2 - : *Définition du champ de forces virtuelles*

Une méthode similaire a également été appliquée par [1]. Le calcul des vecteurs dans ce cas ne provient pas des cellules actives d'une grille mais directement des mesures provenant d'un capteur infra-rouge rotatif.

Nous allons maintenant décrire le système que nous avons réalisé dans le cadre des champs de forces.

4.4 Approche proposée

Le système que nous allons maintenant décrire a été réalisé en 1991 en collaboration avec Frank Wallner dans le cadre de son mémoire d'ingénieur [181]. Il est basé sur une approche de type champs de vecteurs.

4.4.1 Principe général

Un des problèmes principaux des méthodes à base de champs de vecteurs est la présence de minima locaux. Ces minima peuvent être causés par :

- une configuration particulière de l'environnement. La zone d'influence de ce minimum peut être plus ou moins importante.

- une mauvaise lecture capteur faisant croire au robot qu'il est dans une situation de blocage. Une mauvaise lecture peut être due à un bruit électronique. Elle est alors en général passagère. Elle peut également être due à une configuration particulière de l'environnement (aspérité sur le mur, présence d'une porte vernie, ...). La mesure est alors fausse mais constante dans le temps si le robot est immobile.

Dans le premier cas, le mouvement nécessaire afin de sortir du minimum peut aussi bien être de faible amplitude qu'important. Dans le second cas en revanche, un simple mouvement suffit généralement afin de changer le point de vue pour les capteurs et éliminer la source de bruit.

Pour les raisons évoquées ci-dessus, nous avons décidé de privilégier le mouvement dans notre système. La vitesse de translation et de rotation du véhicule est fixée par le niveau supérieur (utilisateur humain ou superviseur symbolique). Ces valeurs sont utilisées par défaut par le contrôleur dont la tâche est de gérer le cap du véhicule. Ce cap est déterminé par la direction du vecteur F_{comm} résultant du champ de force :

$$\vec{F}_{comm} = \vec{F}_{but} + \vec{F}_{obstacles}$$

Le contrôleur n'ayant pas d'influence directe sur la vitesse linéaire du robot, favorisant ainsi l'action, il est nécessaire de mettre en place un système de sécurité bas niveau destiné à limiter les risques de collision. Ce système bas niveau est un module réflexe provoquant l'arrêt du robot lorsque qu'un capteur détecte la présence d'un obstacle dans la direction de déplacement (translation ou rotation) à une distance inférieure à la distance de sécurité. Le système réactif reprend ensuite le contrôle du véhicule.

Nous allons maintenant détailler les différentes composantes de notre approche.

4.4.2 Détermination de $\vec{F}_{but}$

Le rôle de ce vecteur étant de permettre au robot de rejoindre le but désignéΓ il est tout naturellement orienté du centre du véhicule vers le point à atteindre. Plusieurs formules existent en revanche quant au calcul de sa norme. Elle peut être constante sur la totalité de l'espace ou décroître linéairement par exemple au fur et à mesure que le robot s'approche. La seconde possibilitéΓcomme l'a souligné Latombe [109] possède de bonnes propriétés de stabilisation du robot sur le but. En revancheΓsa zone d'influence est limitée (la norme décroît vers 0 lorsque l'on s'éloigne du point à atteindre). La première possibilité présente une zone d'influence illimitée et homogène mais n'est en revanche pas capable de stabiliser le robot au niveau du but. Nous l'avons néanmoins retenue en ajoutant un mécanisme supprimant la navigation réactive et stoppant le robot lorsque celui-ci est à proximité suffisante du point recherché.

$$\vec{F}_{but} = F(cos\alpha\vec{i} + sin\alpha\vec{j})$$

où F est une constanteΓ$\vec{i}$ et $\vec{j}$ les deux vecteurs unitaires attachés au robot et α l'angle vers le but (voir figure 4.3)

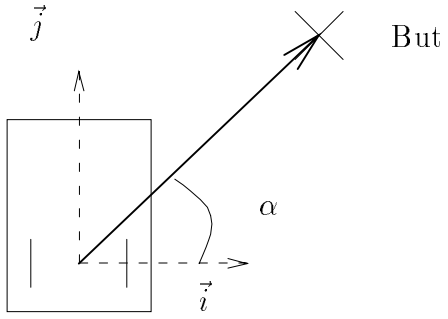


FIG. 4.3 - : *Définition de la force* $\vec{F}_{but}$

4.4.3 Détermination de $\vec{F}_{obstacles}$

De manière à simplifier la tâche de modélisation de l'environnement et d'obtenir un lien direct entre l'action et la perceptionΓnous n'avons pas retenue la solution faisant appel à une grille d'occupation telle que celle proposée par Borenstein [25]. Nous avons au contraire considéré les données provenant directement des capteurs et avons associé une force $\vec{F}_i$ à chacune d'entre elles. La force totale $\vec{F}_{obstacles}$ est la somme de toutes ces forces élémentaires.

Les capteurs ultrasons sont répartis sur la ceinture du robot de part et d'autre du centre de rotation. Supposons qu'un obstacle soit détecté sur le côté droit du véhicule. Si cet obstacle est détecté par un capteur situé en avant du centre de rotation, le véhicule devra tourner vers la gauche pour s'en écarter. Si en revanche, la détection est réalisée par un capteur situé en arrière, il devra tourner vers la droite. La direction du vecteur de force dépend donc de la position du capteur par rapport à ce centre (voir figure 4.4).

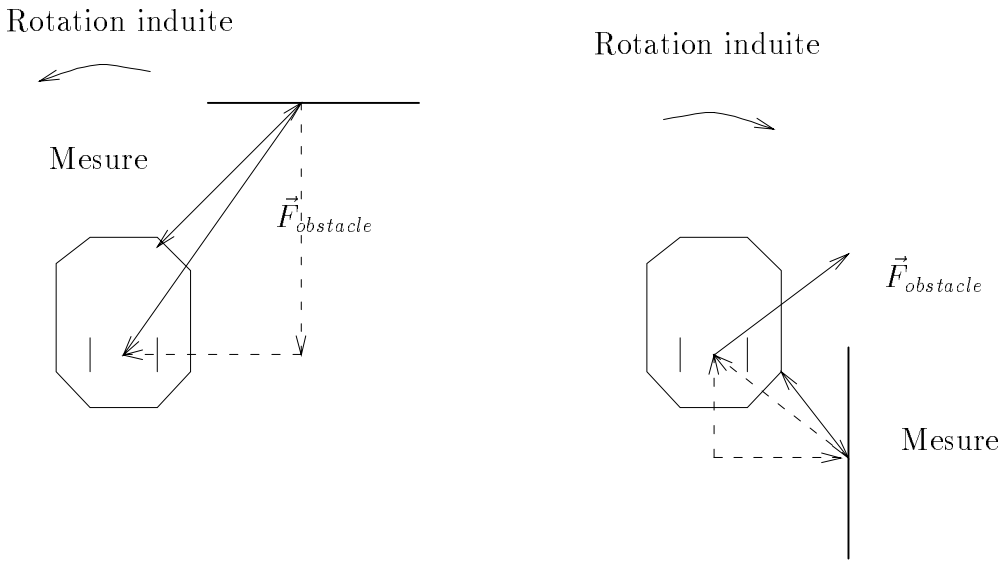


FIG. 4.4 - : *Direction des vecteurs forces associés aux obstacles en fonction du capteur responsable de la mesure*

De manière classique, la norme du vecteur de force doit augmenter si on s'approche de l'obstacle et diminuer dans le cas contraire. Nous avons choisi la fonction suivante :

$$\|\vec{F}_i\| = \frac{|F_{cte}|}{(d_i - d_{min})^n} \tag{4.5}$$

où d_{min} représente la distance minimum à conserver entre le robot et les obstacles, F_{cte} une force constante de répulsion et d_i la distance mesurée par le capteur i. Dans le cadre de notre application, d_{min} a été fixée à la distance minimum pouvant être mesurée physiquement par les capteurs ultrasons utilisés : $12cm$. Afin de déterminer les valeurs des deux autres constantes n et F_{cte}, nous avons considéré la situation où le robot se déplace face à un mur, dans la direction de son but. Etant donné la répartition des capteurs, cet obstacle peut être détecté par un ou plusieurs des 9 capteurs frontaux (voir figure 2.7). Les deux constantes

sont alors calculées considérant les deux distances suivantes :

- la distance d_1 pour laquelle la détection du mur par un seul capteur (situation la plus défavorable) suffit à compenser la force d'attraction du but (et donc à arrêter le robot). Il est clair que si plus de capteurs détectent le mur la distance d d'équilibre entre l'attraction et la répulsion est alors supérieure à d_1.

- la distance d_2 où la détection du mur par les 9 capteurs permet de compenser la force d'attraction du but. Cette distance permet de fixer la proximité entre un obstacle et un but pour que celui-ci puisse être atteint. Si moins de capteurs peuvent détecter l'obstacle le but peut alors être situé à une distance inférieure à d_2 mais de toute façon supérieure à d_1.

4.4.4 Commande générée

Comme nous l'avons indiqué au début de cette section le contrôleur a pour tâche d'imposer la direction du véhicule et n'intervient normalement pas dans la vitesse linéaire. Toutefois nous avons décidé ;

- de stopper la mouvement de translation du robot si le changement de cap est trop important (supérieur à 30^o). Ce module était destiné au robot Mithra qui est un véhicule à chenille. Sur ce type de véhicule l'exécution d'un mouvement combiné de translation et de rotation diminue grandement la précision de la position estimée (on ne peut pas prédire où se trouvera l'axe de rotation) et nécessite un sur-coût énergétique.

- d'exécuter le mouvement en marche arrière (toujours à la vitesse spécifiée) si l'angle du vecteur de commande est supérieur à 165^o en valeur absolue. L'angle est alors remplacé par son complémentaire à 180^o.

4.4.5 Gestion des minima locaux

Comme nous l'avons signalé lors de l'étude bibliographique les minima locaux sont un des problèmes majeurs de telles approches. Le champ de forces que nous avons choisi ne réalise en aucun cas ce que l'on pourrait appeler un *vecteur de navigation* (par similarité avec les fonctions de navigation). Les minima locaux existent. Il s'agit donc de les détecter et de mettre en place quelques heuristiques pour essayer de permettre au robot d'en ressortir.

Un minimum local correspond à un point d'équilibre stable pour le véhicule. Stable signifie que toute tentative de sortie de ce point provoque la génération de commandes l'y ramenant. Une situation potentielle de minimum local sera mise en évidence en détectant une succession de changement de sens dans la commande de direction.

Nous nous sommes intéressés à deux causes principales :

1. le robot doit momentanément s'éloigner du but pour pouvoir l'atteindre.

2. le robot doit rentrer dans un passage étroit (couloir par exemple).

Contournement d'un obstacle : recherche locale

La situation de blocage peut être provoquée par exemple lorsque le robot se trouve parallèle à un murΓle but à atteindre étant situé de l'autre côté (voir figure 4.5). La direction du champ de force va osciller entre 90^o et -90^oΓimposant au robot de ne pas exécuter de translation (demande de rotations de grandes amplitudesΓjamais complètement exécutées par ailleursΓle robot étant rappelé à droite dès qu'il tourne à gauche et inversement).

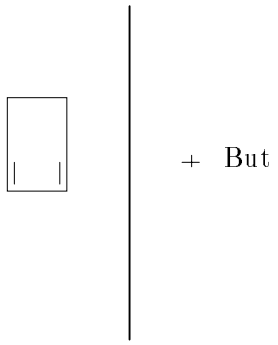


FIG. 4.5 - : *Situation provoquant un minimum local.*

En cas de détection de présence d'un minimum localΓle robot passe en mode recherche. Cela signifie que quel que soit l'angle commandéΓle contrôleur impose une vitesse linéaire constante (sauf si action du module réflexe). Le robot se mettra donc en mouvement et sortira éventuellement de la zone de blocage. Ce mode recherche est supprimé :

– si le but est atteintΓ

– si le robot fait de nouveau face au butΓ

– si le nombre de cycles maximum autorisé en recherche est dépassé.

Le dernier cas signifie que l'on suppose que le robot a quitté la zone d'influence du minimum local et on reprend le mode de fonctionnement normal.

Adaptation de $\vec{F}_{obstacles}$ en fonction de l'environnement

Une des sources classiques de minima locaux pour les méthodes de champs de forces est la présence d'un passage étroit (voir figure 4.6). Le but se trouve dans le couloir mais les forces répulsives provenant des murs sont trop fortes et créent une barrière infranchissable à l'entrée du passage. Le robot est repoussé. De manière à

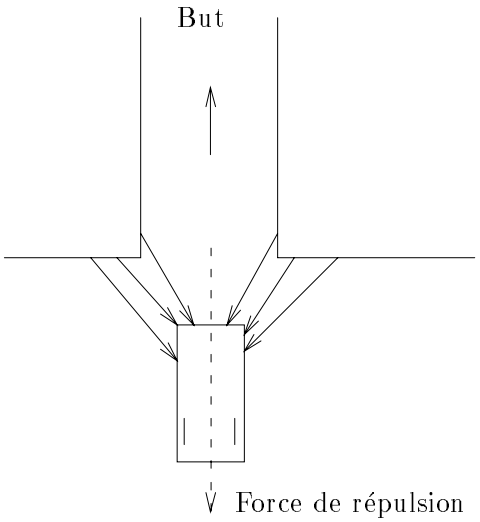


FIG. 4.6 - : *Le robot est repoussé par les deux murs du couloir et ne peut atteindre son but.*

prévenir une telle situation, on observe à chaque cycle la valeur des coordonnées du vecteur force résultant pour la direction gauche et la direction droite sur chaque axe (voir figure 4.7). La présence d'une valeur importante supérieure à un seuil pour F_{gx} et F_{dx} (respectivement F_{gy} et F_{dy}) indique la présence d'une zone étroite dans la direction x (respectivement y). Le champ de force est alors adapté en diminuant à chaque cycle la valeur de F_{cte} (voir équation 4.5) jusqu'à ce que les deux composantes reviennent en dessous du seuil.

La réduction d'intensité du champ de répulsion diminue sa capacité à piloter le robot hors des obstacles. Le contrôle sera donné de plus en plus au module réflexe chargé en dernier recours d'arrêter le robot avant collision. Cette situation doit être temporaire. Dès que la zone étroite est franchie, et donc dès que F_{gx} et F_{dx} (respectivement F_{gy} et F_{dy}) reprennent des valeurs faibles (inférieures à un seuil), F_{cte} est de nouveau augmentée.

4.4.6 Gestion des rotations

Les rotations du robot sont imposées par la position du but et la position des obstacles. En fonctionnement normal, seules de "petites" rotations sont exigées.

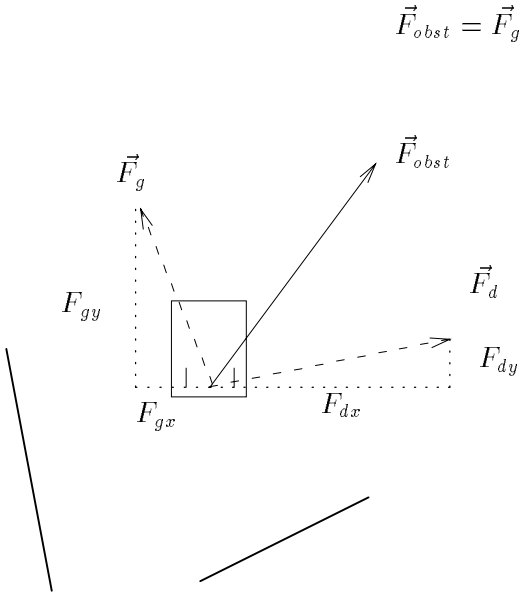


FIG. 4.7 - : *Détection d'une zone étroite en analysant les valeurs respectives de F_{gx} et F_{dx} ainsi que F_{gy} et F_{dy}.*

En revanche, lors de l'apparition d'un nouveau but, le robot peut être amené à changer complètement de direction. Ce changement de direction peut être coûteux en temps voir même être impossible sous la seule action des potentiels (problème de nouveau de minimum local, voir figure 4.8).

De manière à éviter ce problème, nous avons décidé de guider le robot pour exécuter de grandes rotations ($\theta \in [90^o, 270^o]$). Le principe est le suivant :

- on détermine à partir des données capteurs si la rotation est en mesure ou non de se faire.
- si la rotation directe (la plus courte) ne peut être accomplie, on détermine à partir des données capteurs si la rotation inverse peut être exécutée.
- si aucune des deux rotations ne peut être accomplie, on se trouve dans une région de l'environnement trop encombrée pour pouvoir tourner : si cela est possible, on essaie alors de se déplacer en marche arrière afin de pouvoir se dégager (voir prochaine sous-section).

Connaissant maintenant le sens de la rotation ainsi que la valeur de l'angle, il s'agit d'exécuter le mouvement. Le sens ayant été déterminé par une analyse partielle de l'environnement grâce aux capteurs, exécuter directement la rotation peut amener à une collision par un obstacle non détecté initialement. On préférera donc une rotation plus réactive réalisée grâce aux champs de forces. On crée pour

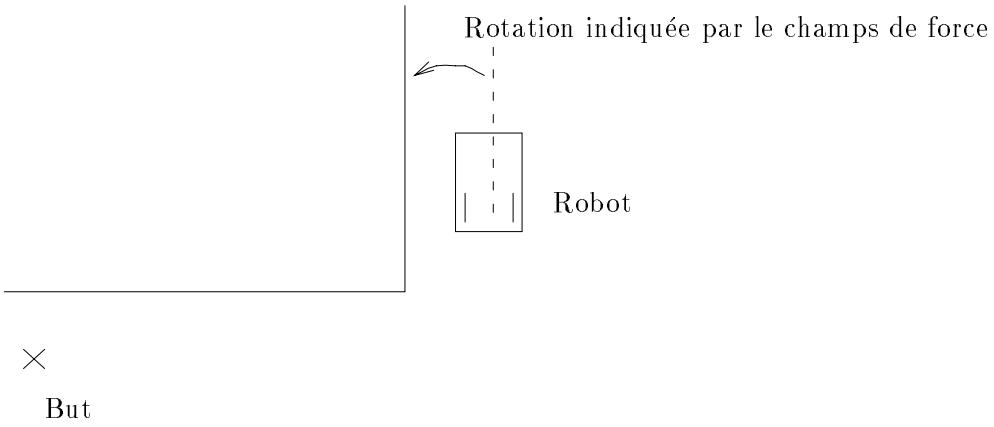


FIG. 4.8 - : *La rotation la plus directe imposée par les potentiels est impossible et provoque une situation de blocage*

ce faire un premier sous-but sur le côté du véhicule correspondant au sens de rotation. Lorsque le robot aura suffisamment tourné et commencera à se déplacer vers celui-ci Γil est alors supprimé et un nouveau est créé pour poursuivre le changement de direction (voir figure 4.9). Remarque : seule la création de 2 sous-buts intermédiaires est nécessaire pour exécuter une rotation de 270^o.

4.4.7 Gestion de la marche arrière

Comme nous l'avons vu précédemment Γsi le robot se trouve dans une région étroite et ne peut pas faire demi-tour pour ressortir Γil exécute alors une marche arrière. Ce mouvement est différent de celui cité section 4.4.4. Il est imposé par la position du but et non par la direction du vecteur de commande. Il sera poursuivi jusqu'à ce que les capteurs détectent suffisamment de place sur la droite ou sur la gauche du robot afin d'exécuter un demi-tour et permettre un repositionnement en marche avant.

4.5 Evaluation

Nous allons illustrer au cours de cette section aux travers de quelques exemples les différents points présentés ci-dessus. Le robot utilisé est un robot simulé.

4.5.1 Evitement d'un obstacle simple

La figure 4.10 illustre la trajectoire suivie par le robot. Le véhicule se déplace en ligne droite. Il rentre dans la zone d'influence de l'obstacle carré Γprovoquant un changement de trajectoire permettant l'évitement.

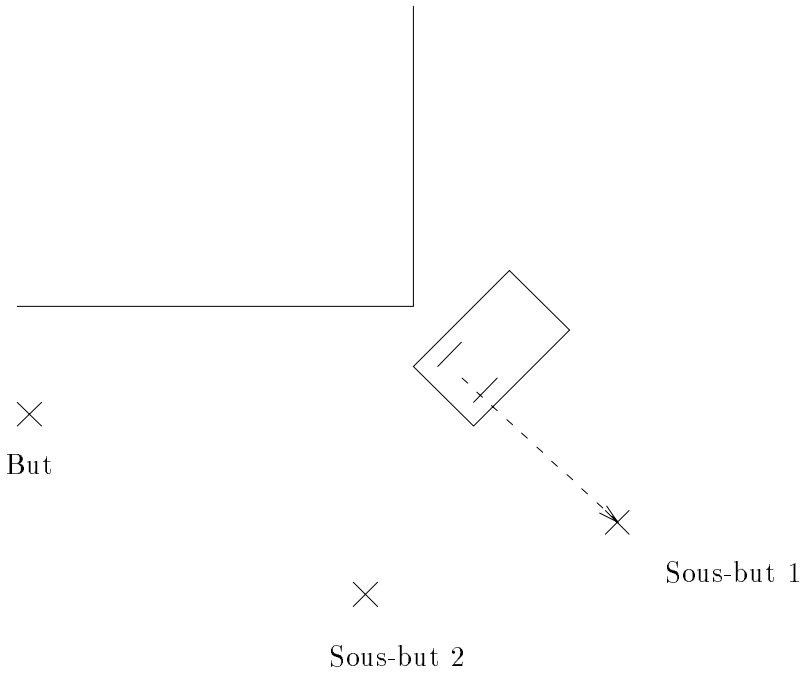


FIG. 4.9 - : *Définition des sous-buts intermédiaires pour le guidage de la rotation*

FIG. 4.10 - : *Contournement d'un obstacle élémentaire*

4.5.2 Illustration de l'adaptation dynamique du champ

Cet exemple a pour but d'illustrer l'adaptation dynamique du champ de potentiel pour le franchissement de passages étroits. Le robot a pour objectif de sortir d'une salle en franchissant une porte.

La première expérience est réalisée en ayant recours à l'adaptation dynamique du champ. Les résultats sont rapportés figure 4.11. La figure en haut à gauche représente la trajectoire du robot. La figure en haut à droite montre l'évolution du paramètre F_{cte}. La valeur augmente lorsque le robot pénètre dans le passage pour redescendre lorsqu'il en ressort. La figure du bas enfin montre l'évolution au cours du mouvement de la distance séparant le robot du but.

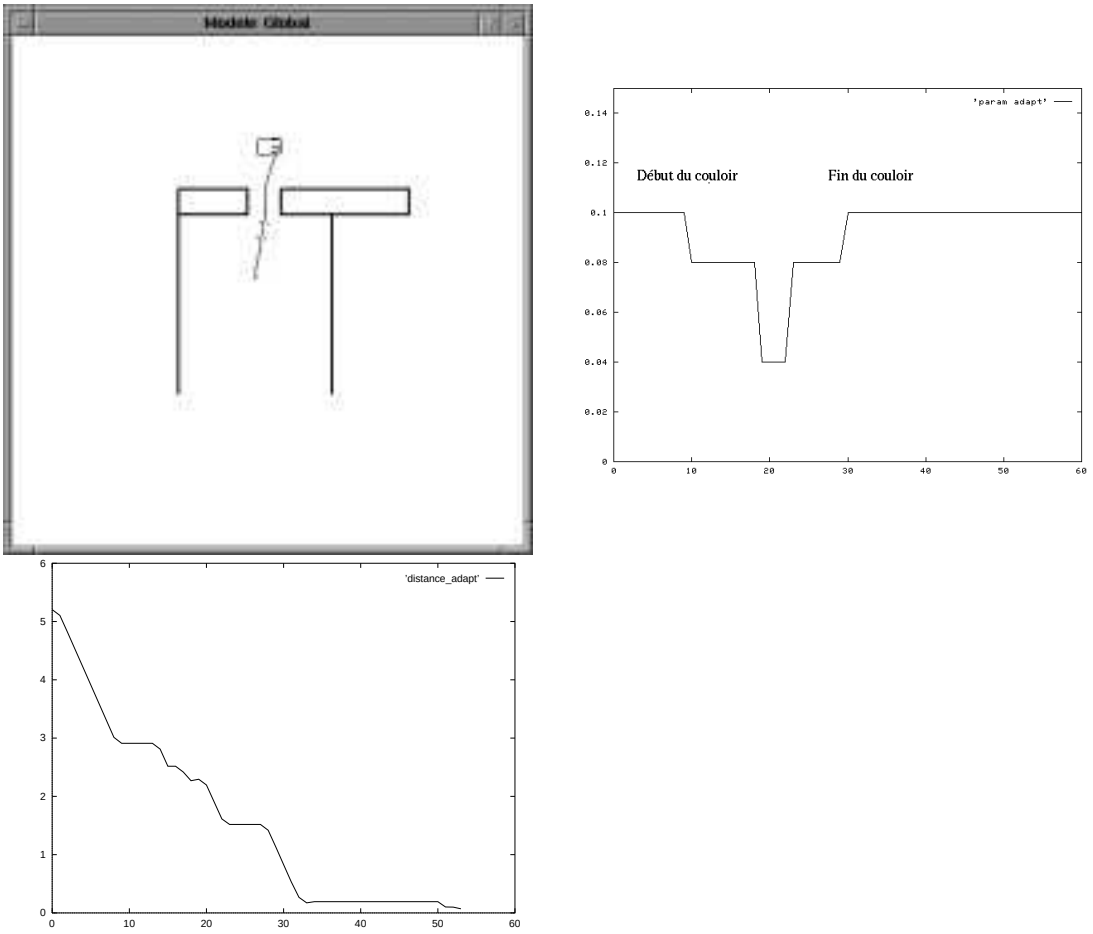


FIG. 4.11 - : *Traversée d'un passage étroit par le robot. La valeur du champ de répulsion est adaptée automatiquement (figure en haut à droite). La figure en bas à gauche représente l'évolution au cours du temps de la distance séparant le robot du but.*

La seconde expérience est identique dans le principe mais on interdit cette fois l'adaptation dynamique du champ. Le robot ne parvient pas à traverser la

porte. Les résultats sont regroupés figure 4.12.

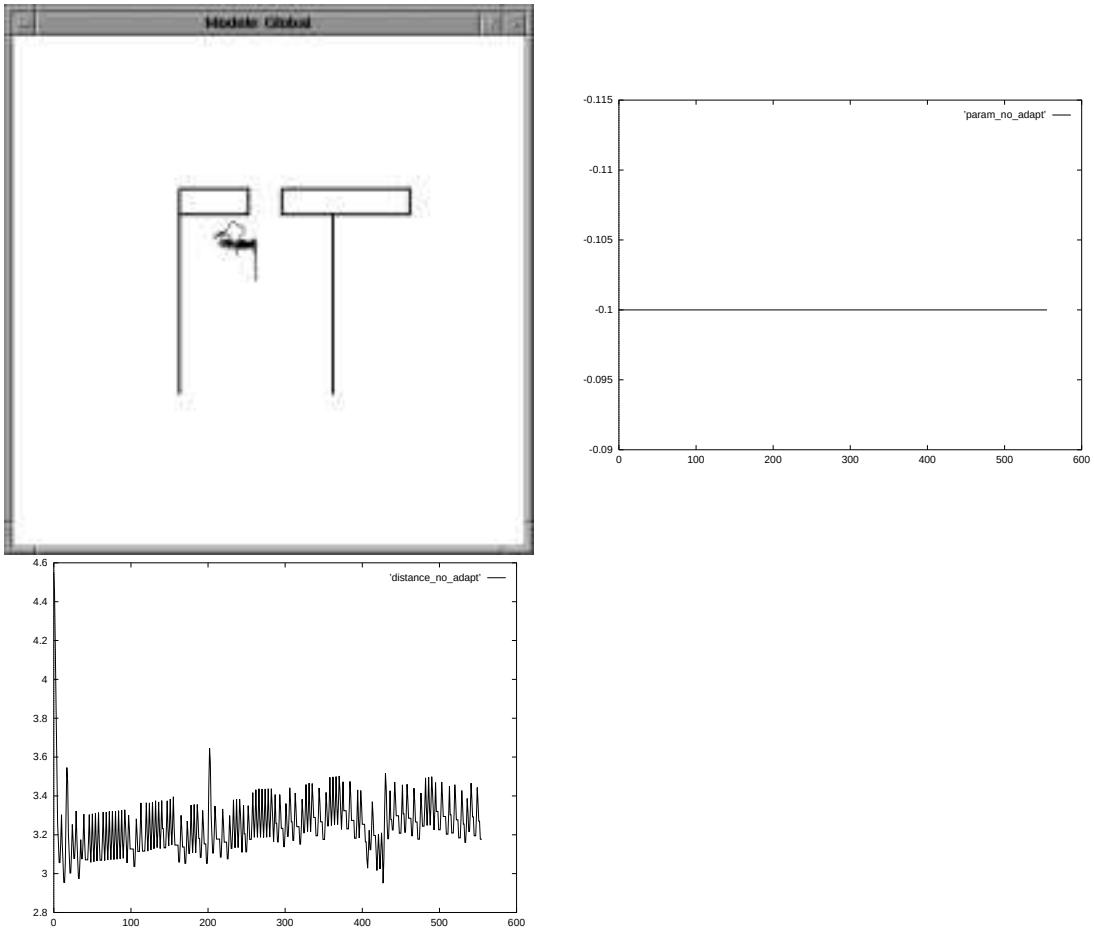


FIG. 4.12 - : *Traversée d'un passage étroit par le robot. La valeur du champ de répulsion est gardée constante (figure en haut à droite). La figure en bas à gauche représente l'évolution au cours du temps de la distance séparant le robot du but.*

4.5.3 Exemple de marche arrière

La tâche du robot est de partir du point AΓde se rendre au point B et revenir enfin au point A (voir figure 4.13). La trajectoire figure 4.14 représente la trace du véhicule lors du parcours *aller*. La figure 4.15 représente trois copies d'écran du chemin retour. Le robot n'a pas la place de faire demi-tour au point B et part en marche arrière (figure du haut). Arrivée au coinΓil détecte une place suffisante et effectue un changement de direction (figure centrale). Il finit ensuite son trajet en marche avant (figure du bas).

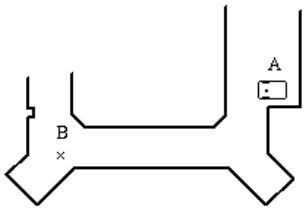


FIG. 4.13 - : *Plan du couloir : le robot doit naviguer entre le point A et le point B.*

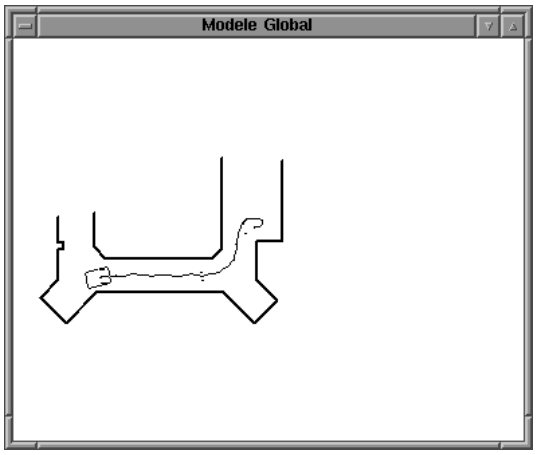


FIG. 4.14 - : *Trajet effectué par le robot pour aller du point A au point B*

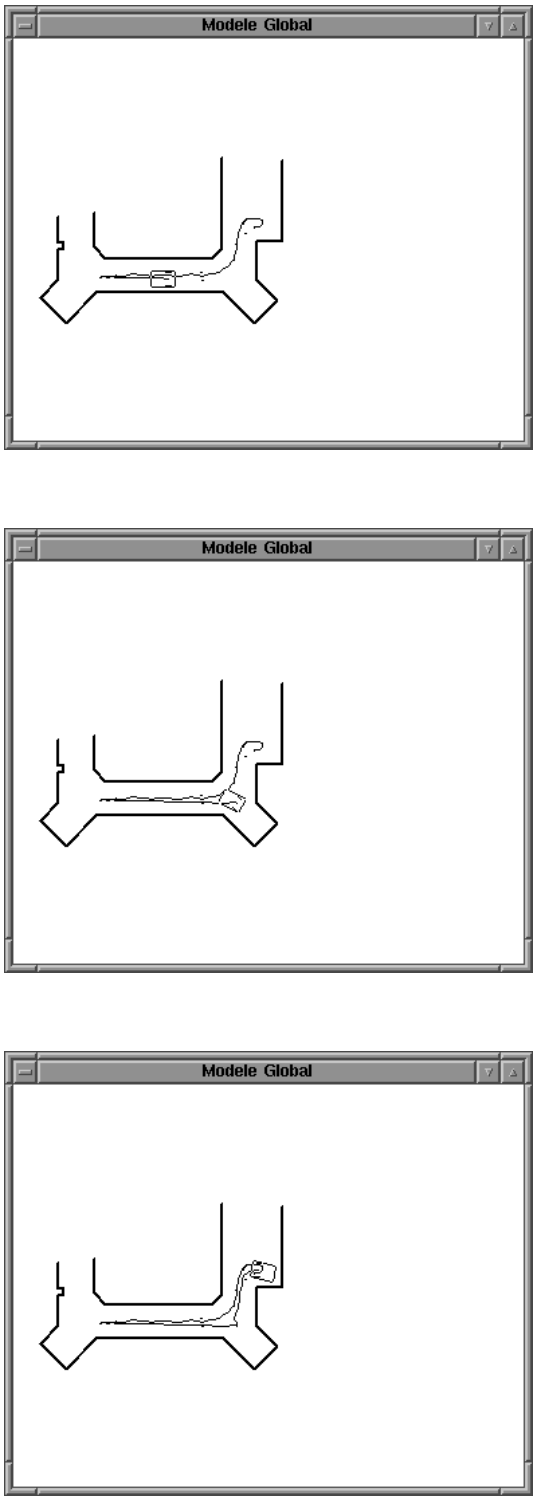


FIG. 4.15 - : *Le robot remonte le couloir en marche arrière (figure du haut), effectue un demi-tour dans l'angle du couloir (figure centrale) avant de rejoindre son but en marche avant (figure du bas).*

4.5.4 Courbe de vitesse

les deux courbes visualisées figure 4.16 représentent respectivement la vitesse linéaire et angulaire en fonction du temps mesurées au cours du déplacement de l'exemple précédent. On peut distinguer sur la première courbe les trois phases du mouvement :

- déplacement en marche avant du point A au point $B\Gamma$
- retour en marche arrière dans le coin du couloirΓ
- reprise de la marche avant afin de rejoindre le point A.

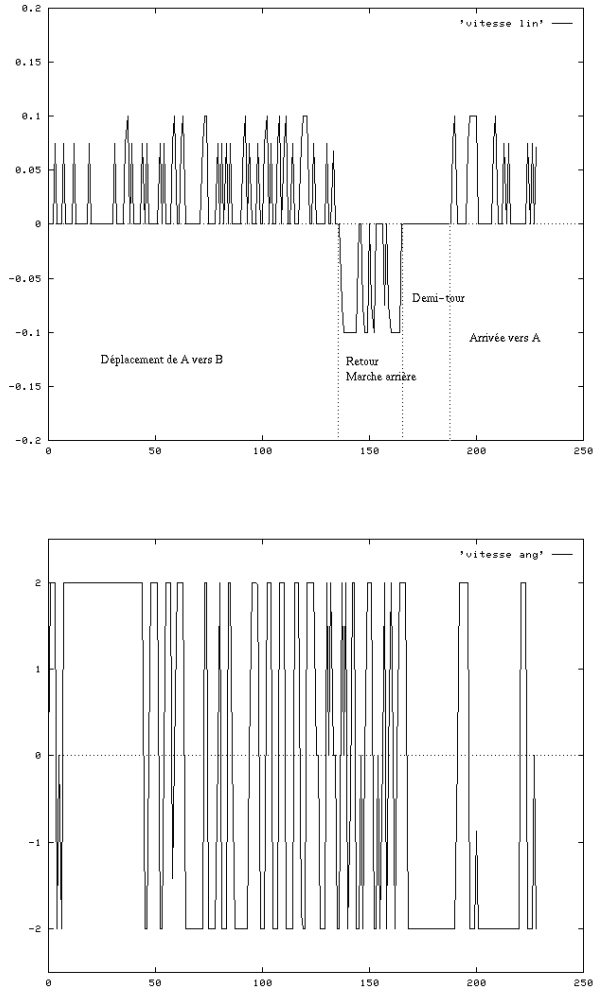


FIG. 4.16 - : *Relevé de la vitesse linéaire (figure du haut) et angulaire (figure du bas) du robot lors du trajet aller-retour entre les deux extrémités du couloir.*

Mais la remarque principale que l'on peut faire en observant ces données est le caractère très irrégulier de la vitesse tant linéaire qu'angulaire du véhicule.

Le principe de l'algorithme est de favoriser le mouvement afin d'éviter au robot de rester bloqué. Le système de contrôle essaie en permanence de générer des ordres de déplacement linéaire à la vitesse spécifiée par l'utilisateurΓle module de réflexe et le contrôleur lui-même stoppant net tout mouvement de translation lorsqu'un obstacle est trop proche ou que l'angle de rotation est trop important. La commande obtenue et de type tout ou rienΓdonnant un comportement hésitant et très peu fluide au robot. Il fait néanmoins remarquer que la simulation utilisée ici est une simulation cinématique et non dynamique. Un système mécanique lourd tel qu'un robot mobile possède une inertie filtrant les effets de commandes de type tout ou rien. Mais le même phénomène de comportement hésitant a également été observé.

4.5.5 Situation d'échec

Le dernier exemple que nous traiterons illustre une situation d'échec. Le robot doit se déplacer de l'autre côté du mur mais ne parvient pas à s'échapper (voir figure 4.17). Cette situation est une situation classique de minimum local pour des approches de type champs de forces. Le contrôleur détecte un point singulier par

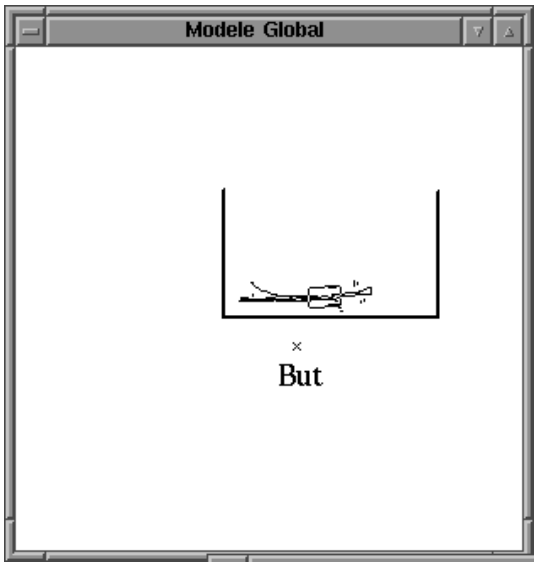


FIG. 4.17 - : *Situation classique de blocage pour les approches de type champs de vecteurs.*

la présence d'un changement de signe continuel de l'angle de rotation demandé. Il passe alors en mode recherche et exécute des mouvements de translation. Le robot ne restera donc pas immobile à la verticale du but mais balayera la région. Les mouvements de translation ne peuvent pas suffire à sortir de la zone d'influence du minimum local. Rappelons toutefois que ce type de problèmes correspond à

celui pouvant être pris en charge par un module de planification (voir figure 4.18).

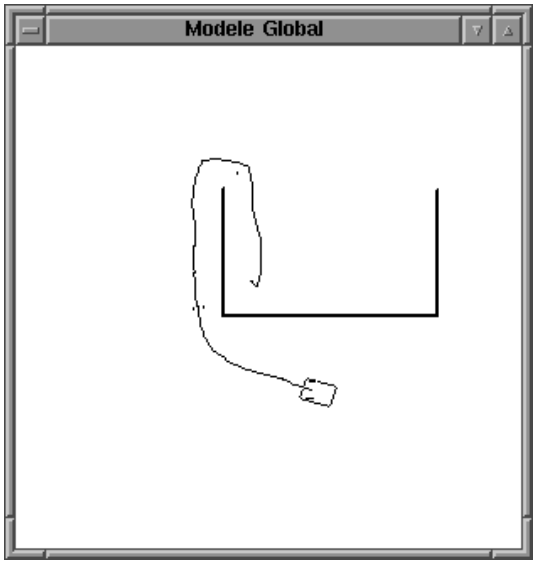


FIG. 4.18 - : *Exemple d'exécution du module de planification couplé au système réactif.*

4.6 Conclusion

Parallèlement aux travaux que nous venons de décrire, Koren et Borenstein ont essayé de caractériser formellement le comportement dynamique d'un robot contrôlé par une approche de type champs de forces virtuelles [102]. Ces travaux sont basés sur une modélisation mathématique du comportement dynamique du véhicule. Les données capteurs sont supposées parfaites et correspondre à ce que le robot verrait en se déplaçant plus ou moins parallèlement à un mur de longueur infini. La résolution des équations différentielles décrivant cette situation a permis de mettre en évidence une instabilité du comportement du véhicule se traduisant sous forme d'oscillations lorsque le robot se trouve confronté à un décrochement du mur qu'il est en train de suivre et pour certaines positions du but à atteindre (voir figure 4.19). Le même phénomène peut être mis en évidence lorsque le robot se déplace dans un couloir se rétrécissant et pour une largeur de couloir inférieure à un certain seuil (voir figure 4.20). Ces oscillations ont pu être mises en évidence mathématiquement grâce aux équations et vérifiées expérimentalement sur un vrai robot.

En plus de ce phénomène, les auteurs citent deux autres problèmes inhérents aux méthodes de type potentiel ou champs de vecteurs :

– présence de minima locaux.

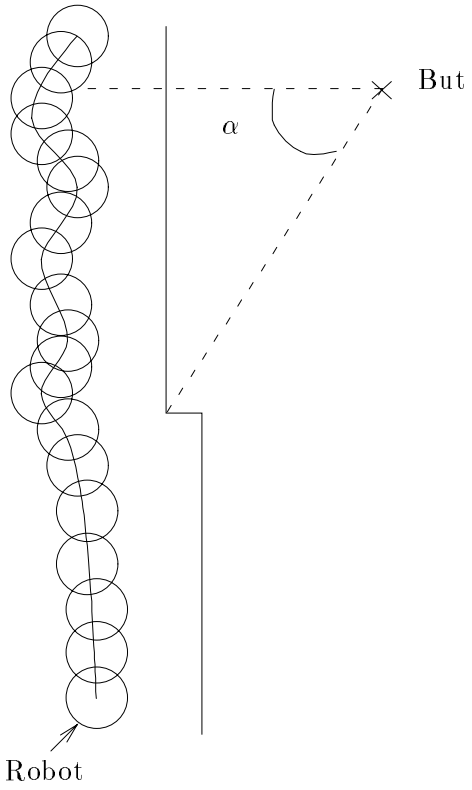


FIG. 4.19 - : *Oscillations du robot lors de la rencontre du décrochement lorsque l'angle α est inférieur à une limite α_{cr}*

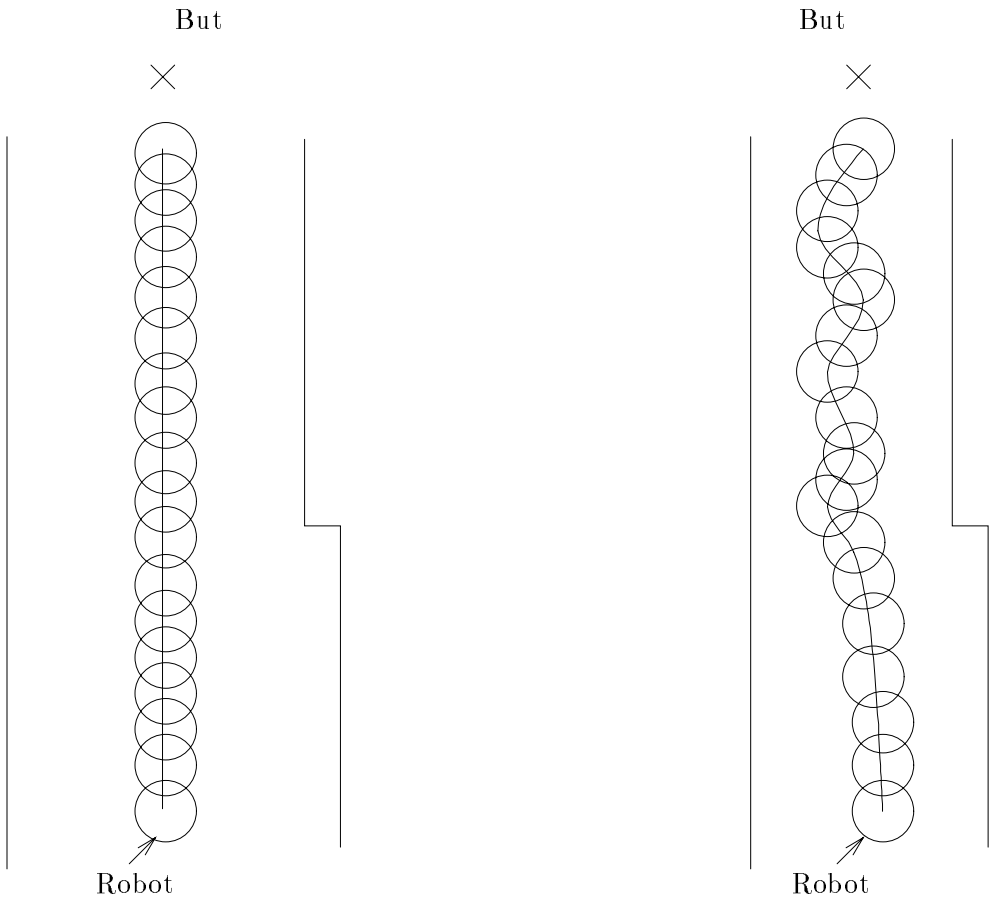


FIG. 4.20 - : *Oscillation en présence d'un couloir étroit*

– refus de franchir des passages étroits.

La méthode que nous avons proposée tente de fournir une solution dans certains cas à ces deux problèmes. Les oscillations ne sont en revanche pas traitées. Le point important est que ce type de problèmes est **inhérent** aux approches de type potentiel. Ceci signifie que quel que soit le réglage des différents paramètres disponibles dans ces approches les problèmes seront toujours présents. Le réglage de ces paramètres ne fait que déplacer la localisation de points singuliers (minima locaux) ou modifier la taille d'un couloir avant la présence d'oscillations par exemple. Il ne permet en revanche pas de faire disparaître ces problèmes.

La famille de fonctions que proposent les approches de type champs de potentiels ou champs de forces ne permet pas d'approcher de manière satisfaisante la fonction d'association perception action que l'on cherche. Nous avons donc décidé de nous intéresser à une famille de fonctions plus large en nous tournant vers un outil mathématique permettant d'approcher avec la précision voulue n'importe quelle fonction continue : la logique floue.

Chapitre 5

Le Contrôle Flou

5.1 Introduction et motivations

Comme nous l'avons vu au cours du chapitre 4Γles méthodes dites de *champs de potentiels* sont des méthodes cherchant à approcher une fonction inconnue en sélectionnant un membre d'une famille de fonctions. Les inconvénients classiques de telles approches sont :

- la présence de minima locaux. Le robot se trouve dans une situation où il ne bouge plus alors qu'il n'a pas atteint son but.

- les oscillations. Elles peuvent être de faible amplitudeΓlorsque le robot par exemple n'est pas en mesure de naviguer en ligne droite dans un couloirΓou de plus grande amplitude lorsque le robot " tourne en rond "Γc'est-à-direΓ lorsqu'il repasse constamment au même endroit.

Le fait que de tels phénomènes soient une caractéristique commune à toutes les fonctions d'une même famille nous indique que cette famille ne présente pas de fonctions proches de la solution que l'on cherche à générer. Nous avons ainsi décidé de nous tourner vers un outil moins restreintΓpermettant de générer un ensemble de fonctions plus vaste que les potentiels : le contrôle flou.

5.2 Le contrôle flou

Le contrôle flou est basé sur la logique floue qui elle-même est une extension de la théorie des ensembles flous. Nous allons revenir plus en détail sur ces trois notions au cours de cette section. Pour une description plus complèteΓle lecteur peut se reporter par exemple à [110] et [111] dont de nombreux éléments sont repris ici.

5.2.1 Historique

La précision des mathématiques telles que nous les connaissons repose en grande partie sur les travaux d'Aristote et de divers philosophes qui l'ont précédé. Dans leurs efforts pour concevoir une théorie de la logique et plus tard des mathématiques ils ont défini un certain nombre de lois dont la loi d'exclusion mutuelle stipulant que toute proposition doit être vrai ou fausse. Le premier a avoir remis en cause cette loi est Platon indiquant l'existence d'une troisième région située entre le vrai et le faux et où ces deux notions opposées se mélangent.

Au début du $20^{ième}$ siècle Lukasiewicz a décrit une logique tri-valuée dont la troisième valeur pourrait être traduite par *possible*. Il explora ensuite une logique à quatre puis cinq valeurs précisant que rien ne pouvait empêcher de dériver une logique avec une infinité de valeurs. Ce n'est qu'en 1965 que Lofti Zadeh a publié une description mathématique de la théorie des ensembles flous et par extension de la logique floue [189].

5.2.2 Les ensembles flous

La notion d'ensembles flous telle qu'elle a été présentée par Lofti Zadeh [189] est une extension de la notion classique d'ensembles. Son but est de permettre de capturer l'imprécision des données et des concepts nous entourant. Par exemple on n'apprend pas à une personne à conduire en lui disant : "commence à freiner 12 mètres avant l'intersection " mais plutôt en lui disant : "commence à freiner lorsque tu seras proche de l'intersection ".

Définition d'un ensemble flou

Soit S l'ensemble de discours et soit A un sous-ensemble de S. En théorie ensembliste classique la fonction d'appartenance associée à A est définie par :

$$I_A \ : \ S \mapsto \{0,1\}$$
$$I_A(x) = \begin{cases} 1 & \text{si x} \in \text{A} \\ 0 & \text{si x} \notin \text{A} \end{cases}$$

Définition 5.1 *la théorie des ensembles flous étend la définition de la fonction d'appartenance en la remplaçant par une fonction continue à valeurs dans l'intervalle* $[0,1]$ *:*

$$\mu_A \ : \ S \mapsto [0,1]$$
$$\mu_A(x) = 0 \Rightarrow x \notin A$$
$$\mu_A(x) = 1 \Rightarrow x \in A$$

Remarque : un sous-ensemble flou A de S est noté :

$$\int_S \mu_A(u)/u$$

indiquant que l'élément u de S appartient à l'ensemble A avec le degré $\mu_A(u)$

Exemple : soit S un ensemble de personnes et A le sous-ensemble flou de S définissant les personnes âgées. Soit Jeanne une personne de S. Supposons que Jeanne ait 75 ans. Selon la définition que l'on se donne des personnes âgéesΓ on peut avoir $\mu_A(Jeanne) = 0.8$Γce qui signifie que le degré d'appartenance de Jeanne a l'ensemble des personnes âgées est de 0.8. Ce nombre étant par ailleurs proche de 1Γcela signifie que l'on considère Jeanne comme étant âgée. On peut de même s'attendre à ce que le degré d'appartenance à A d'une personne de 15 ans soit de 0. Cet exemple permet de mettre en évidence le fait qu'un même concept (iciΓle concept de *âgée*) peut être représenté par une infinité de fonctions d'appartenance différentes (voir figure 5.1).

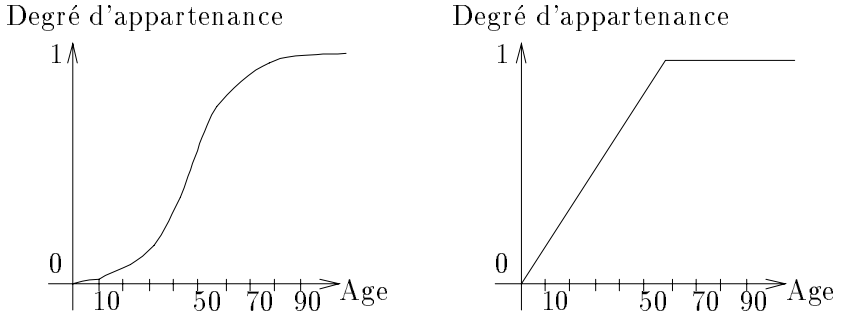


FIG. 5.1 - : *Deux représentations différentes d'un même concept*

Il est important de noter la différence existant entre le degré d'appartenance d'une valeur à un ensemble (donné par la fonction d'appartenance) et la probabilité qu'a cette valeur d'appartenir à cet ensemble. Les valeurs sont dans les deux cas comprises entre 0 et 1 mais leur sens est différent. L'exemple que nous allons maintenant détailler est repris de [24]. Considérons que l'univers du discours S est l'ensemble de tous les liquides et L est le sous-ensemble de S des liquides *potables*. Soit deux bouteilles A et B étiquetées respectivement à l'aide d'une probabilité d'appartenir à l'ensemble L et à l'aide d'un degré d'appartenance à cet ensemble (voir figure 5.2).

Si l'on souhaite boireΓla question posée est laquelle des deux bouteilles faut-il choisir? Le liquide de la bouteille A appartient à l'ensemble des liquides potables avec un degré de 0.91. Cela signifie que le liquide contenu peut être de l'eau salée par exemple mais certainement pas un produit dangereux tel que de l'acide chloridrique. Un degré de 0.91 indique que le liquide contenu n'est pas très éloigné de l'eau pure. Par contreΓle fait que la probabilité que le liquide contenu dans B soit potable est de 0.91 signifie qu'à la suite d'une série d'expériences similairesΓ le contenu de B sera potable dans 91% des cas et non potable dans 9% des cas. Le contenu de la bouteille peut tout aussi bien être de l'eau pure que de l'acide. Les

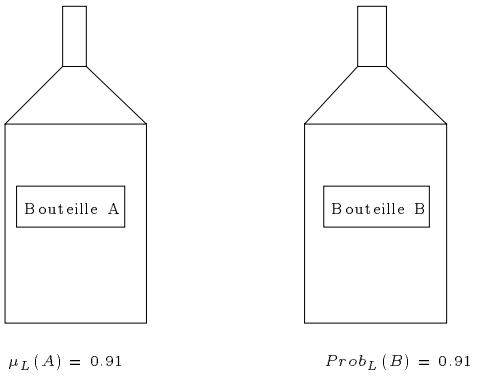


FIG. 5.2 - : *La seule information sur le contenu est donnée par un degré d'appartenance pour la bouteille de gauche et par une probabilité pour la bouteille de droite*

probabilités sont là pour juger à quel ensemble un élément appartient (potable non potable) alors que le degré d'appartenance permet de juger à quel point il appartient à un ensemble. Dans l'exemple précédent le choix raisonnable est la bouteille A. Plus de détails sur la différence entre ces deux notions peuvent être trouvés dans [103].

Définition 5.2 *Le support d'un ensemble flou F est l'ensemble non flou de tous les points s de S tels que $\mu_F(s) > 0$. En particulier, le ou les points s tels que $\mu_F(s) = 0.5$ sont appelés* points médians. *Si le support d'un ensemble flou est réduit à un seul point, on parle alors de* singleton *flou.*

La notion d'ensemble flou étant précisée nous allons maintenant définir les principales opérations permettant de les manipuler.

Opérations ensemblistes élémentaires

Soit A et B deux sous-ensembles flous de S définis respectivement par leur fonction d'appartenance $\mu_A(s)$ et $\mu_B(s)$.

Définition 5.3 (union) *La fonction d'appartenance $\mu_{A \cup B}$ de l'union $A \cup B$ est définie pour tout $s \in S$ par :*

$$\mu_{A \cup B}(s) = \max(\mu_A(s), \mu_B(s))$$

Définition 5.4 (intersection) *La fonction d'appartenance $\mu_{A \cap B}$ de l'intersection $A \cap B$ est définie pour tout $s \in S$ par :*

$$\mu_{A \cap B}(s) = \min(\mu_A(s), \mu_B(s))$$

Définition 5.5 (complément) *La fonction d'appartenance $\mu_{\neg A}(s)$ du complément de l'ensemble flou A est définie pour tout $s \in S$ par :*

$$\mu_{\neg A}(s) = 1 - \mu_A(s)$$

Définition 5.6 (produit cartésien) *Si $A_1, \ldots, A_n$ sont des sous-ensembles flous de respectivement $S_1, \ldots, S_n$, le produit cartésien de $A_1, \ldots, A_n$ est alors un sous-ensemble flou de l'espace produit $S_1 \times \ldots \times S_n$ dont la fonction d'appartenance est définie par :*

$$\mu_{A_1 \times \ldots \times A_n}(s_1, \ldots s_n) = \min(\mu_{A_1}(s_1), \ldots, \mu_{A_n}(s_n))$$

La fonction d'appartenance peut également être définie par :

$$\mu_{A_1 \times \ldots \times A_n}(s_1, \ldots s_n) = \mu_{A_1}(s_1).\mu_{A_2}(s_2) \ldots, \mu_{A_n}(s_n)$$

Définition 5.7 (relation floue) *Une relation floue n-aire est un sous-ensemble flou de $S_1 \times \ldots \times S_n$ et est exprimée par :*

$$R_{S_1 \times \ldots \times S_n} = \{((s_1, \ldots s_n), \mu_R(s_1, \ldots, s_n))/(s_1, \ldots s_n) \in S_1 \times \ldots \times S_n\}$$

Définition 5.8 (composition sup-étoile) *Soit R et S deux relations floues dans $U \times V$ et $V \times W$ respectivement. La composition de R et S est une relation floue notée R o S et définie par :*

$$R \; o \; S = \{[(u, w), sup_v(\mu_R(u, v) * \mu_S(v, w))], u \in U, v \in V, w \in W\}$$

où $*$ représente n'importe quel opérateur de la classe des normes triangulaires (l'opérateur minimum par exemple). Nous reviendrons plus en détail sur cette notion de norme triangulaire lors de la sous-section réservée à la logique floue.

Définition 5.9 (nombre flou) *Un nombre flou F dans un univers de discours continu S est un sous-ensemble flou normal et convexe de S :*

$$\begin{cases} \max_{s \in S} \mu_F(x) = 1 & \textit{normal} \\ \mu_F(\lambda s_1 + (1 - \lambda s_2)) \geq \min(\mu_F(s_1), \mu_F(s_2)) & \textit{convexe} \end{cases}$$

La propriété de convexité permet d'avoir une fonction d'appartenance croissante jusqu'à 1 puis décroissante. Cela permet de satisfaire la propriété intuitive que l'on souhaiterait avoir sur un nombre flou. Par exemple *approximativement 3* devrait avoir une fonction d'appartenance dont le seul maximum est en 3. A partir des nombres flousΓil est alors possible de définir les différents opérateurs arithmétiques tels que l'additionΓla soustractionΓla multiplication et la division. Pour plus de détailsΓle lecteur peut se reporter par exemple à [93].

Les différentes notions élémentaires associées aux ensembles flous étant maintenant préciséesΓnous allons nous intéresser à la principale application : la représentation de connaissances imprécises.

5.2.3 La logique floue : représentation de connaissances et inférence

La logique floue propose une base théorique pour le raisonnement approximatif grâce à un ensemble d'outils permettant la représentation ainsi que la manipulation de connaissances imprécises. Comme nous allons le préciserΓla connaissance est représentée sous forme de contraintes élastiques. L'inférence permet ensuite de propager ces contraintes.

Définition 5.10 (variables linguistiques) *Une variable linguistique est caractérisée par le quintuple* $(x, T(x), S, G, M)$. *x représente le nom de la variable (par exemple, la* température*). $T(x)$ représente l'ensemble des noms des valeurs linguistiques pouvant être prises par x. Chacune de ces valeurs linguistiques est un nombre flou sur S. Par exemple, toujours dans le cas de la température, on peut avoir* T(température) = {très froidΓfroidΓplus ou moins froidΓtièdeΓchaudΓ très chaudΓ...}. *La figure 5.3 illustre les différentes fonctions d'appartenance associées. G est une règle permettant d'associer un nom à une valeur de x. Fina-*

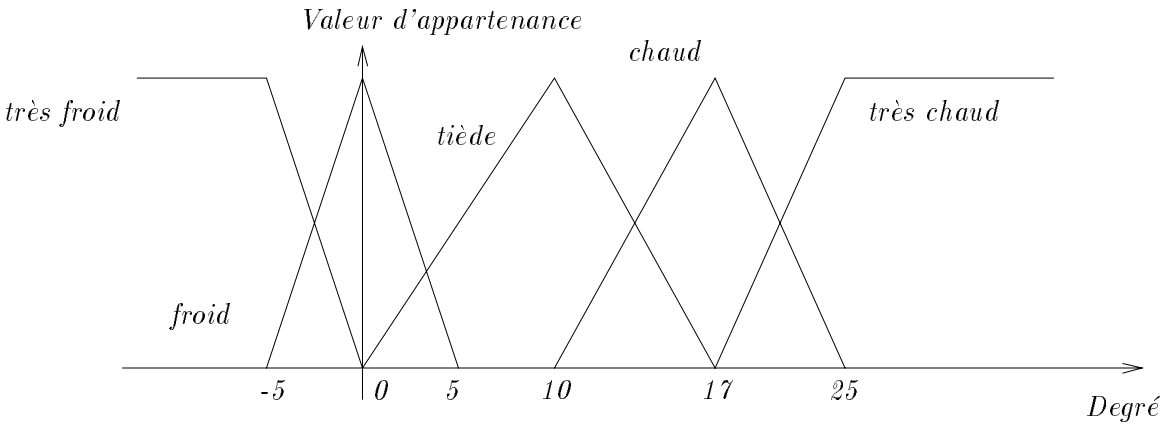


FIG. *5.3 - :* Fonctions d'appartenance associées aux différents nombres flous permettant de décrire la variable linguistique température.

lement, M représente une règle sémantique permettant d'associer à chaque valeur une signification. Par exemple, " froid " peut être interprété comme " les températures comprises entre -5 et 5 degrés " et " tiède " comme " les températures aux alentours de 10 degrés ".

Dans un système expert dit " classique "Γl'information élémentaire a pour forme : *the* ***attribute*** *of* ***object*** *is* ***value***. Par exempleΓ*la température de l'air est 30 degrés*. Plus généralementΓl'objet élémentaire peut s'écrire V *is* A où V

représente une variable (ici la température de l'air) et A sa valeur. Considérons maintenant le cas d'un système flou. Soit V une variable floue prenant ses valeurs dans l'univers du discours S. On associe à V la fonction $\pi_V : S \mapsto [0,1]$ qui à tout élément $s \in S$ associe la possibilité que V ait pour valeur s. Cette fonction est appelée *fonction de distribution de possibilité*. Reprenons la relation élémentaire V *is* A (où A est un ensemble flou). La théorie des possibilités nous indique que cette relation a pour conséquence de contraindre la fonction de distribution de possibilité de la variable V. On a la relation fondamentale suivante :

$$\forall s \in S \ \pi_V(s) = \mu_A(s) \tag{5.1}$$

où μ_A représente la fonction d'appartenance associée à A. Exemple : soit p la proposition *Jean est vieux*. La variable V correspond à *Age(jean)*. La valeur linguistique A correspond à *vieux*. La possibilité que jean soit âgé de 25 ans est donnée par la relation 5.1 : $\pi_V(25) = \mu_{vieux}(25) = 0$.

La contrainte élémentaire créée sur la fonction π_V par la relation 5.1 peut être enrichie et complétée grâce à l'utilisation d'opérateurs de quantification, de combinaison et de modification :

- les opérateurs de quantification. En logique classique, il n'existe que deux quantificateurs : l'universel et l'existentiel. En logique floue, il existe une grande variété de quantificateurs tels que *quelques uns, la plupart, plusieurs* ....

- les opérateurs de combinaison. Il s'agit de la conjonction et de la disjonction.

- les opérateurs de modification. En plus de l'opérateur de négation utilisé dans les systèmes classiques, il existe dans les systèmes flous des opérateurs tels que *très, plus, moins* ....

Une collection de propositions créées à partir de formes élémentaires $V is A$ augmentées éventuellement par les opérateurs précédents constitue une base de connaissances. L'évaluation du " sens " d'une nouvelle proposition par rapport à cette base est réalisée grâce à une approche de type "test-score semantic". Cette approche consiste à évaluer la contrainte apportée par la nouvelle proposition face à l'ensemble des contraintes réalisées par la base de connaissances existante. Nous ne développerons pas plus cet aspect pour nous intéresser plus particulièrement aux opérateurs de combinaison et à l'inférence dans les systèmes flous, permettant ainsi d'aboutir au contrôle flou. Le lecteur intéressé par la théorie des possibilités et sur la représentation de connaissances peut se reporter à [190, 55, 93, 188].

Avant de définir les opérateurs de conjonction et de disjonction nous allons introduire la notion de norme et de co-norme triangulaires.

Définition 5.11 (norme triangulaire) *Une norme triangulaire est une fonction* $* : [0,1] \times [0,1] \mapsto [0,1]$ *possédant les propriétés suivantes :*

Condition aux limites	$\begin{cases} 0 * 0 = 0 \\ 1 * a = a * 1 = a \end{cases}$
Monotonie	$a * b \leq c * d)$ *si* $a \leq c$ *et* $b \leq d$
Symétrie	$b * a = a * b$
Associativité	$a * (b * c) = (a * b) * c$

Quelques exemples classiques de normes triangulaires :

intersection	$x \wedge y = \min(x, y)$
produit	$x.y = xy$
produit borné	$x \odot y = \max(0, x + y - 1)$
produit drastique	$x \cap y = \begin{cases} x & \text{si } y = 1 \\ y & \text{si } x = 1 \\ 0 & \text{si } x, y < 1 \end{cases}$

Définition 5.12 (co-norme triangulaire) *Une co-norme triangulaire est une fonction* $\dot{+} : [0,1] \times [0,1] \mapsto [0,1]$ *possédant les propriétés suivantes (propriétés identiques à la norme triangulaire avec un changement de conditions aux limites) :*

Condition aux limites	$\begin{cases} 1\dot{+}1 = 1 \\ 0\dot{+}a = a\dot{+}0 = a \end{cases}$
Monotonie	$a\dot{+}b \leq c\dot{+}d$ *si* $a \leq c$ *et* $b \leq d$
Symétrie	$b\dot{+}a = a\dot{+}b$
Associativité	$a\dot{+}(b\dot{+}c) = ((a\dot{+}b)\dot{+}c$

Quelques exemples classiques de co-normes triangulaires :

union	$x \vee y = \max(x, y)$
somme	$x\hat{+}y = x + y$
somme bornée	$x \oplus y = \min(1, x + y)$
somme drastique	$x \Cap y = \begin{cases} x & \text{si } y = 0 \\ y & \text{si } x = 0 \\ 1 & \text{si } x, y > 0 \end{cases}$

Définition 5.13 (conjonction floue) *La conjonction floue de A et B est définie* $\forall s \in S$ *et* $\forall t \in T$ *par :*

$$\mu_{AconjB}(s,t) = \mu_A(s) * \mu_B(t)$$

où $*$ *est une norme triangulaire.*

Définition 5.14 (disjonction floue) *La disjonction floue de A et B est définie $\forall s \in S$ et $\forall t \in T$ par :*

$$\mu_{AdisjB}(s,t) = \mu_A(s) \dot{+} \mu_B(t)$$

où $\dot{+}$ est une co-norme triangulaire.

Définition 5.15 (négation floue) *La négation floue de A est définie $\forall s \in S$ par :*

$$\mu_{\neg A}(s) = 1 - \mu_A(s)$$

Comme nous l'avons signalé précédemment, les différents opérateurs que nous venons de définir permettent de mettre en place un réseau de contraintes décrivant la base de connaissances. L'étape suivante consiste maintenant à définir les mécanismes d'inférence permettant la propagation de ces contraintes. La forme générale d'une règle est la suivante : *if x is A then y is B*. La logique classique propose deux mécanismes principaux qui sont :

- Le *modus ponens*. Ce mécanisme permet, connaissant *x is A*, de déduire *y is B*.
- Le *modus tollens*. Ce mécanisme permet, connaissant $\neg B$, de déduire $\neg A$

L'application de contrôle à laquelle on s'intéresse est une application dont les sorties (commandes à appliquer au robot) sont fonctions des entrées (données perçues). Elle s'inscrit donc dans un mécanisme de type chaînage avant, propre au modus ponens et non chaînage arrière, caractéristique du modus tollens. Nous nous intéresserons à l'utilisation directe de l'implication $A \rightarrow B$.

Définition 5.16 (implication floue) *Nous allons définir les 4 grandes familles d'implications floues utilisées (citées par [111]). $*$ représente une norme triangulaire et $\dot{+}$ représente une co-norme triangulaire :*

implication " matérielle "	$A \rightarrow B = \neg A \dot{+} B$	(5.2)
calcul propositionel	$A \rightarrow = \neg A \dot{+} (A * B)$	(5.3)
calcul propositionel étendu	$A \rightarrow B = (\neg A \times \neg B) \dot{+} B$	(5.4)
modus ponens généralisé	$A \rightarrow B = sup\{c \in [0,1], A * c \leq B\}$	(5.5)

De manière générale, une règle d'inférence peut s'écrire sous la forme suivante :

Prémisse 1 : x is A'
Prémisse 2 : if x is A then y is B

Conséquence : y is B'

	x is A' (Prémisse 1)	y is B' (Conséquence)
Critère 1	x is A	y is B
Critère 2-1	x is very A	y is very B
Critère 2-2	x is very A	y is B
Critère 3-1	x is more or less A	y is more or less B
Critère 3-2	x is more or less A	y is B
Critère 4-1	x is not A	y is unknown
Critère 4-2	x is not A	y is not B

TAB. 5.1 - : *Critères intuitifs devant vérifier une implication floue dans le cadre du raisonnement approximatif*

avec $B' = A'oR$ où o est un opérateur de type *sup-étoile* (voir définition 5.8) et R un opérateur d'implication.

Si l'on remplace A' et B' par A et B respectivementΓon retrouve le schéma classique d'un modus ponens. FukamiΓMizumoto et Tanaka [64] ont proposé un ensemble de critères qu'une implication floue doit intuitivement réaliser dans le cadre du raisonnement approximatif. Ces critères sont également cités dans [111] et sont résumés dans le tableau 5.1. Le critère 4-2 provient d'une interprétation de la règle *if x is A then y is B* en *if x is A then y is B else y is not B*.

Les bases de la logique floue ayant été poséesΓnous allons maintenant décrire une des grandes familles d'applications reposant sur ces principes : le contrôle flou.

5.2.4 Le contrôle flou

Le contrôle flou est une application de la logique floue au contrôle de systèmes dynamiques pour lesquels on ne possède pas de modèle satisfaisant. Le schéma général d'un contrôleur est donné par la figure 5.4. Nous allons décrire chaque composant.

Le module de codage

Le module de codage[1] a pour rôle de transformer des données numériques provenant des capteurs en ensembles flous pouvant être manipulés par le contrôleur.

1. En anglais : fuzzyfication interface

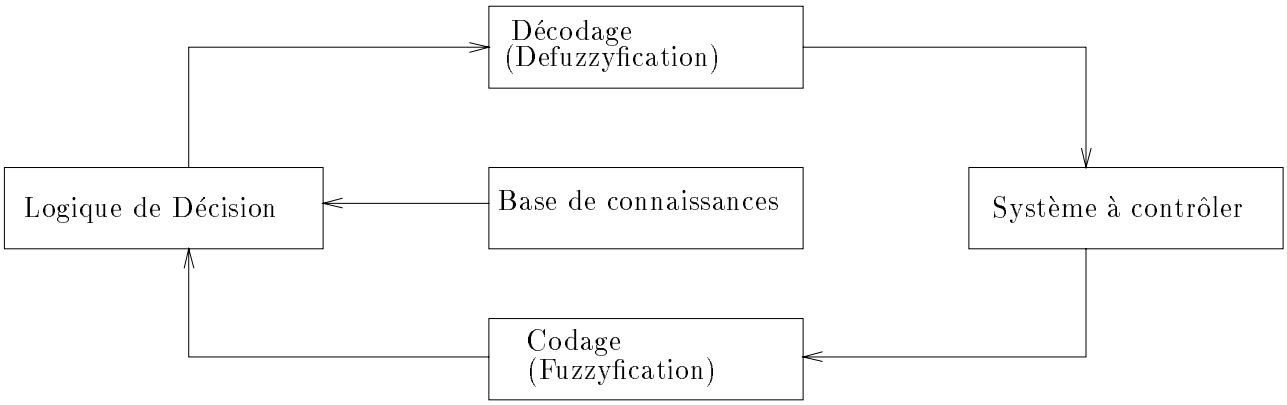


FIG. 5.4 - : *Architecture générale d'un contrôleur flou*

Il n'existe pas de manière unique de réaliser cette transformation. Cela dépend de la nature des données et du problème. Les deux plus courantes sont résumées figure 5.5. L'opération réalisée sur la figure de gauche consiste à remplacer une valeur numérique x_0 en un singleton flou dont le seul point possédant une fonction d'appartenance non nulle est également x_0. Ce type de codage est en général utilisé dans le cas de données d'entrée non bruitées. La figure de droite représente la codification d'une donnée numérique x_0 en un nombre flou (ici par l'intermédiaire d'un triangle). Cette représentation est généralement utilisée dans le cas de données perturbées par un bruit aléatoire. Le triangle est alors centré sur la valeur moyenne et la base correspond à deux fois l'écart type.

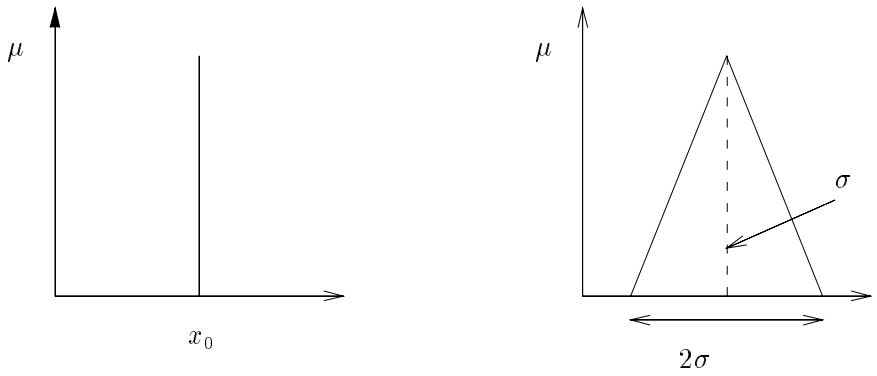


FIG. 5.5 - : *Exemple de deux codages de nombres réels en ensembles flous*

La logique de décision

Le module *logique de décision* représente le cœur du contrôleur flou. Il contient le mécanisme d'inférence permettantΓà partir des données provenant du module de codage et des règles contenues dans la base de connaissancesΓ de calculer les commandes devant être envoyées (via le module de décodage) au système à contrôler. Les règles floues utilisées sont de la forme générale :

if x_{11} is A_{11} and x_{12} is A_{12} and $\ldots x_{1n}$ is A_{1n} then y_1 is B_1 also
if x_{21} is A_{21} and x_{22} is A_{22} and $\ldots x_{2n}$ is A_{2n} then y_2 is B_2 also
.
.
.
if x_{m1} is A_{m1} and x_{m2} is A_{m2} and $\ldots x_{mn}$ is A_{mn} then y_m is B_m

Spécifier la logique de décision revient à choisir un opérateur d'inférenceΓun opérateur de composition d'inférence (opérateur o)Γun opérateur *and* ainsi qu'un opérateur *also*.

Les opérateurs d'inférence les plus couramment utilisés sont les suivants :

- Min (défini par Mamdani) :

$$R_c = A \times B = \int_{U \times V} (\mu_A(u) \wedge \mu_B(v))/(u,v)$$

- Produit (défini par Larsen) :

$$R_p = A \times B = \int_{U \times V} (\mu_A(u)\mu_B(v))/(u,v)$$

- Arithmétique (défini par Zadeh) :

$$R_a = (\neg A \times V) \oplus (U \times B) = \int_{U \times V} (1 \wedge (1 - \mu_A(u) + \mu_B(v)))/(u,v)$$

- Maxmin (défini par Zadeh) :

$$R_m = (A \times B) \cup (\neg A \times V) = \int_{U \times V} ((\mu_A(u) \wedge \mu_B(v)) \vee (1 - \mu_A(u)))/(u,v)$$

- Séquence standard :

$$R_s = A \times V \rightarrow U \times B = \int_{U \times V} (\mu_A(u) > \mu_B(v))/(u,v)$$

avec :

$$\mu_A(u) > \mu_B(v) = \begin{cases} 1 & \mu_A(u) \leq \mu_B(u) \\ 0 & \mu_A(u) > \mu_B(u) \end{cases}$$

– Implication floue booléenne :

$$R_b = (\neg A \times V) \cup (U \times B) = \int_{U \times V} ((1 - \mu_A(u)) \vee \mu_B(v))/(u, v)$$

– Implication floue de Goguen :

$$R_\Delta = A \times V \rightarrow U \times B = \int_{U \times V} (\mu_A >> \mu_B(v))/(u, v)$$

avec :

$$\mu_A(u) >> \mu_B(v) = \begin{cases} 1 & \mu_A(u) \leq \mu_B(v) \\ \frac{\mu_B(v)}{\mu_A(u)} & \mu_A(u) > \mu_B(v) \end{cases}$$

On peut constater que l'implication arithmétique de Zadeh est une implication matérielle avec un opérateur de somme borné (voir la définition des implications dans la sous-section précédente consacrée à la logique floue). De mêmeΓl'implication Maxmin de Zadeh est une implication propositionnelle avec les opérateurs d'intersection et d'unionΓla séquence standard est un modus ponens généralisé avec un produit bornéΓl'implication floue booléenne est également une implication booléenne avec l'opérateur union et que finalementΓl'implication floue de Goguen est un modus ponens généralisé avec un produit algébrique. Les implications Min et Produit de Mamdani et Larsen respectivement ne font par contre pas partie d'une des quatre grandes familles d'implications de la logique floue mais possèdent comme nous allons le voir maintenant des propriétés qui les rendent très intéressantes néanmoins. Lee [111] en particulier a étudié les conclusions rendues par ces diverses implications face aux prémisses présentes dans les critères de Fukami (voir table 5.1). Les hypothèses sont les suivantes :

– $B' = A'oR$ (B' conséquence de la règle R avec la prémisse A').

– R est un des opérateurs d'implication précédemment défini.

– o : opérateur sup-min (l'opérateur étoile utilisé est l'opérateur min).

– A' est ensemble flou de la forme :

$$\begin{array}{l} A = \int_U \mu_A(u)/u \\ \text{very} A = a^2 = \int_U {\mu_A}^2(u)/u \\ \text{more or less} A = a^{0.5} = \int_U {\mu_A}^{0.5}(u)/u \\ \neg A = \int_U (1 - \mu_A(u))/u \end{array}$$

Les fonctions d'appartenance résultat pour la conséquence y de la règle sont fournies dans la table 5.2.

La table 5.3 récapitule les résultats en indiquant pour chaque critère si oui ou non il est vérifié pour une inférence particulière. Il apparaît à la consultation

	A	very A	More or Less A	Not A
R_c	μ_B	μ_B	μ_B	$0.5 \wedge \mu_B$
R_p	μ_B	μ_B	μ_B	$\frac{\mu_B}{1+\mu_B}$
R_u	$\frac{1+\mu_B}{2}$	$\frac{3+2\mu_B-\sqrt{5+4\mu_B}}{2}$	$\frac{\sqrt{5+4\mu_B}-1}{2}$	1
R_m	$0.5 \vee \mu_B$	$\frac{3-\sqrt{5}}{2} \vee \mu_B$	$\frac{\sqrt{5}-1}{2} \vee \mu_B$	1
R_b	$0.5 \vee \mu_B$	$\frac{3-\sqrt{5}}{2} \vee \mu_B$	$\frac{\sqrt{5}-1}{2} \vee \mu_B$	1
R_s	μ_B	${\mu_B}^2$	$\sqrt{\mu_B}$	1
R_Δ	$\sqrt{\mu_B}$	${\mu_B}^{\frac{2}{3}}$	${\mu_B}^{\frac{1}{3}}$	1

TAB. 5.2 - : *Fonctions d'appartenance de la conséquence de la règle en fonction de l'inférence choisie et de l'entrée.*

de ce tableau que les deux opérateurs R_c et R_pΓbien que n'ayant pas une structure logique bien définie sont particulièrement bien adaptés pour le raisonnement approximatif. R_s peut également constituer un choix satisfaisantΓcontrairement à R_aΓ R_bΓ R_m et R_Δ ayant un comportement dans plusieurs cas allant à l'encontre de l'idée intuitive que l'on peut avoir de ce type de raisonnement. Nous

	R_c	R_p	R_a	R_m	R_s	R_δ	R_b
Critère 1	oui	oui	non	non	oui	non	non
Critère 2-1	non	non	non	non	oui	non	non
Critère 2-2	oui	oui	non	non	non	non	non
Critère 3-1	non	non	non	non	oui	non	non
Critère 3-2	oui	oui	non	non	non	non	non
Critère 4-1	non	non	oui	oui	oui	oui	oui
Critère 4-2	non	non	non	non	non	non	non

TAB. 5.3 - : *Satisfaction des critères par les différents opérateurs d'inférence*

ne considérerons par la suite que les opérateurs R_c et R_p.

La prochaine étape dans la définition de la logique de décision consiste à spécifier l'opérateur de combinaison d'inférence en choisissant une norme triangulaire pour l'opérateur étoile. Les 4 principales combinaisons trouvées dans la littérature sont [111] :

- l'opérateur *sup-min* défini par Zadeh en 1973.
- l'opérateur *sup-produit* défini par Kaufmann en 1975.
- l'opérateur *sup-produit-borné* défini par Mizumoto en 1981.

– l'opérateur *sup-produit-drastique* défini par Mizumoto en 1981.

Pour des raisons de coût de calculΓles opérateurs les plus fréquemment employés sont le sup-min et le sup-produit.

Le connectif *and* utilisé en partie gauche de règle se définit généralement comme une conjonction floue. Dans le cas de la règle *if A and B then C*Γl'antécédent sera considéré comme un ensemble flou sur l'univers de discours produit $U \times V$ dont la fonction d'appartenance sera $\mu_{A\times B}(u,v) = \min(\mu_A(u), \mu_B(v))$ ou $\mu_{A\times B}(u,v) = \mu_A(u)\mu_B(v)$. Quant au connectif *also* permettant la prise en compte de plusieurs règlesΓil sera choisi parmi les normes ou co-normes triangulaires. Il faut noter que les propriétés d'associativité et de commutativité de ces normes permettent d'assurer en particulier un résultat identique quel que soit l'ordre d'évaluation des règles. L'opérateur *also* le plus utilisé pour sa simplicité est l'opérateur union. On peut également utiliser l'opérateur *somme* et normaliser la fonction obtenue. On peut remarquer que l'opération de normalisation est inutile lorsque l'algorithme de décodage utilisé est indépendant de l'amplitude de la fonction présentée (ce qui est en général le cas).

Les différents opérateurs nécessaires à la détermination de la logique de décision ayant été explicitésΓnous allons maintenant nous intéresser au calcul de la valeur C' conséquence de l'ensemble des règles composant la base de connaissances. La valeur C' est donnée par la relation :

$$C' = \bigcup_{i=1}^{n} ({A'_1}^i, \ldots, {A'_m}^i) o R_i \tag{5.6}$$

Lorsque les opérateurs d'implication sont ceux de Mamdani et de LarsenΓil est possible d'interpréter la relation 5.6 graphiquement (voir figure 5.6 et figure 5.7). Dans le cadre de la figure 5.6 l'opérateur d'inférence est l'opérateur de MamdaniΓ l'opérateur de composition est *sup-min*Γle connectif *and* est réalisé par la fonction *min* et le connectif *also* par l'opérateur union. De manière à simplifierΓon considérera que le système est formé de 2 règles :

R1 : if x is A_1 and y is B_1 then z is C_1
R2 : if x is A_2 and y is B_2 then z is C_2

Les deux valeurs $\alpha_1 = \mu_{A_1}(x_0) \wedge \mu_{B_1}(y_0)$ et $\alpha_2 = \mu_{A_2}(x_0) \wedge \mu_{B_2}(y_0)$ peuvent être considérées comme les degrés d'activation des règles R_1 et R_2 respectivement. Ces degrés d'activation sont ensuite propagés en partie droite de manière à contraindre la fonction d'appartenance de la conséquence en la bornant. Les fonctions d'appartenance des deux règles sont finalement combinées par union afin d'obtenir le résultat final. Ceci permet de rappeler une propriété importante des systèmes flous. Contrairement aux systèmes à base de règles "classiques"Γtoutes les règles sont activées mais avec un degré correspondant à l'adéquation entre leur partie condition et la situation présente. De plusΓle mode de combinaison

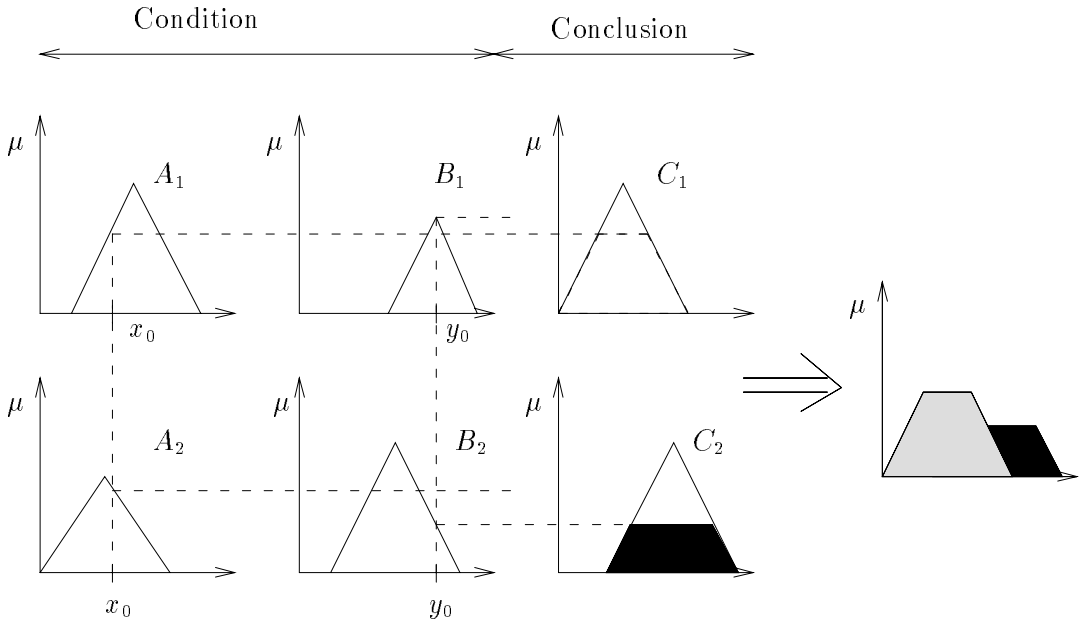


FIG. 5.6 - : *Interprétation graphique du calcul du résultat d'un contrôleur flou de type Mamdani*

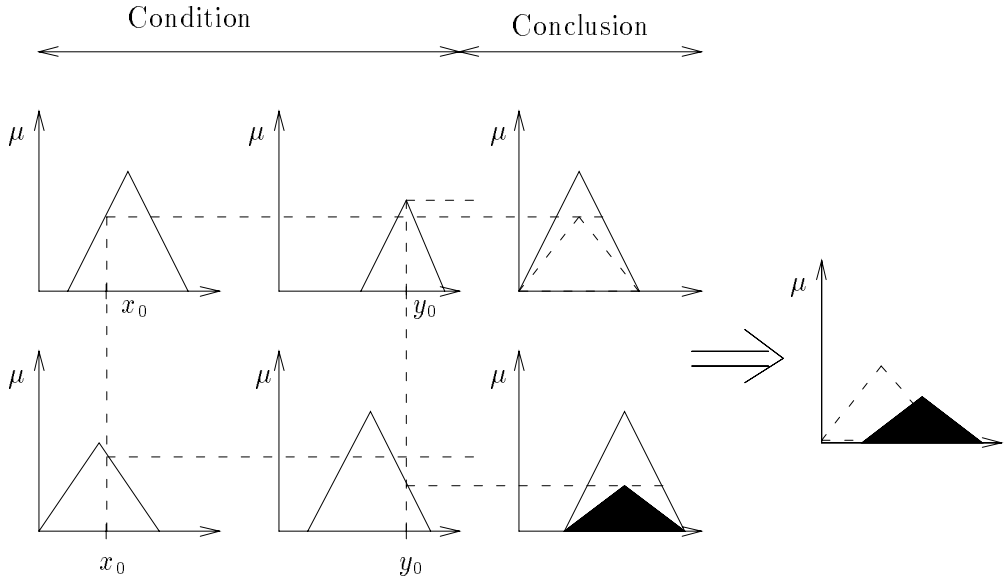


FIG. 5.7 - : *Interprétation graphique du calcul du résultat d'un contrôleur flou de type Larsen*

des résultats de chacune des règles interdit tout phénomène de conflit et donc de blocage nécessitant une arbitration. Le système rend toujours une réponse.

Le calcul de la fonction d'appartenance de la conséquence dans un contrôleur de type Larsen est quant à lui réalisé en multipliant la fonction d'appartenance de la conclusion par le degré d'activation de la règle (voir figure 5.7).

Il existe d'autres types de contrôleurs flous ne s'appuyant pas forcément aussi directement sur la logique floue et le raisonnement approximatif. Nous allons brièvement décrire deux des principaux : les contrôleurs de type *Tsukamoto* et les contrôleurs de type *Sugeno*.

Un contrôleur de type Tsukamoto [176] est constitué de règles *if* x_1 *is* A_1 *and* x_2 *is* A_2 *...and* x_n *is* A_n *then* y *is* B avec $A_1\Gamma \ldots A_n\Gamma B$ des ensembles flous dont les fonctions d'appartenance sont monotones. On peut remarquer que cette contrainte de monotonie peut être supprimée sur les termes A_i et obtenir un système proche d'un système de type Sugeno (voir ci-dessous). Dans un tel systèmeΓune règle R_i produit comme résultat un nombre réel y_i (voir figure 5.8) tel que $C_i(y_i) = \alpha_i$ (où α_i représente le degré d'activation de la règle). Les résultats de chaque règle sont ensuite combinés pour produire le résultat final :

$$y = \frac{\sum_{i=1}^{n} \alpha_i y_i}{\sum_{i=1}^{n} \alpha_i}$$

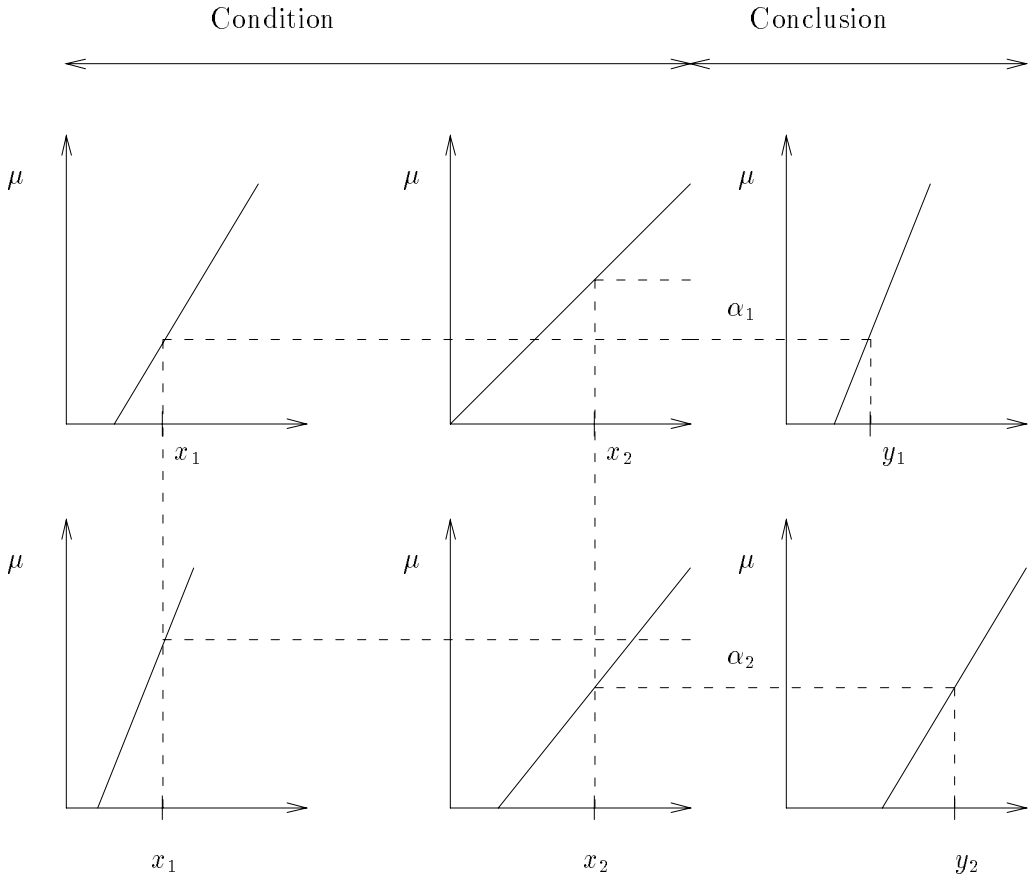
Un contrôleur de type Sugeno [168] est constitué de règles de la forme :

$$R_i : \text{if } {x_1}^i \text{ is } {A_1}^i \text{ and } \ldots \text{and } {x_n}^i \text{ is } {A_n}^i \text{ then } z_i = f_i({x_1}^i, \ldots, {x_n}^i)$$

où f_i est une fonction de l'espace d'entrée du système vers l'espace de commande. Chaque règle produit comme résultat un nombre réel $\alpha_i f_i({x_1}^i, \ldots, {x_n}^i)$ (α_i correspondant au degré d'activation de la règle i). Le résultat final du système est obtenu en combinant le résultat de chaque règle :

$$z = \frac{\sum_{i=1}^{m} \alpha_i f_i({x_1}^i, \ldots, {x_n}^i}{\sum_{i=1}^{m} \alpha_i}$$

Avant de nous intéresser au module de décodageΓnous terminerons cette présentation de la logique de décision en indiquant que elle peut être vue comme un système de mémoire associative [103]. Les règles réalisent une association d'une valeur floue de l'espace d'entrée à une valeur floue de l'espace de sortie. Le système d'inférence a alors pour tâche de réaliser une interpolation entre ces différentes valeurs (la " qualité " de l'interpolation dépendant bien évidemment des opérateurs utilisés). On peut par ailleurs montrer que sous certaines conditionsΓun système flou se comporte comme un approximateur universel [104]Γc'est-à-dire comme un système capable d'approcher avec le degré de précision voulu n'importe quelle fonction continue.



$$y = \frac{\alpha_1 y_1 + \alpha_2 y_2}{\alpha_1 + \alpha_2}$$

FIG. 5.8 - : *Principe général d'un contrôleur de type Tsukamoto*

Le module de décodage

La logique de décision étant maintenant décriteΓla dernière étape consiste à décoder les résultats fournis afin de les envoyer vers le système à contrôler (voir la figure 5.4). Le rôle de ce décodage est de transformer un ensemble flou résultat de l'évaluation des règles en un nombre réel représentant le mieux la distribution de possibilité induite par cet ensemble. Cette étape est bien entendue inutile dans le cas d'un système de type Tsukamoto ou Sugeno. On distingue trois grandes méthodes :

- La méthode du maximum. Cette méthode ne s'applique que si la fonction d'appartenance ne possède qu'un seul maximum (un nombre flou par exemple). Le décodage retourne alors la valeur z_0 telle que :

$$\mu_C(z_0) = \max_{u \in U} \mu_C(u)$$

- La méthode de la moyenne des maxima. Dans le cas d'un univers de commande discretΓla sortie du système est donnée par :

$$z_0 = \sum_{i=1}^{l} \frac{w_i}{l}$$

  $\{w_1, \ldots, w_l\}$ représentant l'ensemble des points où μ_C atteint son maximum.

- La méthode *Center of Area* :

$$z_0 = \frac{\int_a^b u \mu_z(u) du}{\int_a^b \mu_z(u) du}$$

  $[a, b]$ représentant l'intervalle de définition de la commande.

Ceci termine la présentation du fonctionnement des systèmes flous et plus particulièrement des contrôleurs flous. Nous allons maintenant nous intéresser à l'utilisation de tels systèmes dans le cadre de la navigation réactive.

5.3 Le système réactif flou

Nous allons décrire au cours de cette section le système flou de navigation réactive que nous avons développé [140Γ141]. Nous décrirons successivement les différents modules présentés figure 5.4. Nous nous intéresserons tout d'abord au choix des variables d'entrée et de sortieΓau module de codage et de décodageΓ puis à la logique de décision. Nous comparerons finalement les différents choix.

5.3.1 Choix des données d'entrée et de sortie

Le système flou de navigation a pour tâche de contrôler le déplacement linéaire et angulaire du robot en envoyant des commandes au contrôleur de véhicule. Comme nous le verrons par la suite, les règles sont déterminées de manière à reproduire les actes d'un opérateur humain téléopérant le système. Lorsque l'on pilote le robot à l'aide d'un joystick proportionnel, les grandeurs physiques commandées sont la vitesse linéaire et la vitesse angulaire. Nous avons donc choisi ces deux grandeurs comme sorties du contrôleur flou.

Comme nous l'avons indiqué au cours de la section 3.3.2, le système de navigation doit admettre en entrée les données relatives aux capteurs ultrasons ainsi que les données relatives au but. Dans le cas du but, les données retenues sont tout naturellement l'angle entre la direction du robot et la droite reliant le centre du robot au but ainsi que la distance séparant le robot du but (voir figure 5.9).

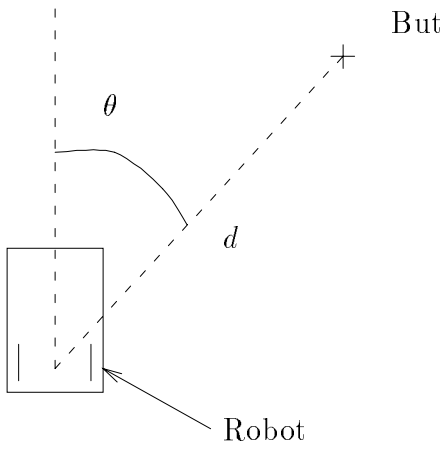


FIG. 5.9 - : *Données d'entrée du système flou relatives au but à atteindre : θ, d*

Le robot mobile ROBUTER utilisé au LIFIA comprend 24 capteurs ultrasons. La première idée intuitive est d'utiliser directement chaque capteur comme variable d'entrée. Ceci est envisageable lorsque le nombre de capteurs reste faible (voir par exemple [162] pour une utilisation directe des mesures fournies par 6 capteurs ultrasons). Mais lorsque le nombre de capteurs, comme dans notre cas, est élevée, cela pose plusieurs difficultés :

- le système est basé sur notre compréhension de la relation entre les mesures des capteurs et la réaction du robot et sur l'expression de cette compréhension sous forme de règles. 24 mesures capteurs représentent une quantité de données relativement difficile à appréhender.

- lié à l'argument précédent, la présence de 24 données capteurs ajoutées aux deux mesures relatives au but génère un espace d'entrée de dimension

élevée pouvant nécessiter un grand nombre de règles pour le partitionner. Supposons que chaque variable d'entrée soit décrite à l'aide de m données linguistiques. Ce nombre m fixe la granularité de la description d'un état du système à contrôler. Une partition complète de l'espace d'entrée nécessite alors m^{26} règles. $m = 3$ par exemple (ce qui correspond à une valeur raisonnable pour notre problème) nécessite la création de 2.5×10^{12} règles. Cette partition totale de l'espace correspond bien entendu au cas le plus défavorable et peut ne pas être nécessaire selon la nature du problème. En particulierΓsi les variables d'entrée peuvent être séparéesΓseules $26 \times m$ règles sont alors nécessaires. Dans le cadre de notre problèmeΓles valeurs des capteurs sont fortement couplées et le nombre de règles risque d'être élevé au détriment des performances.

– enfinΓet comme nous l'avons signalé au cours de la section 2.3.3Γles données provenant des capteurs ultrasons sont bruitées. Il est nécessaire d'opérer un filtrage spatial et (ou) temporel afin de supprimer ce bruit. L'utilisation directe dans le système flou des mesures capteurs signifie la nécessité de réaliser ce filtrage au niveau des règles ce qui risque d'augmenter leur nombre et d'alourdir le processus de création.

Il est donc nécessaire de mettre en place un module de pré-traitement dont le double rôle est de réduire la dimension de l'espace d'entrée et de filtrer les données capteurs.

Les approches existantes peuvent se différencier par le niveau d'abstraction des données manipulées. Ces données peuvent tout d'abord être de haut niveauΓnécessitant une phase de modélisation poussée de l'environnement. Saffiotti propose de maintenir un modèle de l'environnement dont il extrait pour les besoins de la navigation des objets de type mur ou intersection par exemple [149Γ151Γ150Γ152Γ153]. Kato [95] propose de modéliser les obstacles et de représenter l'espace libre entre eux sous forme de portes. Ces portes servent alors de données d'entrée au système. Zhang [193] propose d'utiliser une évaluation de la distance séparant le robot des objets situés devant luiΓà sa droite et à sa gauche. Il propose également de ne garder que les données capteurs concernant un obstacle imprévu (par comparaison entre une carte fournissant ce que les capteurs devraient voir et ce qu'ils voient réellement). Les données d'entrée manipulées peuvent également être de bas niveau. Ishikawa [84] s'intéresse par exemple aux données directement fournies par les capteurs (au nombre seulement de 5) mais également aux variations de ces donnéesΓpermettant d'introduire le temps dans son système.

Cette étude est motivée par la recherche d'un lien direct entre la perception et l'action et par la volonté de ne pas baser directement la commande sur l'utilisation d'un modèle pouvant s'avérer trop pauvre ou trop éloigné de la réalité si les capteurs ne sont pas suffisants ou si le temps de mise à jour est trop important. Nous préférerons donc les approches basées sur une manipulation plus

ou moins directe des données capteurs. Nous étudierons lors du chapitre 7 la possibilité de déterminer automatiquement un processus de pré-traitement. La méthode utilisée est basée sur une réduction de l'espace des données d'entrée en sous-espaces de dimension moindre à l'aide de transformations linéaires. Le problème posé par une telle approche est que l'interprétation de la transformation calculée n'est souvent pas possible. On n'est pas en mesure de pouvoir donner une interprétation à la projection de mesures réelles sur cet espace réduit. On ne peut donc pas écrire de règles associant ces grandeurs à des commandes. Nous avons préféré ne pas faire appel à ce processus automatique et coder nous-mêmes la réduction de dimension. La méthode choisie est inspirée de celle utilisée par Pin [132]. Les 24 capteurs ultrasons sont regroupés en 4 groupes, chaque groupe étant spécialisé dans une direction : *devant, derrière, à gauche et à droite*. Lors de la phase d'acquisition, seule la mesure la plus faible de chaque groupe est retenue (voir figure 5.10). Ce pré-traitement, outre la réduction de dimension de 24 à 4,

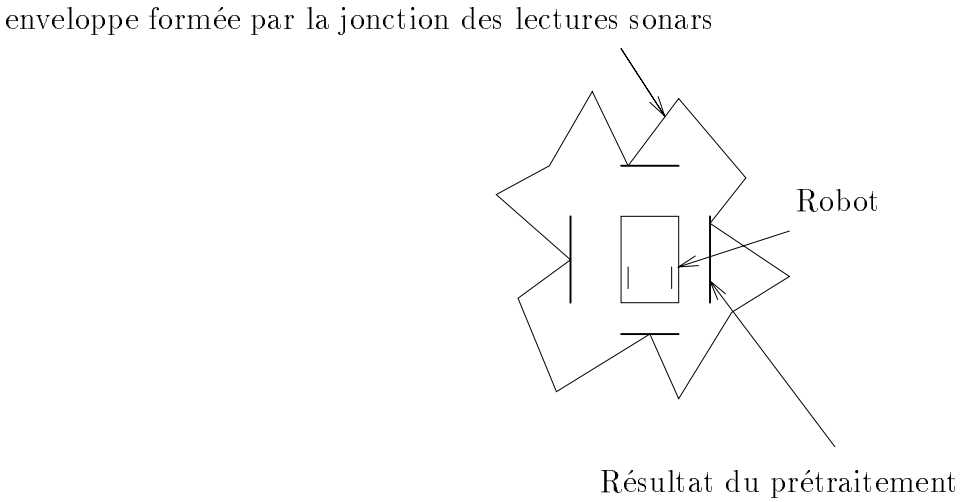


FIG. 5.10 - : *Pré-traitement des données capteurs avant utilisation par le contrôleur flou.*

a pour avantage de réaliser un premier filtrage simple des données. En effet, une des sources de bruit des capteurs ultrasons est la réflexion multiple qui a pour conséquence de générer des points " fantômes " le plus souvent éloignés. Le principal inconvénient de ce pré-traitement est son faible pouvoir de discrimination provoquant un phénomène "d'aliasing" perceptuel. Aliasing perceptuel signifie que deux situations différentes, nécessitant des réactions différentes ne peuvent pas être distinguées l'une de l'autre par le système de perception et sont donc confondues.

Les données d'entrée étant maintenant déterminées, il est nécessaire de les coder afin de pouvoir être manipulées par les règles floues du système.

5.3.2 Module de codage

Comme nous l'avons signalé au cours de la section 5.2.4Γil n'existe pas de manière unique ni systématique de réaliser le codage des données d'entrée. Nous nous sommes intéressés au cours de cette étude à trois d'entre elles :

- la première solution et la plus simple consiste à transformer une valeur numérique x_0 en un singleton flouΓc'est-à-dire en un ensemble flou dont la fonction d'appartenance vaut 1 au point x_0 et 0 partout ailleurs. Ceci revient à considérer que la mesure est non bruitéeΓce qui ne correspond bien évidemment pas à la réalité. Nous verrons lors des résultats expérimentaux que cette approche donne néanmoins de bons résultats.

- la deuxième solution consiste à modéliser lors du codage le bruit du capteur. De manière à simplifier les calculsΓnous représenterons l'erreur de mesure sous forme d'un cercle dont le rayon est fonction de la distance séparant le capteur du point mesuré. Ceci induit donc une erreur $\pm\sigma$ sur la distance x_0 estimée à l'obstacle (voir figure 5.11). Le module de codage transformera cette valeur numérique x_0 en un nombre flou dont la fonction d'appartenance est soit une exponentielleΓsoit un triangle centré sur x_0 et de base 2σ (la base d'une exponentielle est définie en considérant comme nulles les valeurs en dessous d'un certain seuil).

- la troisième solution consiste à considérer que la valeur floue qui sera manipulée par les règles ne correspond pas à la mesure de la distance séparant le robot d'un obstacle mais à la possibilité de présence d'un obstacle. La fonction d'appartenance est alors la suivante (voir figure 5.12) : pour une distance comprise entre 0 et $x_0 - \sigma$Γon considère la possibilité de présence d'un obstacle comme nulle (on suppose le bruit dû à des réflexions multiples filtré). La fonction d'appartenance croit ensuite jusqu'à x_0. Tout ce qui se situe au-delà de x_0 ne peut être vu par occlusion. Il est donc possible qu'il y ait un obstacle. La fonction d'appartenance vaut 1 $\forall x \geq x_0$.

5.3.3 Module de décodage

Le résultat de l'évaluation des règles de contrôle est deux ensembles flous représentant respectivement la distribution de possibilité de la vitesse linéaire et angulaire à commander. Le but du module de décodage est de transformer ces deux ensembles en deux nombres réels les représentant. Comme nous l'avons vu au cours de la sous-section 5.2.4Γil existe trois solutions principales. Ne pouvant garantir la présence d'un seul maximumΓla première approche est exclue. Parmi les deux restantesΓnous avons retenu l'approche *Center of Area* reflétant plus la forme de la fonction d'appartenance que la Moyenne des Maxima (voir figure 5.13).

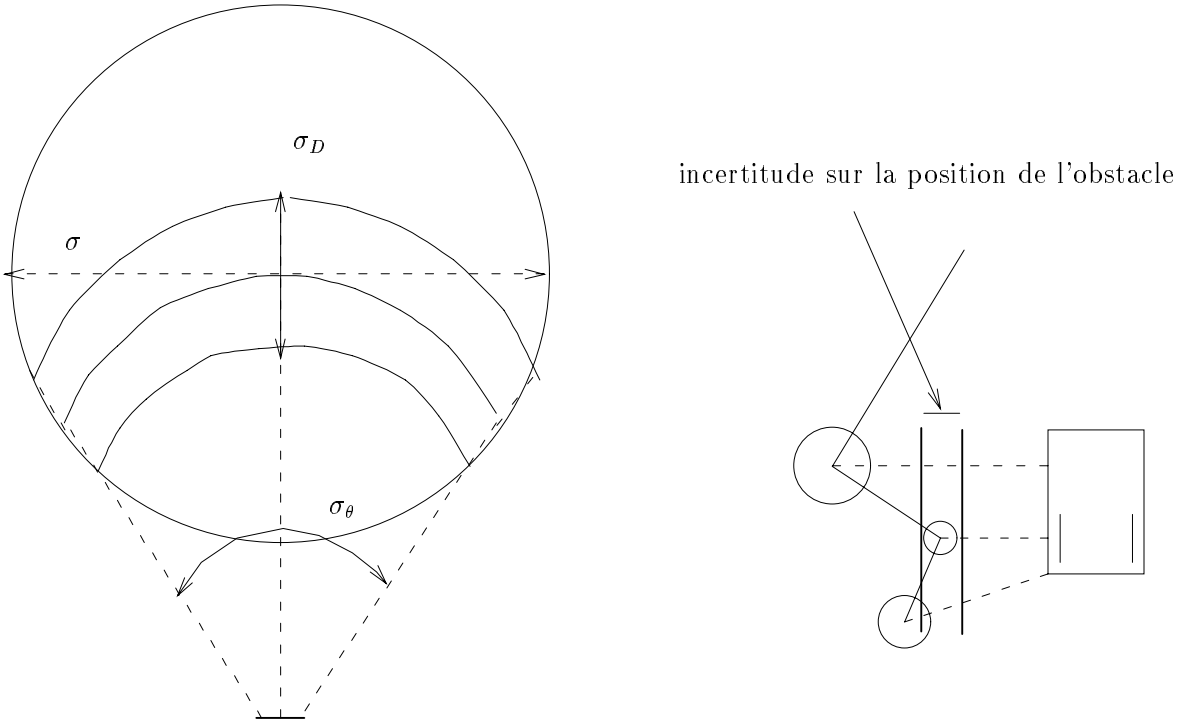


FIG. 5.11 - : *Incertitude sur la distance à l'obstacle*

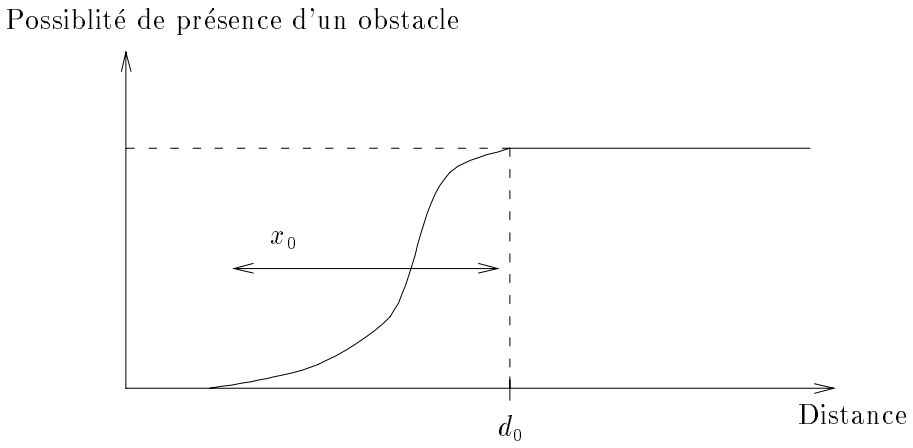


FIG. 5.12 - : *Représentation de la possibilité de présence d'un obstacle à une distance d (la valeur retournée par les capteurs étant x_0)*

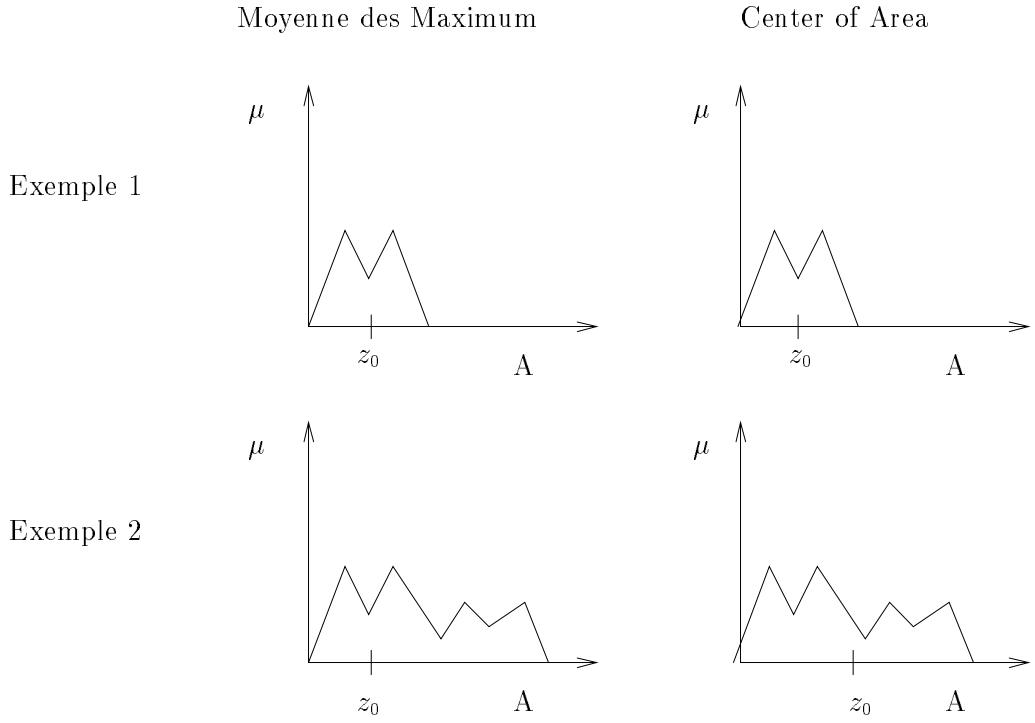


FIG. 5.13 - : *L'approche Center of Area reflète plus la forme de la distribution de possibilité*

5.3.4 Détermination de la logique de décision

Les différents paramètres permettant de sélectionner une logique de décision particulière sont :

- l'opérateur d'implication.
- l'opérateur de composition *sup-étoile*.
- le connectif *and*.
- le connectif *also*.

Nous nous sommes intéressés aux structures suivantes :

- l'implication de *Mamdani* et de *Larsen*.
- l'opérateur de composition *sup-min*.
- l'opérateur *min* pour le connectif *and*.
- l'opérateur *somme normalisée* et *max* pour le connectif *also*.

Ces structures ne sont pas équivalentes et présentent un certain nombre d'avantages et d'inconvénients. En particulier :

- L'opérateur d'implication de Mamdani propage le degré d'activation d'une règle à sa partie droite grâce à l'opérateur *min* (voir figure 5.6). L'opérateur d'implication de Larsen propage le degré d'activation par produit (voir figure 5.7). Il conserve ainsi la forme de la fonction d'appartenance de la partie conclusion et évite une perte d'informations lors de l'activation partielle d'une règle (degré d'activation différent de 1).

- $\max(a, b, \ldots, b) = \max(a, b)$. L'utilisation de l'opérateur max pour le connectif *also* implique que le résultat final du système n'est pas influencé par le nombre de règles arrivant à une même conclusion. Trois règles différentes arrivant à la conclusion que le robot doit ralentir ont le même effet que si une seule règle y arrive. Ce phénomène n'existe pas dans le cas de l'utilisation de la *somme normalisée* (voir figure 5.14).

Pour les deux raisons évoquées précédemment nous avons choisi de retenir une implication de type Larsen et un connectif *also* de type somme normalisée.

La dernière étape (et la plus délicate) consiste maintenant à déterminer les règles.

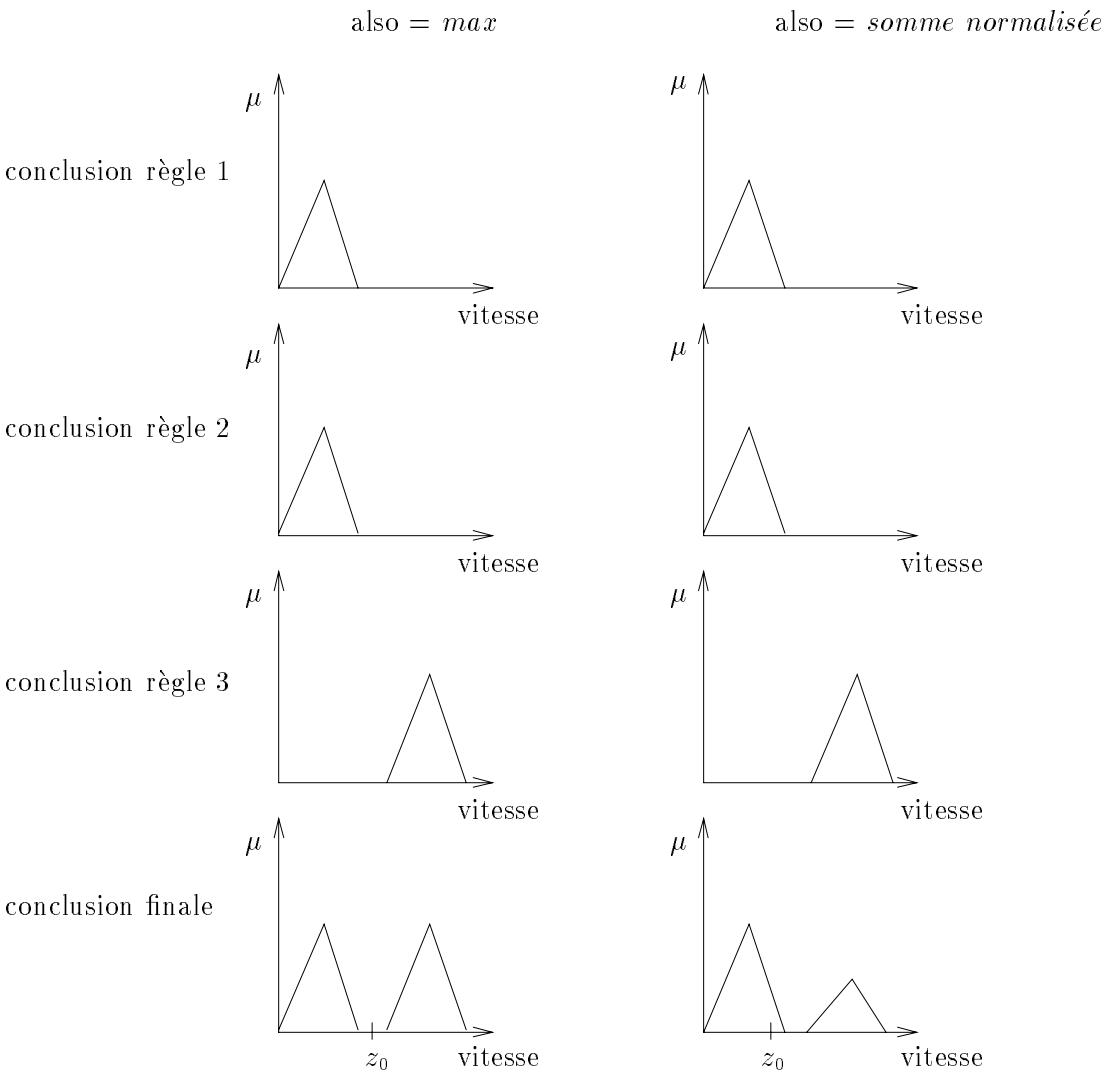


FIG. 5.14 - : *L'utilisation de la somme normalisée pour la combinaison des règles permet de prendre en compte le nombre de règles arrivant à une conclusion similaire*

5.3.5 Détermination des règles

Il existe quatre méthodes principales permettant de dériver des règles de contrôle [168] :

1. en interrogeant un expert. Le but du contrôleur flou est alors non pas de modéliser le système à contrôler mais de modéliser le comportement de la personne en train de le contrôler. On constate que nombreuses décisions prisent lors du contrôle d'un système sont de nature linguistique et non numérique et peuvent donc s'exprimer dans le formalisme proposé par la logique floue.

2. par observation des commandes fournies par un opérateur. Il existe des systèmes qu'un opérateur humain est en mesure de contrôler sans pouvoir expliciter les règles utilisées. Il s'agit alors de construire ces règles à partir d'un ensemble d'exemples formés de couples *(observation, action)*. Dans le cas de la navigation réactive nous verrons au cours du chapitre 7 que dans le cas neuronal la génération d'exemples représentatifs permettant au contrôleur de généraliser correctement lors de la présentation d'un cas nouveau est un problème. Ce problème risque d'être similaire si le système neuronal est remplacé par un système flou.

3. en construisant un modèle flou du système. Cela consiste tout d'abord à créer une description linguistique du processus dynamique que l'on souhaite contrôler. Cette description peut être considérée comme un modèle flou de ce processus. Connaissant ce modèle on cherche ensuite à extraire des règles permettant de le contrôler. Cette approche est en quelque sorte la réplique floue des approches automatiques " classiques ". Ce type d'approche semble plus adapté aux problèmes où le contrôleur a pour but de faire suivre au système une trajectoire de référence.

4. par apprentissage. Les approches basées sur l'apprentissage ont pour objectif de créer automatiquement une base de règles ou de modifier une base déjà existante. L'apprentissage peut être basé sur un ensemble de couples de points *(entrée,sortie)*. On parle alors d'apprentissage supervisé. Il peut être également basé sur une modification en ligne du contrôleur afin d'améliorer les performances du système contrôlé. On parle alors d'apprentissage renforcé. Nous reviendrons plus en détail sur ce type d'approche au cours du chapitre 8.

Nous utiliserons donc la première approche afin de déterminer les règles utilisées dans notre système de navigation réactive.

Les règles étant formulées par un opérateur humain selon sa compréhension du pilotage du véhicule il est possible que l'espace d'entrée ne soit pas entièrement couvert et que certains états du robot n'activent aucune règle

($\forall\ i\ \in\ [1\ \ldots\ n], \alpha_i\ =\ 0$). Dans une telle situation, le système flou répond que la situation est en dehors de sa compétence et qu'il ne connaît pas la commande à appliquer. Cette propriété très intéressante et ne se retrouve pas chez certains approximateurs universels tels que les réseaux de neurones à couche par exemple, fournissant une réponse quel que soit l'entrée.

De manière à limiter le risque d'obtenir des commandes non souhaitées, nous avons étendu l'absence de réponse à d'autres situations que la stricte nullité du degré d'activation de toutes les règles. Le critère ($\forall i \in [1 \ldots n] \alpha_i \leq \epsilon$) n'est pas correct car plusieurs règles concluant " faiblement " le même résultat renforcent ce résultat qui peut donc être considéré. Il faut plutôt s'intéresser à la valeur maximum prise par la distribution de possibilité de la conclusion. Si celle-ci est trop faible (inférieure à un seuil fixé), cela signifie qu'aucune réponse n'est vraiment possible et que le système est donc en dehors de sa zone de compétence. Quelle commande faut-il alors générer ? Deux solutions sont possibles :

1. on ne génère aucune commande et on laisse le système évoluer selon sa propre dynamique. Dans le cas d'un robot mobile, cela signifie le laisser continuer selon sa vitesse courante. Le robot continue à se déplacer et peut éventuellement sortir de la zone d'incompétence du système flou afin que celui-ci puisse éventuellement reprendre le contrôle. Cela veut dire par contre que pour des raisons de sécurité, le contrôleur flou ne pouvant pas remplir sa tâche pendant un ou plusieurs cycles, un processus de type réflexe, dont la fréquence d'activation doit être bien supérieure à celle du contrôleur, doit surveiller la présence d'obstacles imprévus et arrêter éventuellement le robot.

2. la deuxième solution consiste à arrêter immédiatement le robot. Cette solution est plus sûre et est préférable lors de la phase de mise au point du système. L'arrêt du robot permet d'analyser la situation dans laquelle il se trouve et de décider s'il faut modifier (en généralisant par exemple) une règle existante ou créer une nouvelle règle pour traiter cette situation particulière.

Piloter un robot mobile vers un but tout en évitant un ensemble d'obstacles imprévus est un problème complexe. Il semble difficile de l'aborder de manière globale et de fournir une solution (base de règles) générale adaptée à l'ensemble des situations. On est par contre en mesure d'expliciter la réaction que doit avoir le véhicule face à certaines situations particulières. Le comportement global de navigation réactive vers un but peut être décomposé en un ensemble de comportements élémentaires (ensemble de règles) possédant chacun une compétence particulière. Le double problème est :

1. comment détecter ces situations particulières ?

2. quel lien existe-t-il entre ces comportements élémentaires ?

Avant de revenir sur ces deux points particuliersΓnous allons tout d'abord nous intéresser aux différents comportements que nous avons mis en place. La distance séparant le robot d'un obstacle est décrite à l'aide de deux données linguistiques : *(près, loin)*. En reprenant les trois variables d'entrée attachées aux capteurs ultrasons *(obstacle gauche, obstacle droit, obstacle devant)*Γon obtient 8 situations différentes répertoriées figure 5.15.

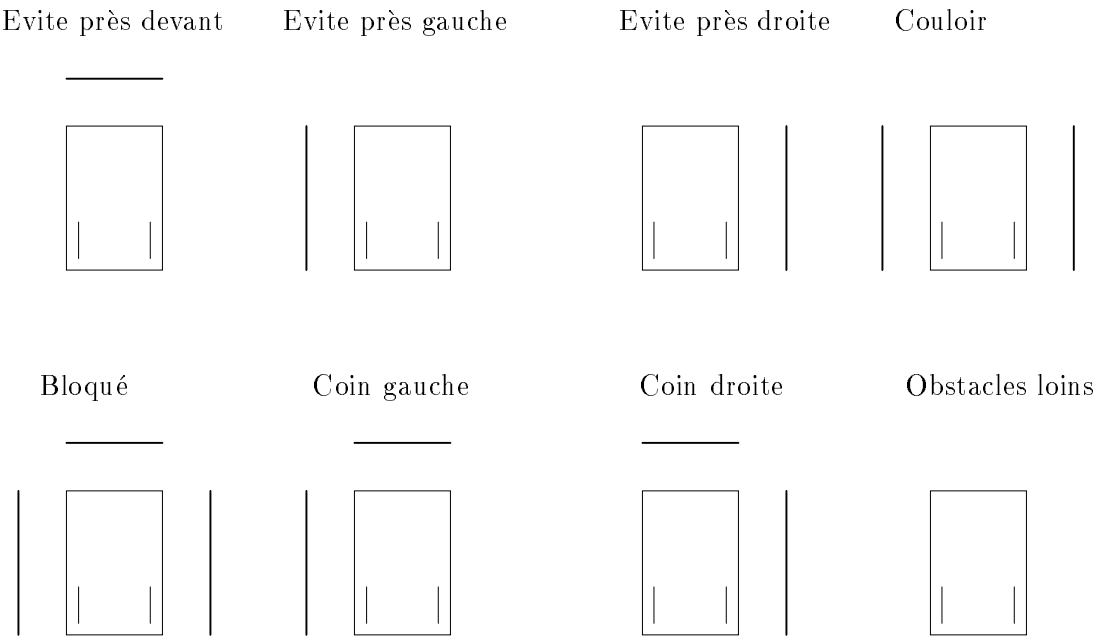


FIG. 5.15 - : *Représentation des 8 situations perceptives différentes associées aux capteurs ultrasons. Sur les quatre données perceptives, seules celles correspondant à* près *sont représentées*

Les réactions associées à ces situations sont les suivantes :

Evite près devant : on arrête le mouvement de translation du robot et on demande une rotation lente à droite.

Evite près gauche : on ne fournit pas de consigne pour la vitesse de translation et on demande une rotation lente vers la droite de manière à s'éloigner de l'obstacle.

Evite près droite : de mêmeΓon ne fournit pas de consigne pour la vitesse de translation et on demande une rotation lente sur la gauche.

Couloir étroit : pas de consigne pour la vitesse linéaire. On demande une vitesse angulaire nulle pour rester dans l'axe du couloir.

blocage : on arrête le mouvement de translation du robot et on demande une rotation lente vers la droite (de manière soit à réaliser un demi-tourΓsoit à apercevoir une issue par changement de point de vue résultant de la rotation).

Coin gauche : on arrête le mouvement de translation du robot et on demande une rotation lente à droite.

Coin droit : de mêmeΓon arrête le mouvement de translation du robot et on demande une rotation lente à gauche.

On peut remarquer que dans la situation *Evite près devant*Γl'arrêt du mouvement linéaire du robot pourrait être suffisant. La demande d'une rotation à droite permet de fixer une direction d'échappement par défaut et contribuer ainsi à éviter le blocage classique du robot face à un murΓle but à atteindre étant aligné avec le robot mais situé de l'autre côté (voir figure 5.16).

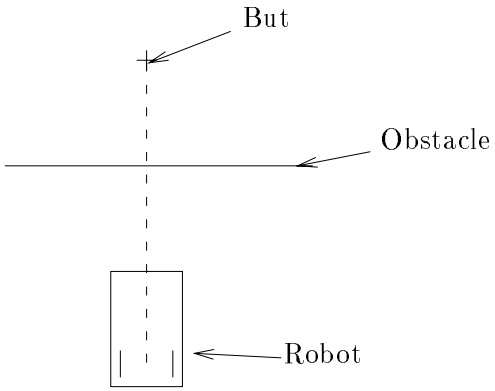


FIG. 5.16 - : *Situation classique de blocage*

A ces huit comportements élémentairesΓon ajoute un comportement permettant d'atteindre le but proposant une rotation sur la droite ou sur la gauche selon la position du point à atteindre et réglant la vitesse linéaire en fonction de la distance à parcourir.

La traduction de ces divers comportements sous forme de règles est immédiate. Il s'agit par contre de fixer les fonctions d'appartenance des données linguistiques en parties condition et conclusion des règles. La figure 5.17 résume les fonctions retenues pour la partie condition. Ce sont des gaussiennes. Lorsque l'on ne connaît pas de manière certaine l'intervalle $[x_0 - \delta, x_0 + \delta]$ dans lequel les données doivent être comprisesΓet que donc plusieurs intervalles peuvent être formulés selon l'expert interrogéΓla fonction d'appartenance représentant le mieux la fusion de l'avis de tous les experts est une gaussienne [105]. Le support $(\{x/\mu(x) \neq 0\})$

d'une fonction d'appartenance en partie condition détermine la zone d'influence d'une règle. Plus le support est grand et plus la règle est générale. Nous avons choisi le support de ces fonctions de manière à avoir un bon recouvrement de l'ensemble des valeurs pouvant être prises par les données d'entrée (les limites du support d'une donnée linguistique correspondent au centre des données linguistiques adjacentes).

Pour la partie conclusion, nous avons choisi comme fonction d'appartenance des gaussiennes centrées sur la vitesse correspondante ($0.1ms^{-1}$ par exemple pour une vitesse linéaire " normale "). Contrairement à la partie condition, le support des données linguistiques de la conclusion n'a pas une interprétation intuitive aisée permettant de le déterminer facilement. Nous verrons néanmoins plus tard grâce à la relation 5.7 que cela est possible dans certains cas.

Les différents comportements étant mis en place, nous allons revenir aux deux problèmes cités précédemment : comment les détecter et quel lien doit-il exister entre eux ? Beom [20] propose une détection de la situation grâce à un réseau de neurones à couches. La mesure retournée par chaque capteur ultrason est discrétisée en trois valeurs différentes. L'ensemble des combinaisons possibles de ces valeurs entre les différents capteurs fournit la totalité des entrées provenant du module de perception. Beom associe à chacune de ces combinaisons la situation correspondante et obtient ainsi la base d'apprentissage de son réseau. Il faut remarquer que dans cet exemple, la totalité des situations et de leur réponse associée est connue par avance. Le réseau n'a donc pour rôle que de réaliser une table de correspondance avec un faible coût mémoire. Nous reviendrons plus en détails sur les réseaux de neurones et leurs propriétés au cours du chapitre 7.

Pour chaque situation perceptive, le réseau fournit le comportement correspondant. La réponse est unique et seul un comportement à la fois sera actif et aura le contrôle du véhicule à chaque instant. Ceci a pour conséquence de créer une surface de contrôle pouvant être discontinue. De plus, cette association unique de chaque entrée perceptive à une situation répertoriée ne correspond pas à la réalité. Elle peut par contre être décrite à l'aide de concepts flous. A chaque instant, le robot se trouve dans toutes les situations mais avec des degrés différents. Dans le cas de la figure 5.18, le robot est à la fois dans la situation *obstacle gauche* et *coin gauche* avec un degré supérieur pour *obstacle gauche*.

La détection de la situation est réalisée par un ensemble de règles floues de la forme :

```
si obstacle_gauche est pres et obstacle_droit est pres et
non obstacle_devant est pres alors couloir est vrai
```

vrai étant défini par une fonction d'appartenance constante $\forall x \ \mu_{vrai}(x) = 1$. Cette règle produit une variable floue *couloir* dont la fonction d'appartenance est une fonction constante ayant pour valeur le degré d'appartenance de la situation courante à la situation *couloir*. On peut remarquer que le résultat est identique que l'on utilise un opérateur d'inférence de type Larsen ou Mamdani. Les variables

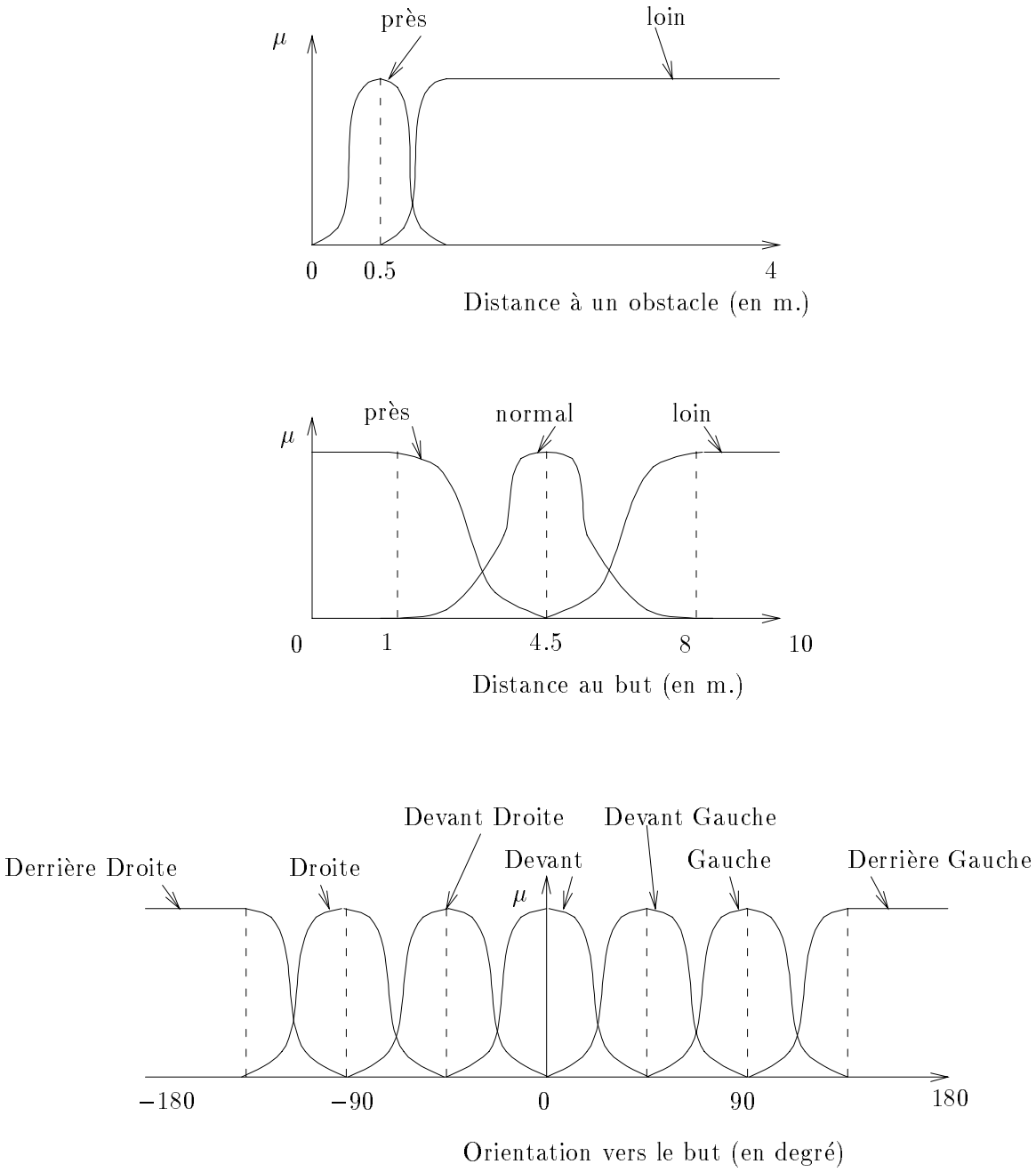


FIG. 5.17 - : *Définition des fonctions d'appartenance des données linguistiques présentes dans la partie condition des règles*

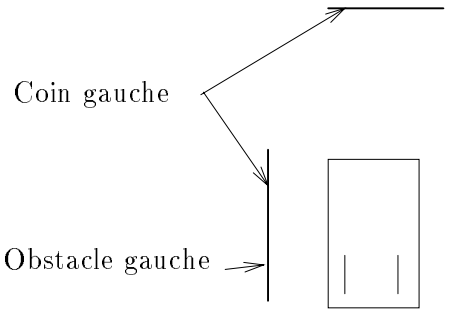


FIG. 5.18 - : *Le robot se trouve à la fois dans la situation* obstacle gauche *et* coin gauche

associées à chaque situation décrite figure 5.15 sont ensuite utilisées en partie gauche des règles décrivant le comportement correspondant. Les règles associées à chaque comportement sont alors groupées en une seule base de règles.

Pour chaque situation iΓil existe une zone de l'espace d'entrée pour laquelle $\mu_{situation_i} = 1$ et $\forall j \neq i \ \mu_{situation_j} = 0$. Il existe donc des zones dans lesquelles un seul comportement est actif. L'inférence floue permet de réaliser une interpolation entre ces différentes zones et de créer une surface de commande continue reliant les commandes associées à chaque comportement défini par le concepteur du système. Ceci permet donc d'avoir un transition " douce " entre les différents comportements.

Cette approche de fusion de comportements que nous avons suivie est très similaire à celle proposée par Saffiotti [149Γ151Γ150Γ152Γ153]. Saffiotti définit chaque comportement comme une structure de contrôle $S_i =< A_i, B_i, C_i >$Γoù A_i est un objet réel ou virtuel du modèle de l'environnement (un objet réel du modèle est un objet observé alors qu'un objet virtuel est un objet placé par le systèmeΓ comme par exemple un but à atteindre ou un mur imaginaire à suivre)ΓB_i est l'ensemble des règles précisant le comportement à tenir face à l'objet A_i et enfin C_i représente un prédicat flou indiquant dans quel contexte le comportement doit être activé. Par exempleΠla structure de contrôle $S_1 =< CP_1, Goto_1, near(CP_1) >$ est associée au comportement permettant d'atteindre un but. CP_1 est un point de contrôle de coordonnées (x, y)Γ$goto_1$ est un ensemble de règles permettant de rejoindre ce pointΓet $near(CP_1)$ indique que ces règles ne doivent être activées que dans une zone proche du but.

La fusion des comportements permet de générer des commandes dans n'importe quelle situation à partir de commandes imposées par le concepteur et applicables dans des situations voisines. L'interpolation réalisée dépend des opérateurs mis en place dans la logique de décision. Pin [132Γ133] utilise un outil plus souple permettant d'influencer la forme de la courbe. Il s'agit de coefficients associés à chaque comportement et multipliés à la fonction d'appartenance conclusion de ce comportement. Ces coefficients permettent d'établir un ordre de priorité. Par

exemple, éviter un obstacle frontal est plus important que de se détourner d'un obstacle latéral qui ne menace pas directement le véhicule. De même, atteindre le but est moins important qu'éviter les obstacles. Le point intéressant est que cette priorité n'est pas rigide. Soit A et B deux comportements et $p_a > p_b$ leur priorité associée. Pour un degré d'activation égal, le comportement A sera prédominant sur le comportement B. Si en revanche le degré d'activation de A devient inférieur à celui de B, B peut devenir prédominant sur A. Nous verrons au cours de la sous-section 5.4.1 les limitations d'une telle approche.

Remarque : Soit C un contrôleur flou de type Larsen. Nous supposerons que le connectif *also* utilisé est la somme normalisée et que les fonctions d'appartenance des données linguistiques sont toutes des exponentielles (paramètres (σ, μ)). La valeur retournée par C peut s'écrire :

$$z = \frac{\sum_{i=1}^{n} w_i \alpha_i \mu_i \sigma_i}{\sum_{i=1}^{n} w_i \alpha_i \sigma_i} \tag{5.7}$$

où n représente le nombre de règles, α_i le degré d'activation de la règle i et w_i son poids associé. μ_i et σ_i sont les deux paramètres de la donnée linguistique de la partie conclusion. Les détails des calculs peuvent être trouvés section 8.7. Si on pose $\sigma'_i = w_i \sigma_i$, on obtient un système flou équivalent dont tous les poids sont à 1. Le support de la conclusion est donc relié dans ce cas à l'importance relative de la règle par rapport aux autres.

Les règles étant maintenant spécifiées, nous allons nous intéresser à leur mise au point et aux différents résultats expérimentaux obtenus. La totalité des règles est fournie dans l'annexe B.

5.4 Résultats expérimentaux

Nous allons présenter au cours de cette section les résultats expérimentaux obtenus avec l'approche réalisée.

5.4.1 Edition de la surface de contrôle

Un des grands avantages des approches floues est la grande flexibilité des systèmes obtenus, permettant de "sculpter" la fonction selon ses besoins favorisant ainsi la mise au point.

Comme nous l'avons indiqué au cours de la section 5.3.5, les règles du système sont regroupées en comportements élémentaires (un pour la gestion du but et 8 pour l'évitement d'obstacles). Le principe général de la conception est de développer chacun de ses comportements séparément et de les regrouper au sein d'une même base en les hiérarchisant grâce à des poids associés. La liste des comportements et des règles est fournie annexe B.

Le contrôleur obtenu est constitué de deux fonctions (une pour la vitesse linéaire et une pour la vitesse angulaire) comportant chacune 5 variables (3 distances aux obstacles plus la distance et l'angle vers le but). La figure 5.19 représente la commande en vitesse linéaire obtenue en fonction de la distance au but et de la distance à un obstacle frontal. Les autres variables sont gardées constantes. Lorsque l'obstacle frontal est éloigné du robot (distance $> 1.5m$)Γla

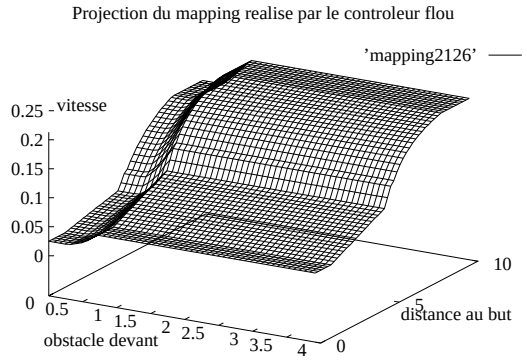


FIG. 5.19 - : *Vitesse linéaire commandée en fonction de la distance au but et à un obstacle frontal.*

vitesse commandée ne dépend que de la distance au but et décroît avec celle-ci. Lorsque la distance devient inférieure à $1.5m$Γle comportement *evite_pres_devant* devient actif et propose une vitesse nulleΓdéformant la fonction. On constate néanmoins que la vitesse commandéeΓcombinaison de la vitesse proposée par les deux comportements actifsΓest non nulle lorsque la distance frontale à l'obstacle tend vers 0Γprovoquant ainsi une collision. Une simple modification des poids de manière à rendre le comportement d'évitement plus prépondérant sur la gestion du but ne résout pas le problème. L'évitement d'obstacles propose une vitesse $v_0 = 0$Γla gestion du but une vitesse $v_1 \neq 0$. La vitesse finale sera comprise entre ces deux valeursΓplus proche de l'une ou de l'autre selon la valeur respective des deux poids.

Ce problème peut être résolu en n'imposant plus une séparation totale entre les deux comportements mais en intégrant les données relatives aux obstacles dans la gestion du but permettant ainsi de l'inhiber en cas de danger :

```
si but_distance est b_pres & non obstacle_devant est o_pres
        alors vitesse est lent_avant
si but_distance est b_normal & non obstacle_devant est o_pres
        alors vitesse est normal_avant
si but_distance est b_loin & but_angle est devant &
        non obstacle_devant est o_pres alors vitesse est rapide_avant
```

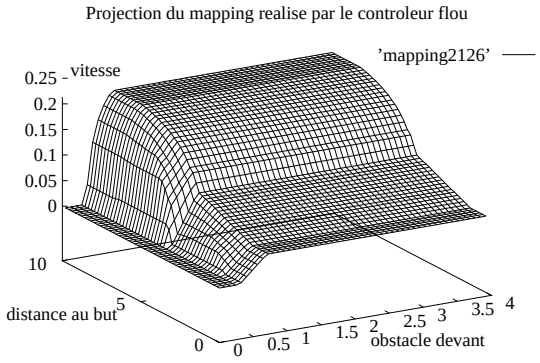


FIG. 5.20 - : *Prise en compte de la distance des obstacles dans le comportement de gestion du but*

La figure 5.20 montre la nouvelle surface obtenue. Ce principe d'inhibition d'un comportement par un autre est utilisé par Watanabe et Pin [184] dans leur méthodologie de construction d'une base de règles floues. L'objectif est de permettre la construction d'un comportement complexe à partir d'un assemblage de comportements flous élémentaires. Le principe consiste entre autre afin d'éviter les conflits de n'autoriser en tout point de l'espace d'entrée qu'au plus un seul comportement pleinement actif (degré d'activation égal à 1). Les zones où aucun comportement n'est pleinement actif sont appelées *zones de tendance*. L'inhibitionΓen n'autorisant qu'un seul comportement à prendre le contrôleΓévite le risque de génération de commandes érronées par résolution de conflits. Elle ne peut en revanche pas s'appliquer à la totalité de l'espace d'entréeΓun des principes du flou étant de ne pas avoir un seul comportement actif à chaque instant mais d'offrir une transition douce entre les comportements existants.

L'exemple que nous avons illustré ici est un exemple simple. Rappelons que les deux fonctions que nous considérons sont définies sur un espace de dimension 5 et que les trajectoires du robot dans cet espace sont rarement limitées à un plan (plus de 2 coordonnées varient simultanément dans le cas général). Ceci rend difficile la visualisation de la surface de commande associée à un cas particulierΓ permettant une analyse visuelle de la source du problème.

5.4.2 Exemples d'exécution

L'exemple d'exécution que nous allons maintenant présenter a été réalisé à l'aide du robot ROBUTER. Le domaine d'évolution du véhicule est un espace libre carré d'environ 5 mètres de côté délimité par un ensemble de tables et de chaises. Nous avons placé un carton au sein de cet espace de manière à créer un obstacle. Le robot ne possède aucune connaissance préalable de son environnement (position du cartonΓdes chaises etc) et doit rejoindre un certain nombre de

buts.

La succession de photos figure 5.21 illustre un exemple de trajectoire réalisée. Le but est indiqué par une croix sur la première image. Etant déjà partiellement engagé derrière le carton, le robot n'est pas en mesure de pouvoir tourner directement afin de l'atteindre. Ce type de situation correspond à un minimum local pour une solution classique de type potentiel. Elle correspond également à un minimum local pour notre système flou dans le cas où le comportement gérant le but n'est pas couplé à la détection d'obstacles (voir section précédente). Dans le cas contraire, le robot n'est plus "attiré" par le but mais gère uniquement les obstacles proches.

Les comportements tels qu'ils sont décrits en annexe B réalisent un suivi de contour. Le robot va effectuer son demi-tour en suivant le bord du carton. Le véhicule se déplace tout d'abord parallèlement à la première face. Une fois celle-ci dépassée, il a de nouveau la possibilité de tourner à droite ou à gauche. La gestion du but n'est plus inhibée et reprend le contrôle en imposant un virage à gauche (direction du but). Le robot se trouve alors en présence de la seconde face du carton qu'il va également suivre et ainsi de suite jusqu'à se trouver face à son objectif.

les différentes expérimentations réalisées nous ont permis de mettre en évidence que le prétraitement sensoriel retenu (voir section 5.3.1) gère de manière satisfaisante des obstacles de petites dimensions reconnus difficiles comme les pieds de chaises et de tables. En effet, les différents cônes d'émission des capteurs latéraux ou frontaux du robot se recouvrent partiellement. Les pieds seront donc détectés dans la majorité des cas et seront perçus comme des barrières, redonnant une consistance à un obstacle n'en possédant pas (voir figure 5.22).

Les règles telles que nous les avons décrites ne permettent pas au robot d'atteindre systématiquement son but et peuvent le laisser bloqué. Les causes de telles situations sont doubles :

1. Incapacité de détecter un passage libre.

2. Conflit entre plusieurs comportements annulant la commande finale.

Le prétraitement que nous avons choisi permet, comme nous l'avons indiqué précédemment, de donner une consistance à des obstacles n'en possédant pas. En contre-partie, il empêche une détection fine de la situation courante et peut percevoir une situation comme bloquée alors qu'elle ne l'est pas (voir figure 5.23). Ce pré-traitement est donc adapté à des environnements peu encombrés contenant des objets de toutes tailles.

La deuxième cause de minimum local est le conflit dans la génération de la commande finale (en particulier entre la gestion du but et l'évitement d'obstacles). Ces conflits surviennent dans la zone de transition entre les régions de prédominance de chaque comportement. L'existence d'une telle zone de transition découle du principe du contrôle flou cherchant à produire une surface continue

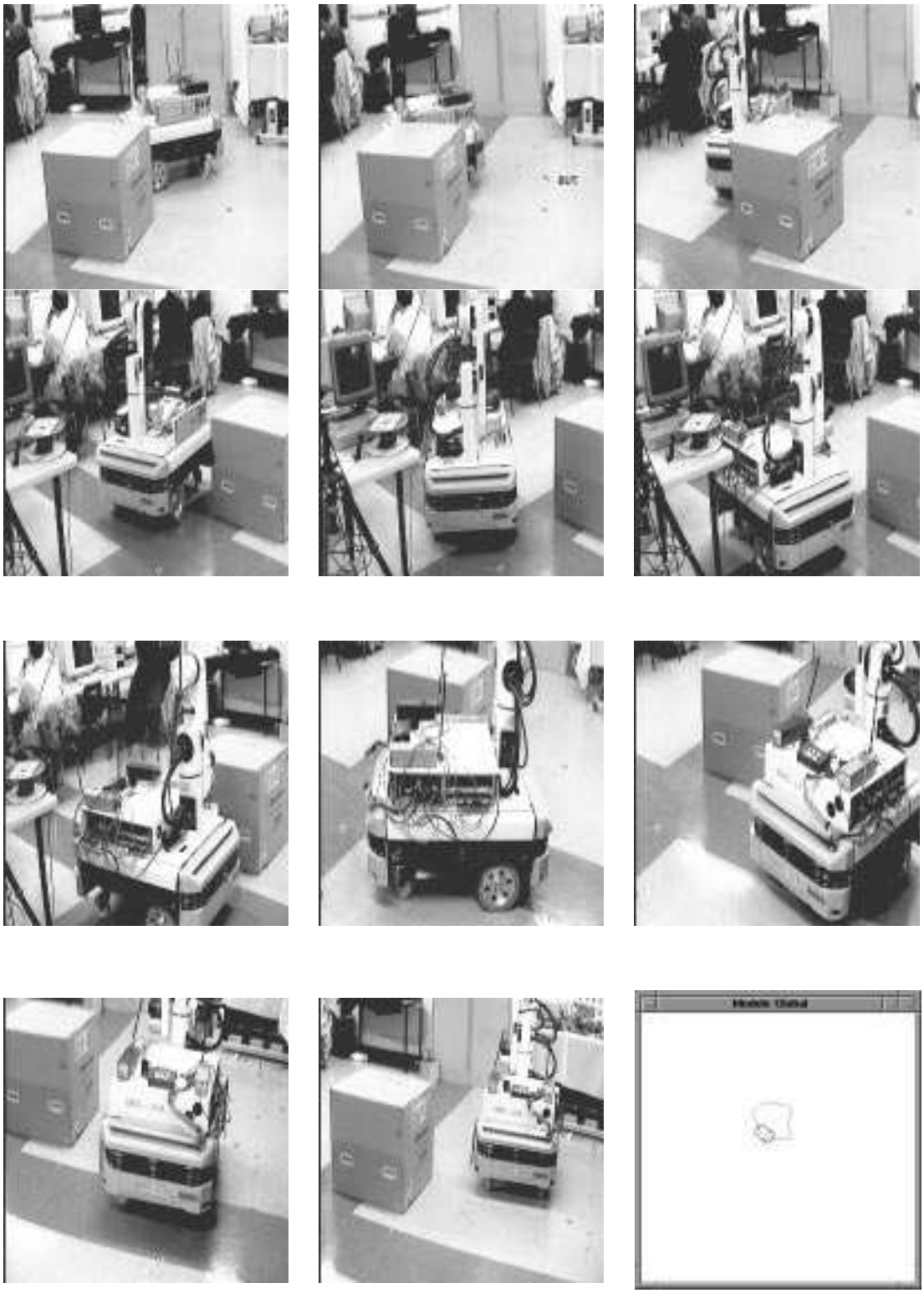


FIG. 5.21 - : *Exemple de déplacement effectué par le robot piloté par le contrôleur flou.*

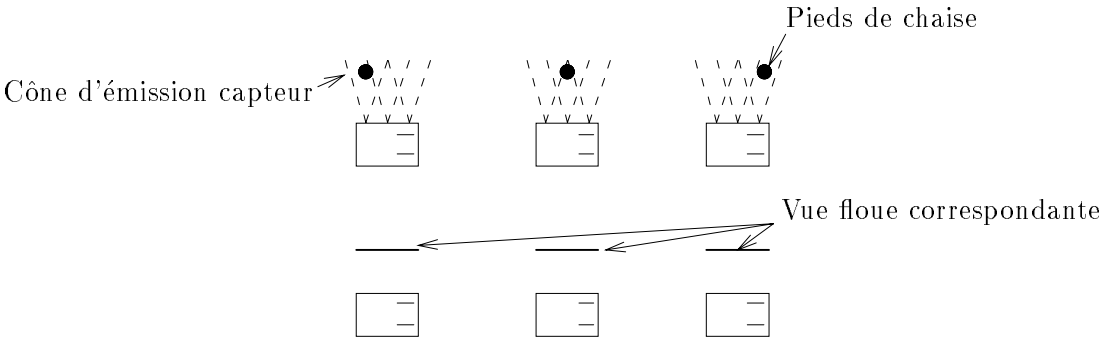


FIG. 5.22 - : *Détection et prise en compte d'obstacles étroits tels que les pieds de chaises.*

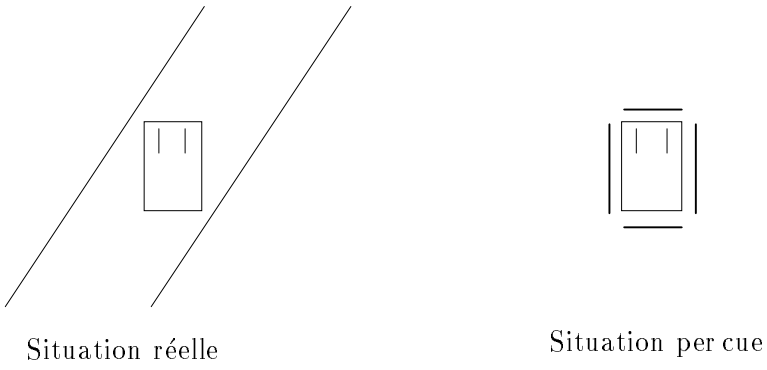


FIG. 5.23 - : *Mauvaise estimation de la situation : le robot pense être bloqué alors qu'il ne l'est pas.*

"douce" entre un ensemble de points imposés. La figure 5.24 illustre un tel blocage. Le robot dispose de zones libres autour de lui et n'est donc pas bloqué par

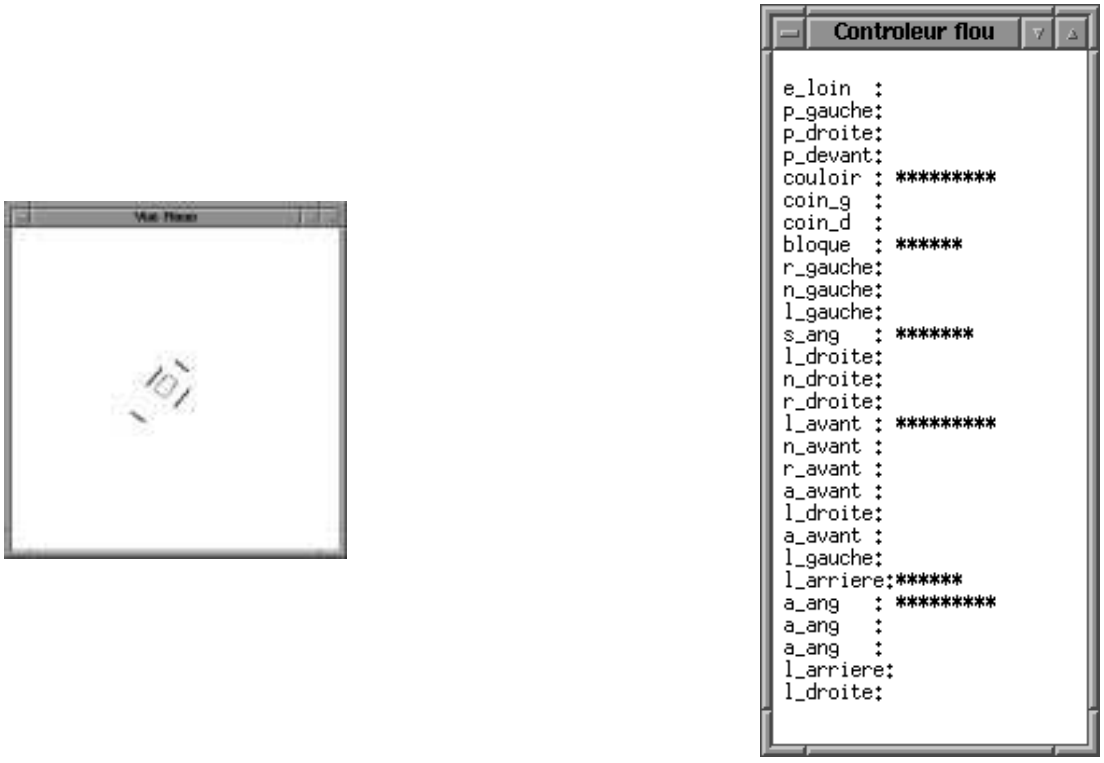


FIG. 5.24 - : *Présence d'un minimum local. La figure de gauche représente les données perceptives utilisée. La figure de droite représente les degrés d'activation des règles.*

les obstacles. Les degrés d'activation des règles laissent apparaître un conflit dans la vitesse de déplacement produisant une commande nulle (*l_arriere* correspond à lent arrière et *lent_avant* à lent avant. Le nombre d'étoiles correspond au degré d'activation de la règle associée).

5.5 Conclusion

La logique floue est un formalisme permettant de construire une transformation continue entre un espace d'entrée et un espace de sortie à l'aide de connaissances fournies par le concepteur et exprimées sous forme de règles. La transformation obtenue peut être facilement éditéeΓmodifiée localement ou nonΓoffrant ainsi une grande souplesse pour la mise au point.

La propriété d'approximateur universel pour les systèmes flous indique que ce formalisme a la puissance nécessaire pour décrire la solution que nous recherchons. Mais il n'existe pas de principe général permettant de déterminer les règles à partir du problème.

Le contrôle flou est basé sur la fusion de résultats individuels fournis par les règles. Ce type d'approche favorise les situations de conflits entre plusieurs propositions. Cela peut conduire à des décisions incorrectes. Cette notion de conflit entre règles est similaire à la notion de conflit entre forces dans les méthodes de type champ de forces et provoque le même type de problèmes (minima locaux ...). Ce point sera redéveloppé lors de la conclusion finale de ce document.

Enfin la logique floue nous a permis de développer un système réactif de navigation réalisant ces objectifs dans certaines situations et échouant dans d'autres. Nous avons constaté en particulier que le réglage des différents paramètres dans un environnement pouvait fournir de bons résultats pour cet environnement et ne pas répondre correctement face à une situation nouvelle. De manière à augmenter l'adptabilité du système de navigation nous nous sommes tournés vers le domaine de l'apprentissage automatique faisant l'objet de la suite de ce manuscrit.

Chapitre 6

L'apprentissage en contrôle

Le but de ce chapitre est de présenter les grandes familles d'approches traitant du problème d'apprentissage en contrôle. Une étude plus approfondie des approches pouvant être utilisées dans le cadre de notre problème sera réalisée au cours des chapitres 7 et 8.

6.1 Introduction

Le contrôle est une discipline dont le but est de fournir une fonction F générant des actions permettant à un processus de réaliser les objectifs désirés (voir figure 6.1).

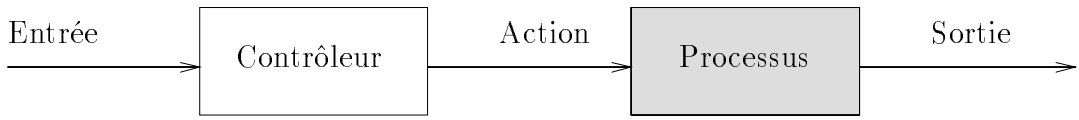


FIG. 6.1 - : *Schéma général d'un système de contrôle*

Depuis le $19^{ième}$ siècle, les automaticiens ont développé une méthodologie permettant de déterminer les contrôleurs adaptés à une classe restreinte de problèmes. En particulier, dans le cadre de processus linéaires et de fonctions objectifs quadratiques, il existe un ensemble de techniques prouvées permettant de déduire un contrôleur adapté possédant des propriétés de stabilité et d'optimalité. Mais de tels résultats ne sont pas encore disponibles pour une famille plus générale de problèmes comprenant des systèmes non linéaires ou stochastiques. En particulier, lorsque les modèles mathématiques fournis sont incomplets ou inadaptés au processus, les résultats obtenus peuvent être médiocres. Face à une telle situation, une des solutions proposée a été d'avoir recours à des techniques de contrôle adaptatif ou à des techniques d'apprentissage.

Comme nous l'avons précisé au cours du chapitre 2Γun système mobile non holonome équipé de capteurs ultrasons est un système non linéaire complexe bruité. Une approche automatique classique visant à extraire analytiquement une loi de commande en fonction du modèle du processus n'est actuellement pas réalisable dans le cas général. Nous avons étudié au cours des chapitres 4 et 5 deux méthodes utilisées pour contourner ce problème. Ces deux approches diffèrent par les techniques mises en œuvre pour les réaliser (forces répulsives et logique floue). Elles sont basées néanmoins toutes les deux sur le même principe consistant pour le programmeur à construire hors ligne un contrôleur ne s'appuyant pas sur un modèle dynamique du système mais sur la façon dont il pense qu'il doit être piloté. La prise en compte des caractéristiques dynamiques est alors réalisée par un ajustement manuel des paramètres sur le robot lors de la réalisation d'essais.

La direction de recherche que nous avons décidé de suivre étant celle de l'apprentissage en contrôleΓnous allons nous intéresser essentiellement à l'architecture des grandes familles d'approches en contrôle adaptatif sans rentrer dans le détail des algorithmes.

6.2 Le Contrôle adaptatif

Cette présentation reprend celle proposée par Irving [82]. Après une définitionΓ nous décrirons les principes généraux de trois grandes approches.

6.2.1 Définition

Les recherches concernant la commande adaptative ont débuté vers le début des années 1950 et ont été motivées par le besoin de fournir une aide de haute performance pour le pilotage d'avions. Après une première période de relatif succèsΓles premiers résultats théoriques fondamentaux sont apparus vers 1960 et ont permis au domaine de réaliser de grandes avancées. De plusΓles progrès rapides de la micro-électronique ont rendu possible la réalisation de tels régulateurs de manière simple et peu coûteuse.

Un régulateur adaptatif est un régulateur de structure classique ou de structure plus complexe muni de coefficients ajustables dont l'ajustement à l'aide d'un algorithme convenable permet d'étendre le domaine de fonctionnement. De manière plus généraleΓon peut également définir un régulateur adaptatif comme toute méthode de commande utilisant une mise à jour en temps réel du modèle mathématique du système à régler. Les différentes méthodes de commande adaptative se différencient par la manière choisie pour réaliser l'ajustement des coefficients. On distingue principalement :

- la commande adaptative à paramètres pré-programmésΓ
- la commande adaptative à modèle de référenceΓ

– la commande adaptative avec identification du système à régler.

Nous allons présenter brièvement le principe de ces différentes approches.

6.2.2 Commande adaptative à paramètres pré-programmés

Le modèle mathématique du système que l'on souhaite asservir dépend de la valeur d'un paramètre θ variable dans le temps. On suppose que le système possède une variable auxiliaire y_θ mesurable physiquement et représentant assez fidèlement les variations du paramètre θ :

$$y_\theta = f(\theta)$$

Dans le cas d'un avion, ce paramètre y_θ peut être par exemple sa vitesse, son altitude ou le niveau de ses réservoirs.

Les paramètres optimaux $v = r(\theta)$ du régulateur peuvent alors être obtenu par la fonction :

$$v = r(\theta) = r(\hat{f}^{-1}(y_\theta))$$

$\hat{f}$ étant une estimation de la fonction f. La fonction $r(\hat{f}^{-1}(.))$ s'appelle la fonction de pré-programmation des paramètres. Le schéma général d'un tel système est représenté figure 6.2. Ce schéma de contrôle adaptatif est très utilisé en milieu industriel. Son inconvénient principal est qu'il n'existe pas de mécanisme permettant de palier une mauvaise fonction de pré-programmation. L'adaptation des paramètres est en quelque sorte réalisée en "boucle ouverte", en ne tenant compte que d'informations "a priori" sur le système. Les deux autres méthodes que nous allons maintenant décrire réalisent quant à elles une adaptation des paramètres en "boucle fermée", tenant compte d'informations "a posteriori" sur le système à régler. Ces informations sont contenues dans les valeurs des variables d'entrée et de sortie du système qui seront fournies aux algorithmes d'ajustement des paramètres.

6.2.3 Commande adaptative à modèle de référence

Dans cette approche, l'utilisateur fournit un modèle appelé *modèle de référence* répondant aux spécifications du comportement *externe* du système bouclé (régulateur + système à régler). Ce modèle de référence admet en entrée la consigne et fournit la sortie désirée du système total. La différence entre la sortie réelle et la sortie désirée permet d'ajuster le régulateur. Comme nous l'avons indiqué dans le paragraphe précédent, le mécanisme d'ajustement utilise également les informations fournies par les variables d'entrée et de sortie du système. L'architecture générale est présentée figure 6.3.

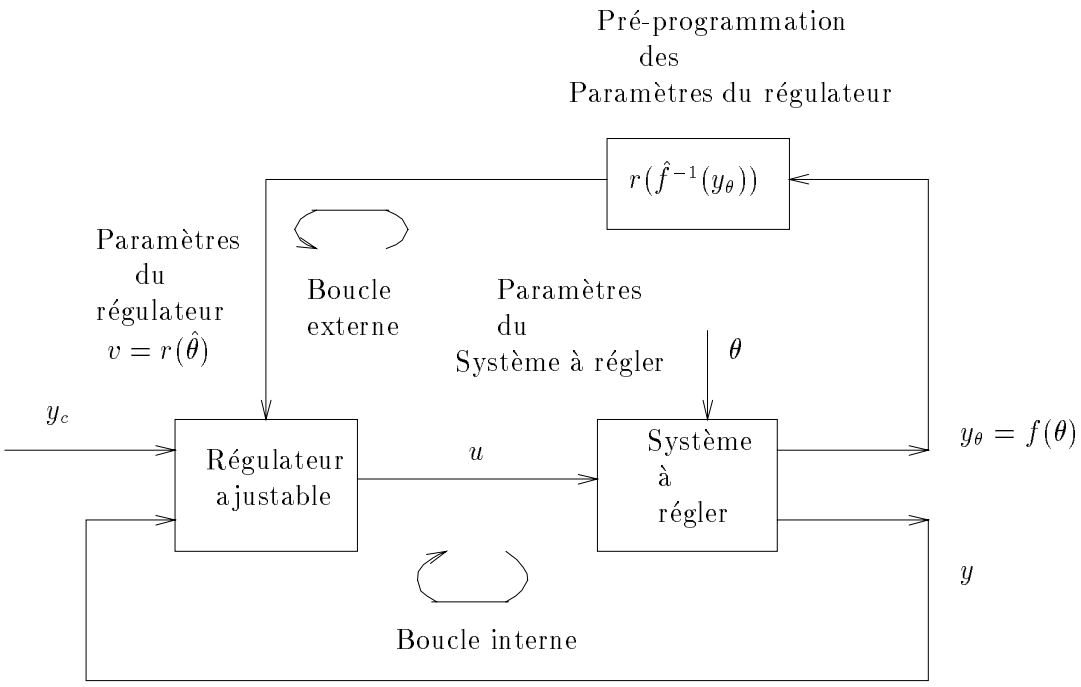


FIG. 6.2 - : *Systèmes adaptatifs à paramètres pré-programmés*

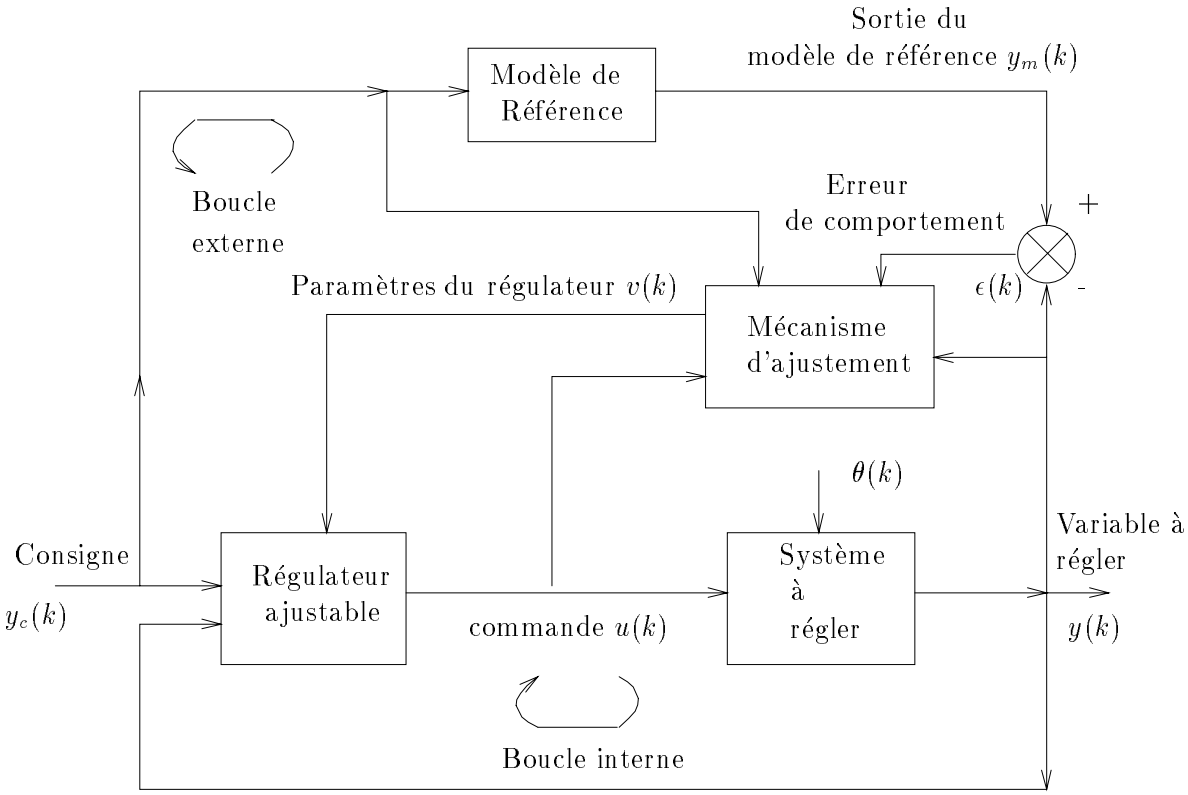


FIG. 6.3 - : *Commande adaptative à modèle de référence*

6.2.4 Commande adaptative avec identification du système à régler

La dernière grande catégorie d'approches et la plus naturelle consiste à identifier à l'aide d'un identificateur à paramètres ajustables le système que l'on souhaite contrôler. Le régulateur est ensuite ajusté grâce à cet identificateur. Le schéma général de ce type d'approche est donné figure 6.4

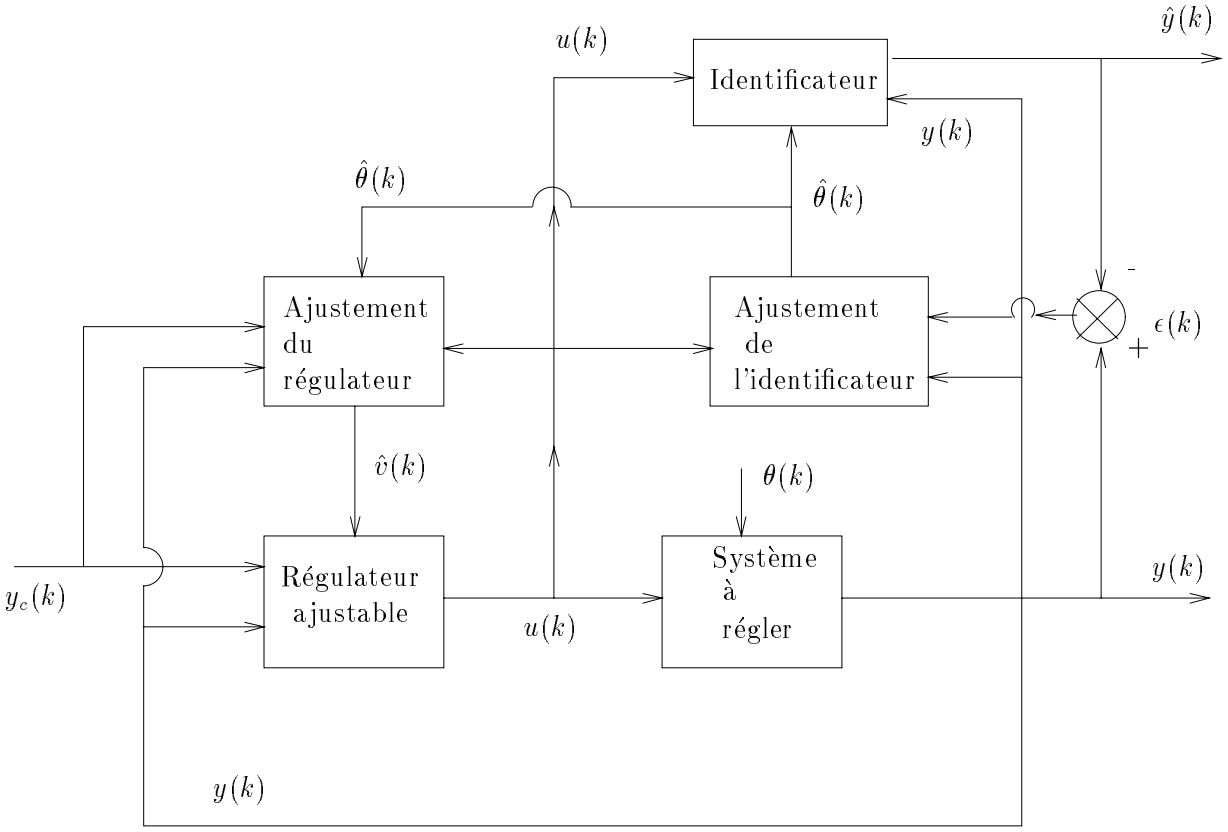


FIG. 6.4 - : *Commande adaptative avec identification du système à régler*

L'ajustement de l'identificateur est réalisé en comparant pour une commande donnée la sortie réelle du processus avec la sortie prévue. La mise à jour de l'identificateur est réalisée en calculant une estimation $\hat{\theta}$ du paramètre θ du système réel. Cette estimation est à l'origine du réglage du régulateur ajustable.

Cette brève présentation du contrôle adaptatif étant achevéeΓnous allons maintenant nous intéresser à l'apprentissage en contrôle.

6.3 L'apprentissage en contrôle

L'apprentissage en contrôle possède des avantages similaires au contrôle adaptatif et utilise des architectures en général très voisines (identification de systèmesΓ etc). La différence entre ces deux approches réside dans les techniques mises en œuvre pour parvenir au but désiré. Alors que le contrôle adaptatif s'appuie sur un modèle paramétrique du système à contrôler et ne s'autorise essentiellement que des mises à jour automatiques des valeurs des paramètresΓl'apprentissage en contrôle peut utiliser des techniques de décision de haut niveauΓde reconnaissance de situations etc. Prenons le cas par exemple d'un processus possédant plusieurs modes de fonctionnement. Alors qu'un processus adaptatif devra réadapter ses paramètres à chaque changement de modesΓun système d'apprentissage en contrôle pourra être en mesure de détecter une situation déjà rencontrée et réutiliser des paramètres calculés lors de la première rencontre.

6.3.1 Définition

Un système de contrôle a pour but de générer des commandes envoyées à un processus afin de satisfaire un objectif. Le comportement du contrôleur peut alors être jugé grâce à des critères de performances. Ces critères peuvent être par exemple la stabilité du systèmeΓson temps de réponseΓsa précision etc. Ils dépendent bien évidemment de l'objectif que l'on s'est fixé. Dans ce contexteΓ apprendre en contrôle signifie modifier le contrôleur de manière à augmenter ses performances telles qu'elles sont mesurées par les critères. Si on considère qu'un contrôleur implémente une fonction F_W : *Entrée* $\mapsto$ *Sortie*Γapprendre signifie déterminer la valeur de l'ensemble des paramètres W. Le terme paramètre est ici pris au sens large et peut désigner aussi bien paramètres numériques que règles dans un système de production ou dans un système flou par exemple.

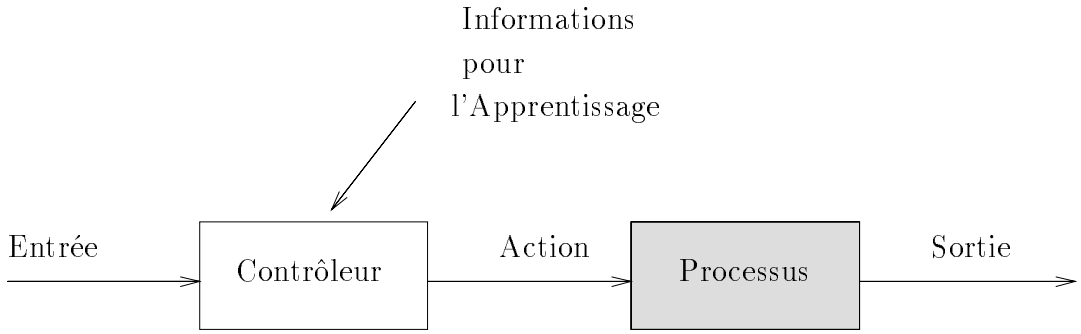


FIG. 6.5 - : *Architecture générale d'un système d'apprentissage en contrôle*

La figure 6.5 représente l'architecture générale d'un tel système. Par rapport à

l'architecture "classique" (voir figure 6.1)Γle contrôleur reçoit en plus des entrées un ensemble d'informations lui permettant d'apprendre. La nature de ces informations va conditionner le type d'apprentissage pouvant être utilisé et permet de classer les différentes approches en plusieurs catégories. Cette classification reprend celle proposée par plusieurs chercheurs et exposée par Gullapalli dans son mémoire de thèse [74].

6.3.2 Informations disponibles : paradigmes d'apprentissage

Selon le type d'informations disponiblesΓon peut distinguer trois catégories d'apprentissage (voir figure 6.6) :

- l'apprentissage superviséΓ
- l'apprentissage par maître distantΓ
- l'apprentissage renforcé.

L'apprentissage supervisé

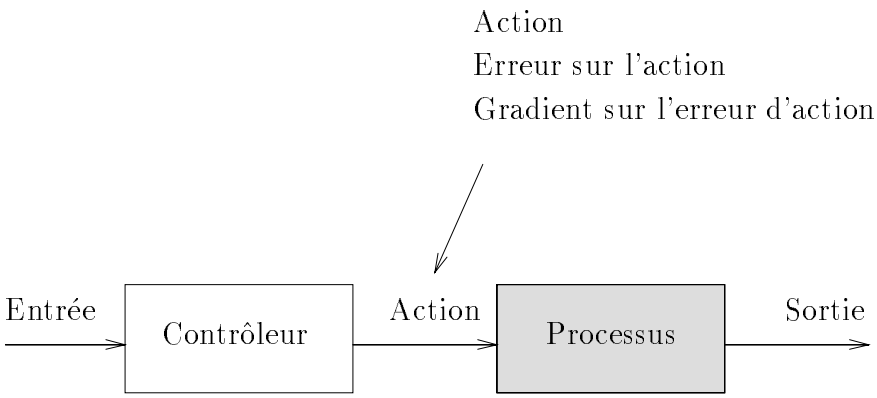
En apprentissage superviséΓle maître fournit soit l'action qui devrait être exécutéeΓsoit un gradient sur l'erreur commise (au niveau de l'action). Dans les deux casΓle maître fournit au contrôleur une indication sur l'action qu'il devrait générer afin d'améliorer ses performances.

L'apprentissage renforcé

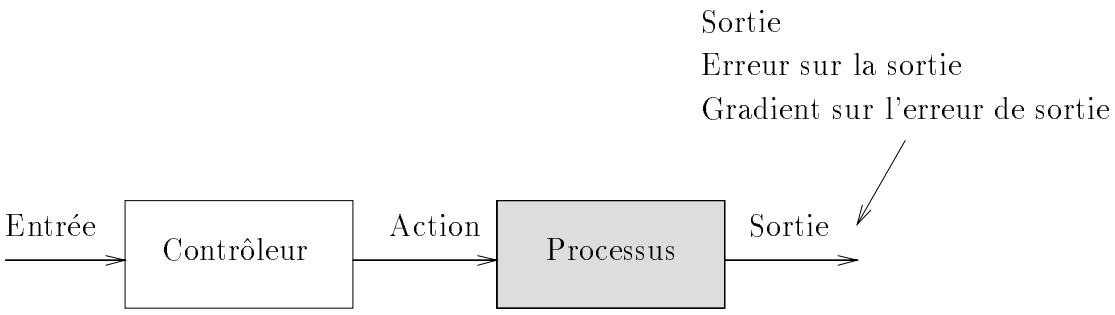
Par oppositionΓle maître en apprentissage renforcé a un rôle d'évaluateur et non pas d'instructeur. Il est en général appelé *critique*. Le rôle du critique est de fournir une mesure (en général le critère de performance cité précédemment) indiquant si l'action générée par F_W est appropriée ou non. Il laisse le contrôleur déterminer seul comment il doit modifier ses actions de manière à obtenir une meilleure évaluation dans le futur. Contrairement donc à l'apprentissage supervisé Γle critique connaît les objectifs visés mais ne sait pas comment les atteindre.

L'apprentissage avec un maître distant

L'apprentissage par maître distantΓcomme l'on appelé Jordan et RumelhartΓ est une troisième catégorie d'apprentissage située entre l'apprentissage supervisé et l'apprentissage renforcé. Le maître est dit distant car les informations fournies pour l'apprentissage sont en relation avec la sortie du processus et pas directement avec les actions générées par le contrôleur. Ces actions sont de type "sortie"Γ "erreur sur la sortie"Γou "gradient de cette erreur". Ce type d'informations est



a) Apprentissage Supervisé



b) Apprentissage avec maître distant

c) Apprentissage Renforcé

FIG. 6.6 - : *Représentation des trois catégories d'apprentissage : l'apprentissage supervisé, distant et renforcé*

celui naturellement disponible dans beaucoup de problèmes de contrôle tels que le suivi de trajectoire ou l'asservissement sur une position donnée.

Comme dans le cas de l'apprentissage renforcéΓle contrôleur doit découvrir seul l'action qu'il doit générer afin d'obtenir la sortie désirée. On peut d'ailleurs remarquer qu'en considérant l'association "processus - critique" comme un seul processusΓl'apprentissage renforcé peut alors être vu comme un cas particulier de l'apprentissage avec maître distant.

Maintenant que nous avons brièvement présenté les différents types d'informations disponiblesΓnous allons nous intéresser aux principales méthodes d'apprentissageΓpermettant d'exploiter ces informations afin d'atteindre l'objectif désiré.

6.3.3 Méthodes d'apprentissage

Les méthodes présentées brièvement ici reflètent le type d'informations disponibles. Nous allons les replacer dans le cadre de l'apprentissage supervisé Γrenforcé et distant.

L'apprentissage supervisé

Comme nous l'avons rappelé précédemmentΓle but de l'apprentissage est de fournir une valeur à l'ensemble de paramètres W associés au contrôleur F_W. Dans le cas de l'apprentissage supervisé Γles informations disponibles sont soit l'actionΓ soit l'erreur sur cette actionΓsoit encore un gradient de cette erreur par rapport à W. Modifier W est alors relativement direct. Les algorithmes peuvent néanmoins être très sophistiqués de manière à garantir à F_W des propriétés satisfaisantes d'interpolation et d'extrapolation (par rapport à l'ensemble des données initialement fournies). Parmi les algorithmes classiquesΓcitons l'utilisation des réseaux de neurones et plus particulièrement l'algorithme de rétropropagation ([134Γ58] par exemple). Nous reviendrons plus en détails sur ce point lors du prochain chapitre.

L'utilisation de telles approches présuppose l'existence d'un expert capable de fournir un ensemble d'exemples formés de situations et d'actions correctes associées. Ces exemples doivent être suffisamment nombreux et représentatifs de la tâche à accomplir. Ceci peut être une réelle difficulté dans le cas de systèmes complexes [134Γ191]).

Apprentissage distant et apprentissage renforcé

Les méthodes d'apprentissage associées à l'apprentissage renforcé ou à l'apprentissage distant sont plus complexes à mettre en œuvre que les méthodes associées à l'apprentissage supervisé. Ceci est dû essentiellement au fait que les actions correctes à exécuter dans une situation donnée ne sont pas fournies directement par le maître mais doivent être inférées à partir du couple *(entrée, sortie)*

ou *(entrée, évaluation)* dans le cas de l'apprentissage distant ou de l'apprentissage renforcé respectivement. Il existe un *vide* entre les informations disponibles pour le contrôleur (sortie, évaluation) et les informations dont il a besoin pour remplir sa tâche (actions à exécuter).

Les méthodes développées pour permettre l'apprentissage dans de tels systèmes ont pour but de combler ce vide. On peut les classer en deux grandes catégories selon qu'elles utilisent ou non un modèle :

- les méthodes indirectes,
- les méthodes directes.

Nous allons maintenant revenir sur ces deux grandes catégories d'approches.

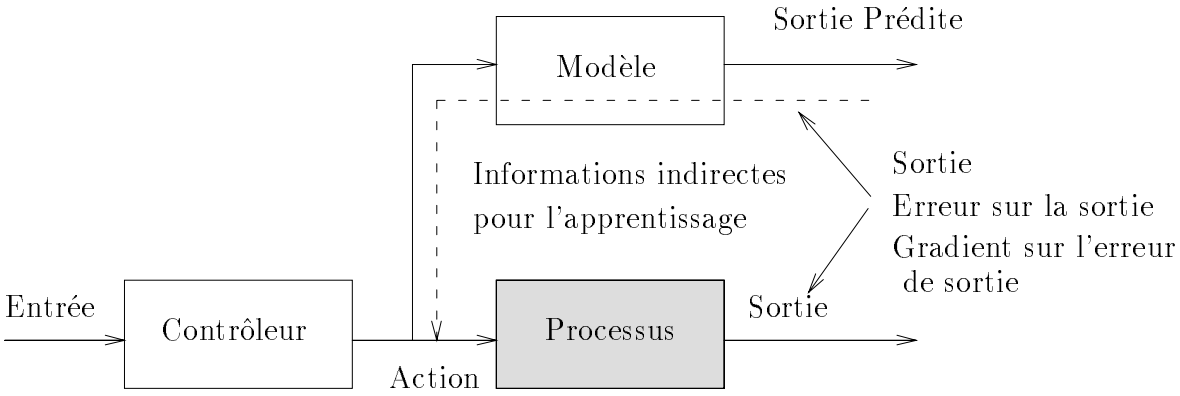
Les méthodes indirectes

Les méthodes indirectes proposent d'obtenir des informations directement utilisables par le contrôleur et donc de combler le vide en construisant un modèle de la transformation $action \mapsto évaluation$ (ou *sortie* dans le cas de l'apprentissage distant). Ce modèle peut être utilisé d'au moins deux manières différentes.

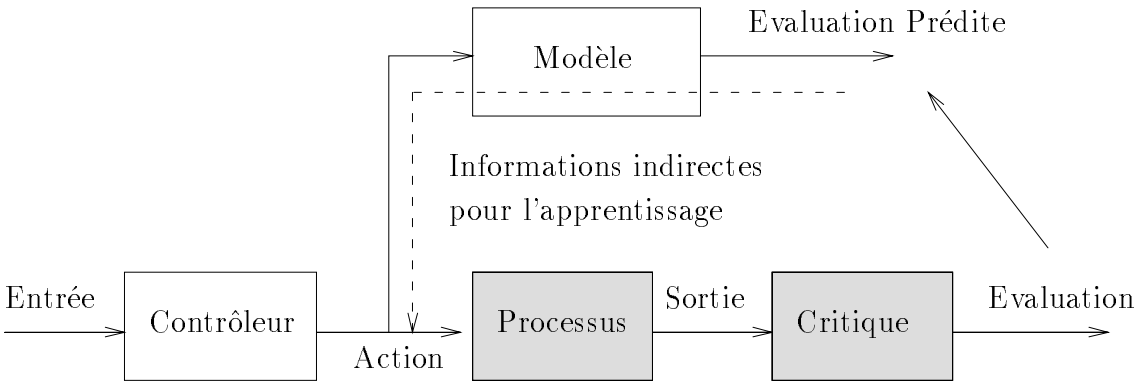
Il peut tout d'abord être utilisé de manière directe afin de simuler le comportement du processus dans le temps. Ceci a surtout été utilisé en intelligence artificielle et plus particulièrement dans les programmes de jeux [154] afin de générer des arbres de recherche. Mais il peut également être utilisé pour des problèmes de contrôle. Le principal inconvénient d'une telle approche est que la recherche directe est en générale sous-contrainte et très chère en terme de coût de calculs.

Par opposition, le modèle peut être utilisé de manière indirecte, dans un rôle d'explication, permettant d'inférer les actions appropriées qui satisferont les objectifs de contrôle. En particulier, si le modèle de la transformation est différentiable, il est possible grâce au jacobien de connaître la variation nécessaire des actions afin d'obtenir la variation souhaitée de la sortie (respectivement de l'évaluation). Cette approche est illustrée figure 6.7. Une telle information peut également être obtenue dans le cas de systèmes non différentiables. Si par exemple le modèle utilisé est une base de règles, il est alors possible de connaître l'entrée à fournir afin d'obtenir la sortie désirée grâce à l'utilisation du chaînage arrière.

Jordan et Rumelhart se sont intéressés à l'apprentissage indirect par maître distant et ont proposé une approche neuronale de ce problème [88]. Celle-ci est basée sur l'utilisation de la rétropropagation. Un premier réseau de neurones est entraîné de manière à obtenir un "modèle direct" du processus à contrôler. Cette phase correspond à une phase d'identification. Une fois que le modèle obtenu est jugé suffisamment proche du système, il est figé et le contrôleur peut commencer à apprendre. Le modèle direct a pour rôle de prédire la sortie correspondant à l'action proposée (voir figure 6.7). La différence entre la sortie prédite et la sortie



a) Apprentissage avec maître distant



b) Apprentissage Renforcé

FIG. 6.7 - : *Utilisation d'un modèle de la transformation* (action, sortie) *ou* (action, évaluation).

souhaitée est rétropropagée à travers le modèle direct fournissant la variation d'action nécessaire. Cette variation d'action est ensuite utilisée pour permettre au contrôleur d'apprendre. Jordan et Rumelhart [88] ont mis également en évidence qu'en rétropropageant non plus la différence entre la sortie prédite et la sortie souhaitée mais la différence entre la sortie réelle et la sortie souhaitée le contrôleur est encore capable d'apprendre une loi de commande correcte en utilisant un modèle direct imprécis.

Nous allons maintenant présenter la deuxième famille d'approches proposant de combler le vide entre les informations disponibles et les actions à exécuter : il s'agit des méthodes directes.

Les méthodes directes

Par opposition aux méthodes indirectes les méthodes directes n'utilisent pas un modèle du processus mais le processus directement afin d'acquérir des données pour l'apprentissage.

Dans le cas de l'apprentissage distant si l'entrée du contrôleur correspond à la sortie désirée du processus (par exemple une consigne en position) on peut alors réaliser une identification directe du modèle inverse (voir figure 6.8). Lors de la phase d'apprentissage un ensemble d'actions aléatoires est fourni au processus. Celui-ci va générer les sorties correspondantes. Les couples *(sortie, action)* sont présentés au contrôleur comme exemples afin de lui permettre d'apprendre un modèle inverse du processus. Une fois que la phase d'apprentissage est terminée la sortie du contrôleur ainsi obtenu est directement connectée à l'entrée du système à contrôler.

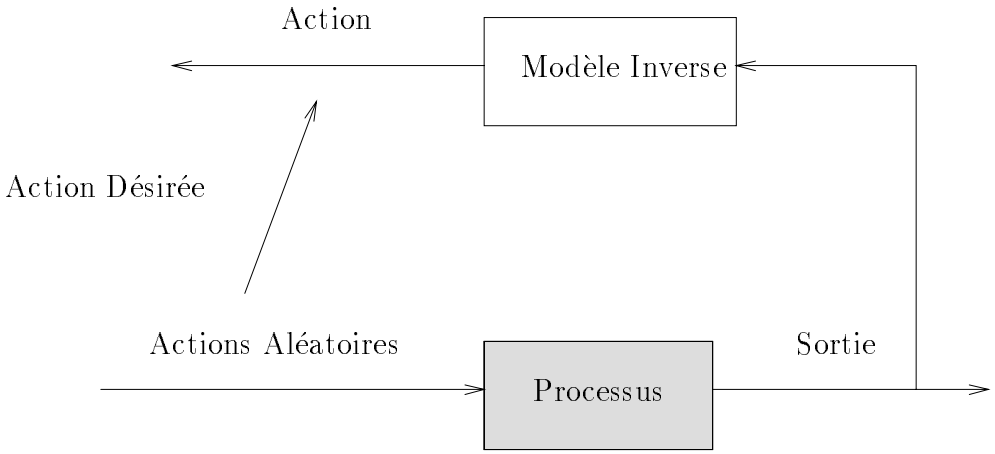
Il existe une deuxième sous-famille d'approches directes dont le but est de déterminer directement le gradient de la sortie (ou de l'évaluation) en fonction de l'action. Cette estimation est réalisée en perturbant l'action envoyée au processus et en analysant les conséquences de cette perturbation sur le signal de sortie (figure 6.9). Cette perturbation peut être réalisée en ajoutant par exemple un bruit aléatoire. Cette idée est très ancienne et a tout d'abord été utilisée pour la détermination de paramètres en contrôle adaptatif. Elle est également à la base de la plupart des approches en apprentissage renforcé direct (voir par exemple [74 121 23]).

6.3.4 Paradigmes et algorithmes

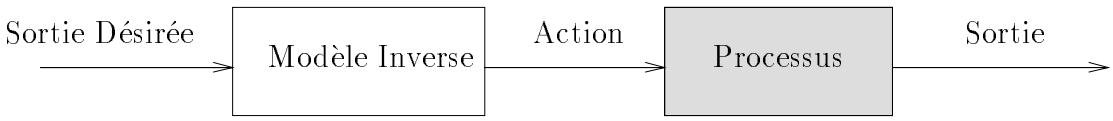
Les différents types d'approches ont été présentés. On distingue 5 catégories :

Supervisé

Maître Distant	Renforcé	
		Méthodes Indirectes
		Méthodes Directes

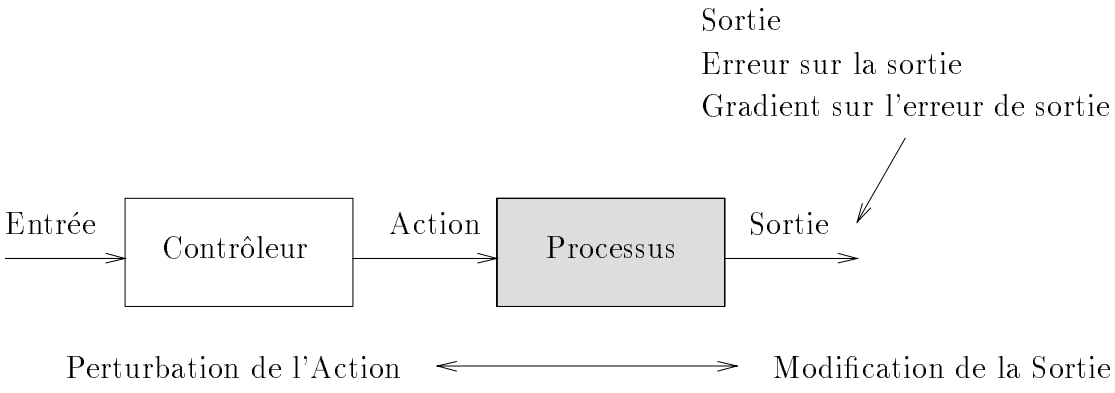


a) Apprentissage du modèle inverse

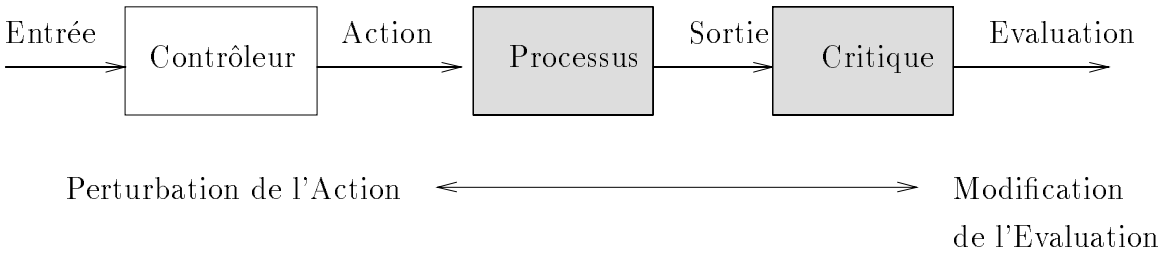


b) Utilisation du modèle inverse comme contrôleur.

FIG. 6.8 - : *Création du contrôleur par apprentissage direct du modèle inverse*



a) Apprentissage avec maître distant



b) Apprentissage Renforcé

FIG. 6.9 - : *Détermination du gradient de la transformation* action $\mapsto$ sortie *ou* action $\mapsto$ évaluation *par perturbation de l'action.*

Le choix entre ces paradigmes d'apprentissage est déterminé par la nature des informations disponibles et par la façon dont le problème est posé. Comme l'ont souligné Jordan et Rumelhart [88]Γil existe une différence entre un *paradigme d'apprentissage* et un *algorithme* d'apprentissage. Il est toujours possible par exemple de transformer une erreur sur une action en critère d'évaluation au moyen d'une norme signée. Ceci signifie qu'un problème relevant d'un paradigme d'apprentissage supervisé peut être traité à l'aide d'un algorithme d'apprentissage supervisé ou à l'aide d'un algorithme d'apprentissage renforcé. De mêmeΓun problème d'apprentissage renforcé peut être transformé en problème d'apprentissage distant en considérant le couple processus + module d'évaluation comme le processus à contrôler.

Les différents principes généraux de l'apprentissage en contrôle étant présentésΓnous allons nous intéresser à leur application possible à la robotique.

6.4 Application à l'apprentissage en robotique

La robotique est un domaine présentant un ensemble de problèmes particuliers. Ces problèmes (que nous rappellerons dans un premier temps) imposent une restriction sur les algorithmes d'apprentissage utilisés qui doivent être en mesure de les pendre en compte et de les traiter. Nous resituerons ensuite les différents paradigmes de l'apprentissage dans le cadre de la robotique.

6.4.1 Problèmes particuliers

La robotique est un domaine traitant de l'interaction de systèmes réels dans un environnement réel. Ceci est à l'origine de problèmes non triviaux devant être considérés lors du choix d'un algorithme d'apprentissage :

Bruit des capteurs. Les capteurs utilisés en robotique sont en général des capteurs bon marché (comme les capteurs ultrasons) mais pouvant générer des données bruitées (comme la mauvaise estimation d'une distance) ou incohérentes (détection d'un obstacle n'existant pas).

Résultat d'actions non déterministe. Un robot est un système mécanique complexe dont les caractéristiques sont susceptibles de se modifier au cours du temps (vibrationΓvariation du niveau de charge des batteriesΓetc). De plusΓle véhicule n'a pas une connaissance complète de l'environnement qui l'entoure et peut ne pas être en mesure de détecter un élément important. Par exempleΓdans le cas d'un véhicule à chenilleΓle type de revêtement utilisé sur le sol a une influence sur la position du centre de rotation du robot. Dans le cas du robot ROBUTER utilisé au LIFIAΓla position des roues folles à l'avant du véhicule peuvent générer un écart non prévisible lors du démarrage du véhicule. Tous ces éléments font que la même action

exécutée par le robot dans deux situations qu'il perçoit comme identique peut avoir deux résultats différents.

Réactivité. Un robot doit pouvoir réagir dans une situation imprévue avec un temps de réponse compatible avec la situation. S'il est par exemple en train de traverser une voie encombréeΓil ne peut pas se permettre de s'arrêter au milieu afin d'évaluer la situation et d'élaborer un plan d'action.

Incrémentalité. Dans le cas d'un robot mobile autonome évoluant dans un environnement dynamique qui ne lui est pas réservéΓl'ensemble des situations possibles auxquelles il sera confronté ne peut pas être prévu à l'avance. De plusΓcomme nous l'avons indiqué précédemmentΓle fonctionnement même du robot est susceptible d'évoluer au cours du temps. Il sera donc confronté lors de son existence à des situations où pour pouvoir continuer sa tâche il sera nécessaire de modifier ou d'enrichir ses connaissances sans pour autant remettre en cause celles déjà acquises. L'espace d'entrée étant très vasteΓstocker l'ensemble des données utilisées depuis le début du processus d'apprentissage nécessiterait trop d'espace mémoire et n'est pas donc envisageable. De mêmeΓsélectionner afin de ne garder parmi la totalité des données que celles représentatives de la tâche et donc utiles pour l'apprentissage est un problème délicat [134Γ191]. Il est préférable que l'apprentissage d'une nouvelle connaissance soit réalisé par présentation de nouvelles données sans avoir à présenter à nouveau les anciennes. L'algorithme utilisé doit être incrémental.

Temps limité pour l'apprentissage. Dans le cadre d'un apprentissage en ligne réalisé sur le robotΓon ne dispose que d'une puissance de calcul limitée. De plusΓle robot devant continuer d'évoluerΓil est nécessaire que le résultat de l'apprentissage soit disponible le plus rapidement possible. Le coût d'un algorithme d'apprentissage peut être important si :

1. il s'exécute en un seul pas complexeΓ
2. la convergence nécessite l'exécution d'un nombre important de pas.

Si la fréquence de présentation de chaque situation est élevéeΓle problème correspondant à la deuxième situation peut être résolu par l'emploi d'un algorithme incrémental. Lors de la présentation d'une situationΓseul un nombre faible de pas est réalisé de manière à conserver un temps d'exécution raisonnable. La convergence est alors assurée par les diverses présentations au cours du temps de la même situation lors du fonctionnement du système. Cette solution n'est pas applicable dans le cas d'un robot mobile. L'espace perceptuel est très vaste et la fréquence de présentation d'une même situation est donc très faible. Il est nécessaire d'utiliser au maximum chaque situation présentée.

Les propriétés nécessaires que doivent avoir les algorithmes d'apprentissage ayant maintenant été présentéesΓnous allons resituer les différents paradigmes dans le cadre de la robotique.

6.4.2 Paradigmes possibles

La robotique est un domaine très vaste dans lequel on retrouve les trois paradigmes cités sous-section 6.3.2. Nous allons donner brièvement quelques exemples pour chacun d'entre eux.

Dans le cadre de l'apprentissage supervisélΓle système apprend grâce à un expert fournissant un ensemble d'exemples représentatifs de son fonctionnement. Un tel paradigme se retrouve dans deux situations :

- le système robotique auquel on s'intéresse est actuellement contrôlé par un opérateur humain. Ce schéma correspond à celui d'un système téléopéré (voir par exemple [134Γ136Γ135]). De telles approches ont connu un grand succès industriel pour les robots de peinture ou de manutention sur les chaînes de montage automobile par exemple.

- le système robotique est actuellement asservi par un contrôleur "rigide" dont les performances sont jugées insuffisantes (voir par exemple [61Γ89]).

Dans le cadre de l'apprentissage distantΓle système apprend grâce aux informations disponibles à la sortie du processus. Ce paradigme se retrouve en robotique dans les problèmes suivants :

- le système (robot de manipulation ou robot mobile) dispose ou construit de manière automatique une description géométrique de son environnement. En fonction du but fixéΓil construit ensuite grâce à une planification de mouvements le chemin permettant de se rendre de sa position courante à sa position finale. Le contrôleur a enfin pour tâche d'asservir le système robotique sur le chemin ainsi calculé. Les informations disponibles sont alors les écarts commis par le robot par rapport à la trajectoire idéale ([43] par exemple).

- le système accomplit sa tâche en suivant un chemin présent physiquement dans l'environnement et détecté par ses capteurs. Cela peut être par exemple le suivi d'un mur dans un bâtiment ou le suivi d'une ligne blanche ou d'un bord de route pour un système en extérieur ([179] par exemple).

- le système doit suivre un autre véhicule.

Dans le cadre de l'apprentissage renforcéΠle système ne dispose plus d'exemples de commandes ou de trajectoires de référence. L'apprentissage est réalisé uniquement grâce à une évaluation des résultats basée sur l'objectif fixé. Seul le but est

donné. Le travail nécessaire pour le concepteur dans les approches classiques ou dans les approches supervisées (indiquer les étapes pour parvenir au but) est ici laissé à la machine (voir [75Γ123] par exemple). Ce paradigme correspond en fait à un des objectifs fondamentaux de la robotique.

Notre étude porte sur la navigation en milieu encombré et sur l'évitement d'obstacles par l'utilisation de systèmes réactifs. Comme nous l'avons indiqué auparavantΓnous nous sommes intéressés plus particulièrement au problème du couplage direct entre la perception et l'action. Nous ne considérons pas les approches basées sur la construction préalable d'un chemin. Notre problème peut en revanche se définir naturellement comme :

- Comment atteindre le but en évitant les obstacles.
- Comment reproduire un comportement similaire à celui d'un véhicule contrôlé par un opérateur humain.

Il s'apparente donc plus à un paradigme d'apprentissage supervisé ou renforcé qu'à un paradigme d'apprentissage distant. Nous allons nous intéresser en détail à ces deux types d'approches au cours des chapitres 7 et 8.

Chapitre 7

Naviguer par l'observation

Nous allons nous intéresser au cours de ce chapitre à l'utilisation du paradigme de l'apprentissage supervisé dans le cadre de la navigation réactive pour un robot mobile. Ce travail a été réalisé en collaboration avec Volker Hansen dans le cadre de son mémoire de DEA [77].

7.1 Introduction

Nous avons étudié au cours des chapitres 4 et 5 deux approches heuristiques pour la détermination d'une fonction réalisant l'association action-perception dans le but d'accomplir une tâche de navigation. Nous allons maintenant nous intéresser au cours de ce chapitre et du chapitre suivant à la détermination automatique de telles fonctions.

Le paradigme utilisé au cours de ce chapitre est celui de l'apprentissage supervisé tel qu'il a été défini section 6.3.2. Il s'agit à partir d'un ensemble $\{(\vec{x_1}, \vec{y_1}), \ldots, (\vec{x_n}, \vec{y_n})\}$ d'exemples de couples *(entrée, sortie)* de déterminer la fonction f telle que :

- f satisfait les exemples : $\forall i \in \{1, \ldots, n\}\ f(\vec{x_i}) = \vec{y_i}$
- f possède de " bonnes " propriétés de généralisation :

$$\forall x\ \forall i\ \vec{x} \neq \vec{x_i},\ \vec{y} = f(\vec{x})$$

  $\vec{y}$ étant une commande permettant de satisfaire l'objectif imposé par la tâche de navigation. Cette propriété de généralisation est capitale dans le type de problème que nous traitons car l'espace d'entrée est vaste et seule une petite sous-partie pourra être présentée sous forme d'exemples.

L'apprentissage supervisé est un apprentissage généralement exécuté hors-ligne. Cela signifie que parmi les différents points mis en évidence lors de la sous-section 6.4.1Γ le temps de convergence n'est pas un des plus importants.

Un algorithme nécessitant l'exécution de plusieurs centaines de pas avant de converger ne constitue pas dans ce cas particulier un problème. L'incrémentalité reste en revanche un des critères majeurs pour le choix d'une approche.

L'une des formes les plus classiques de l'approximation de fonctions est l'approximation polynomiale. Supposons que l'on souhaite approcher une fonction $f : \Re \mapsto \Re$ à partir d'un ensemble de n couples de données $(x_i, f(x_i))$. Il s'agit de déterminer les m coefficients w_i du polynôme $P(x) = w_1 + w_2 x + \ldots + w_n x^{n-1}$ approchant le mieux la fonctionΓc'est-à-dire minimisant la moyenne de l'erreur quadratique :

$$E = \frac{1}{n} \sum_{i=1}^{n} \epsilon(x_i)^2 \tag{7.1}$$

ϵ étant défini par :

$$\epsilon(x) = f(x) - P(x)$$

La valeur de E après minimisation permet de juger la qualité de l'approximation. Cette qualité est par rapport aux points définis et ne préjuge en rien de la qualité de la généralisation obtenue. Le nombre m de coefficients reflète à la fois la quantité de mémoire permettant de stocker l'approximateur ainsi que le nombre de couples exemples qu'il sera nécessaire pour le déterminer. Un résultat classique de ce domaine est qu'un approximateur quelconque possédant m paramètres nécessitera un nombre d'exemples proportionnel à m afin de les déterminer. Soit P un polynôme défini de $\Re^n$ dans $\Re$ et de degré d. Le nombre total de coefficients est :

$$K_d^0 + K_d^1 + \ldots K_d^n$$
$$\text{avec } K_m^p = \frac{(m+p-1)!}{p!(m-1)!}$$

A titre d'exempleΓun polynôme de degré 5 sur un espace d'entrée de dimension 5 (dimension de l'espace réduit utilisé par le système flou) nécessite la détermination de 252 coefficients. Un polynôme de degré 5 sur un espace d'entrée de dimension 26 (24 capteurs ultrasons + la distance et l'orientation vers le but) nécessite 169911 coefficients. Le nombre de coefficients nécessairesΓet par conséquent la taille mémoire et le nombre d'exemples requisΓexplose avec la dimension de l'espace d'entrée. L'approche polynomiale ne convient donc pas dans le cadre de notre problème.

Il existe dans le domaine des réseaux de neurones un ensemble d'approches possédant la propriété *d'approximateurs universels parcimonieux*. La propriété de parcimonie signifie que le nombre de paramètres à déterminer n'explose pas avec la dimension du problème. Cette propriété très importante sur le plan pratique (taille mémoire + nombre d'exemples) ainsi que d'autres propriétés que nous verrons au cours de la prochaine section ont contribué à l'utilisation de telles techniques dans de nombreux domaines et de nombreuses applications.

7.2 Apprentissage neuronal de fonctions

Les réseaux de neurones artificiels ont été introduits aux alentours de 1940 afin de modéliser le fonctionnement du cerveau humain. L'objectif était d'obtenir un système automatique reproduisant les propriétés des systèmes biologiques :

- un parallélisme massif permettant d'obtenir des temps de réponse très rapides malgré la lenteur des entités de calcul et la complexité de l'information traitée.

- une capacité d'apprentissage permettant de généraliser à partir d'un nombre limité d'exemples.

- une capacité de prendre en compte des informations bruitées et inconsistantes en réalisant un calcul robuste et tolérant contre les fautes.

La dénomination réseau de neurones est actuellement une dénomination assez floue englobant un nombre très vaste de techniques ayant souvent un rapport éloigné avec le fonctionnement du cerveau humain mais possédant en commun la mise en œuvre d'automates pouvant fonctionner en parallèle et pouvant communiquer entre eux. Ils sont utilisés entre autre pour des problèmes d'approximation de fonctions de classification (la classification peut éventuellement être vue comme un cas particulier de l'approximation de fonction) de filtrage etc.

Nous allons maintenant présenter quelques unes des grandes familles de réseaux utilisés dans le domaine de l'approximation de fonctions que nous avons regroupées en deux catégories :

- Utilisation d'un modèle fixe.

- Apprentissage du modèle.

7.2.1 Utilisation d'un modèle fixe

De manière générale nous désignerons par $F(W, X)$ une famille de fonctions admettant pour variable le vecteur X et pour paramètre le vecteur W. Le choix d'une fonction particulière au sein de cette famille est réalisé par instanciation du vecteur paramètre W. Nous désignerons par modèle fixe les familles de fonctions telles que le vecteur W est de dimension fixée n'évoluant pas au cours de l'apprentissage. Dans le cas de l'approximation polynomiale telle qu'elle a été décrite au cours de l'introduction de ce chapitre cela signifie que le degré des polynômes utilisés est figé.

Il existe une grande variété d'approches dans ce domaine. Nous allons revenir sur deux d'entre elles.

Radial Basis Function

L'idée générale est la suivante: l'approximation polynomiale a recours à un ensemble de monômes dont la combinaison pondérée doit approcher la fonction que l'on cherche. Chacun de ces monômes est défini sur la totalité de l'espace d'entrée. On peut également réaliser l'approximation de la fonction en créant une partition de l'espace d'entrée et en associant à chacune de ces partitions un nombre réel. Dans le cas d'une fonction de $\Re$ dans $\Re$l'on obtient une approximation à l'aide d'une fonction constante par morceau. Les monômes sont remplacés par les fonctions caractéristiques des éléments de la partition. On peut encore aller plus loin en remplaçant la partition par un découpage en région pouvant se chevaucher et en remplaçant la fonction caractéristiquel'constante par définition sur la région à laquelle elle est associée par une fonction ϕ quelconque prenant ses valeurs sur cette région. Les régions sont alors appelée des *champs récepteurs*[1]. F s'écrit :

$$F = \left\{ \sum_{i=1}^{n} w_i \phi_i(\vec{x}) / w_i, \phi_i(\vec{x}) \in \Re \right\}$$

Si la fonction $\phi_i(\vec{x})$ peut se mettre sous la forme $\phi_i(\vec{x}) = \Phi_i(\|\vec{x} - \vec{x_i}\|)$l'on parle alors de fonctions radiales. L'une des fonctions radiales la plus fréquemment utilisée est la gaussienne :

$$\phi_i(\vec{x}) = e^{-\left(\frac{\|\vec{x}-\vec{x_i}\|}{r_i}\right)^2}$$

$\vec{x_i}$ correspondant au centre et r_i au rayon du champ récepteur. La représentation neuronale de ce système est donné figure 7.1. Le vecteur d'entrée x est présenté simultanément à toutes les cellules. Le degré d'activation de chaque cellule est donnée par la fonction ϕ_i. Ces degrés sont ensuite transmis à la couche de sortie où ils sont sommés. La liaison entre la cellule activée i et la couche de sortie est pondérée par le poids w_i.

Comme dans le cas de l'approximation polynomialel'l'apprentissage est réalisé en déterminant les coefficients minimisant l'erreur donnée par l'équation 7.1. La fonction qui associe l'erreur aux coefficients est une fonction convexe possédant un seul minimum au point d'annulation de son gradient. Les coefficients w_i sont solutions du système d'équation :

$$\nabla \left(\frac{1}{k} \sum_{i=1}^{k} \epsilon(\vec{x_i})^2 \right) = -\frac{2}{k} \sum_{i=1}^{k} \epsilon(\vec{x_i}) \phi(\vec{x_i}) = 0 \qquad (7.2)$$

avec :

$$\begin{aligned} &\phi(\vec{x}) = (\phi_1([\vec{x}), \phi_2(\vec{x}), \ldots, \phi_n(\vec{x})) \\ &W = (w_1, w_2, \ldots, w_n) \\ &\epsilon(\vec{x}) = y(\vec{x}) - W\phi(\vec{x}) \end{aligned}$$

1. En anglais : receptive fields

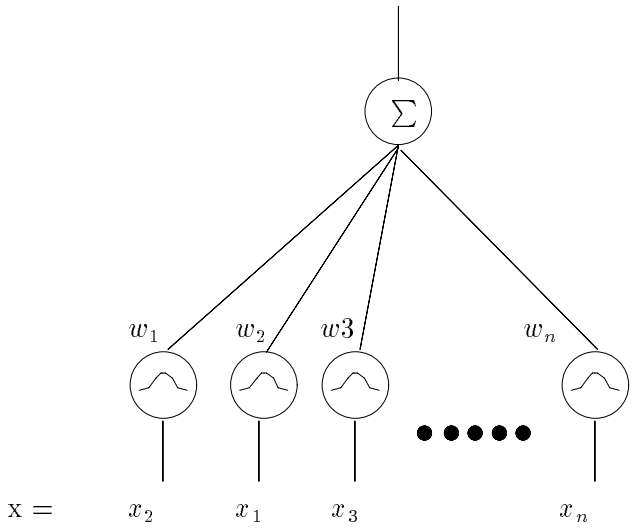


FIG. 7.1 - : *Architecture d'un réseau de type* Radial Basis Function.

La résolution du système complet et donc la détermination des paramètres en une seule étape peut être remplacée par un processus itératif de recherche d'un minimum appelé descente de gradient (le gradient indiquant la direction opposée dans laquelle il faut se déplacer pour atteindre le minimum) :

$$w_{t+1} = w_t + \beta \sum_{i=1}^{k} \epsilon_t(\vec{x_{i,t}})\phi(\vec{x_{i,t}}) \tag{7.3}$$

où β représente la vitesse d'apprentissage. La détermination des paramètres grâce à l'équation 7.2 ou à l'équation 7.3 nécessite la connaissance de la totalité des couples de points $(x, f(x))$ et ne respecte donc pas notre besoin d'incrémentalité cité sous-section 6.4.1. Dans le cas où les exemples ne sont connus que un par un Widrow et Hoff [187] ont proposé une formule de mise à jour incrémentale des paramètres ne calculant le gradient que sur le point courant au lieu de tous les points :

$$w_{t+1} = w_t + \beta\epsilon_t(\vec{x_t})\phi(\vec{x_t}) \tag{7.4}$$

Cette loi de mise à jour présente des similarités avec celle proposée par Rosenblatt dans le cadre du perceptron [145]. Elle est connue sous le nom de *Least Mean Square* ou *LMS*.

Seul le point courant étant nécessaire au calcul la loi de mise à jour *LMS* permet en théorie de réaliser un système incrémental. Cela ne reste vrai en pratique que si les ensembles de poids mis à jour par chaque point exemple sont relativement disjoints (champs récepteurs correctement répartis par rapport aux

exemples et délimitant des régions à peu près disjointes dans le cas d'un réseau RBF). Dans le cas contraire, lors de la prise en compte d'un nouveau point, l'algorithme pourrait être en mesure de modifier des poids fixés par un point précédent et donc d'oublier ce point. Ce phénomène d'oubli, que l'on retrouve pour la même raison chez les réseaux à propagation unilatérale que nous définirons par la suite, est illustré annexe C.

Nous avons jusqu'à maintenant supposé connus le centre et le rayon des différents champs récepteurs. Ces champs peuvent être disposés de manière uniforme dans l'espace d'entrée. Cette solution n'est adéquat que si la fonction f est elle aussi échantillonnée de manière régulière et ne présente pas d'irrégularité (voir [54] par exemple). Dans le cas contraire, il est nécessaire d'apprendre la position de ces différents champs lors d'une première phase à l'aide d'un algorithme de classification de type *k-mean* par exemple (description dans [54]). Ce recours à une double phase pour l'apprentissage est une fois de plus en désaccord avec notre besoin d'incrémentalité. Il serait préférable de pouvoir apprendre simultanément la position des champs récepteurs ainsi que les paramètres w_i. Ceci conduit à des modèles de réseaux plus complexes et en particulier aux réseaux à propagation unilatérale[2].

Les réseaux à propagation unilatérale

Un réseau à propagation unilatérale est un réseau formé d'une couche d'entrée, d'une couche de sortie et d'une ou plusieurs couches intermédiaires appelées couches cachées. Chaque neurone d'une couche c est connecté au travers de poids w à un ou plusieurs neurones de la couche $c+1$ (voir figure 7.2). Les neurones calculent la somme pondérée de leurs entrées $I = \sum_{i=1}^{n} w_i o_i$ et transmettent en sortie $O = f(I) = \frac{1}{1+e^{\frac{-x}{T}}}$.

Un réseau à couche réalise une transformation fortement non linéaire des données d'entrée vers les données de sortie. L'algorithme d'apprentissage a pour tâche d'ajuster les différents poids w afin de minimiser l'erreur de sortie fournie par l'équation 7.1 et apprendre ainsi à la fois un codage interne des entrées et une transformation de ce codage vers les valeurs de sortie.

L'approche utilisée pour la mise à jour des paramètres est également basée sur le principe de la descente de gradient. Contrairement aux réseaux de type *Radial Basis Function*, la fonction associant l'erreur aux poids n'est plus convexe et il y a donc risque de présence de minima locaux. Il n'y a plus garantie de convergence vers le minimum global. La transformation *(entrée,sortie)* réalisée par le réseau peut être très complexe et n'est pas facilement explicitable. Un certain nombre de méthodes ont été développées de manière à pouvoir néanmoins calculer ses dérivées partielles par rapport aux différents paramètres w afin de déterminer le gradient. La plus célèbre d'entre elles est la rétro-propagation du gradient

2. En anglais : feedforward networks

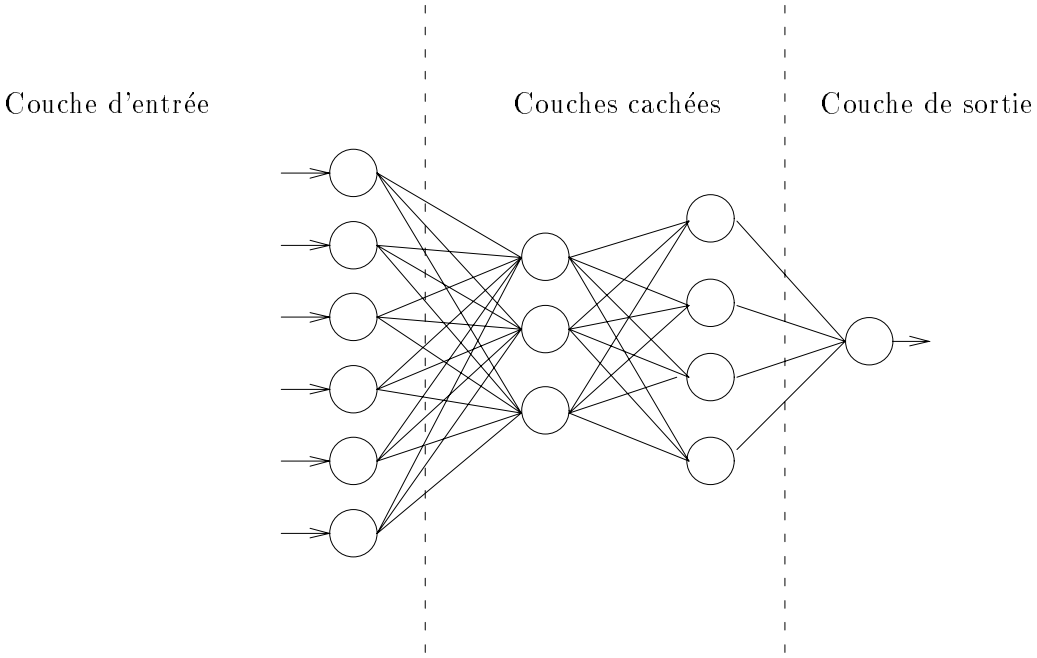


FIG. 7.2 - : *Architecture d'un réseau à propagation unilatérale comprenant 2 couches cachées.*

proposée par Rumelhart [147] dont nous ne détaillerons pas les calculs ici.

L'utilisation de cet algorithme de mise à jours des paramètres nécessite la connaissance de l'ensemble des points à apprendre sous peine " d'oublis ". De plusΓla convergence de la fonction d'erreur vers le minimum est en générale très lente à cause d'interactions entre les coefficients lors de l'apprentissageΓ(voir [87] par exemple) et nécessite donc beaucoup de cycles de calculs et une multiple présentation des exemples.

Les deux méthodes présentées au cours de cette sous-sectionΓoutre leur non incrémentalitéΓont comme point commun la nécessité de préciser un modèle avant de démarrer la phase d'apprentissage. Dans le cas des *Radial Basis Functions*Γce modèle est le nombre de champs recepteurs ainsi que leur position si on souhaite réaliser l'apprentissage en une seule passe. Dans le cas des réseaux à propagation unilatéraleΓle modèle est déterminé par le nombre de couches cachées ainsi que le nombre de neurones sur ces couches cachées (le nombre de neurones sur les couches externes étant bien évidemment fixé par la dimension de l'espace d'entrée et de sortie). La détermination de ce modèle est très importante et a de grandes conséquences sur les capacités de généralisation du système.

On génère un ensemble de points répartis aléatoirement sur la surface $z = f(x, y)$. Ces points sont présentés à deux réseaux de neurones différents. La figure 7.3 affiche les deux surfaces apprisent.

$$\forall x \in [0,1]\ \forall y \in [0,1]\ f(x,y) = 0.9e^{\frac{(x-0.5)^2+(y-0.5)^2}{0.25^2}}$$

La figure de gauche représente la fonction apprise par un réseau comportant une couche cachée de 5 neurones. La figure de droite représente la fonction apprise par un réseau comportant une couche cachée de 6 neurones. Le modèle utilisé par le réseau de droite génère une fonction passant par les différents points imposés mais ne propose pas une généralisation correcte.

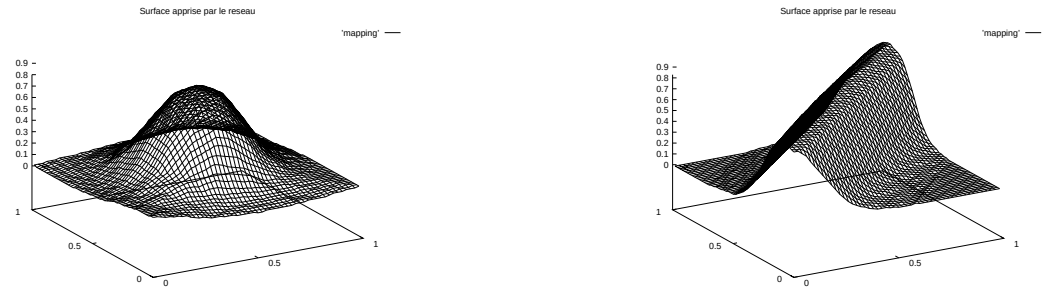


FIG. 7.3 - : *Les fonctions proposées par le réseau de la figure de gauche et de la figure de droite passent toutes les deux par les points imposés. Mais seule la fonction de la figure de gauche généralise correctement.*

Rissamen [144] a démontré qu'une explication des exemples avec un modèle simple est plus probable qu'une explication complexe. Il faut donc rechercher le

réseau le plus simple possible capable d'apprendre les exemples en espérant qu'il fournisse une bonne généralisation. La généralisation est validée en séparant la base de données en deux parties : la première sert à l'apprentissage la deuxième permet de tester la généralisation. Le choix du modèle est très important. Dans le cas de la navigation réactive (comme dans la plupart des problèmes) nous ne possédons pas d'informations permettant de guider ce choix. Il est nécessaire de procéder par essais successifs. Une telle approche peut être coûteuse en temps et en moyen de calcul.

Nous allons considérer maintenant un ensemble de méthodes "moins rigides" permettant d'apprendre le modèle de la fonction ainsi que la fonction elle-même.

7.2.2 Apprentissage d'un modèle

Nous allons nous intéresser au cours de cette sous-section au problème plus général consistant à trouver le modèle de la fonction et à apprendre ses paramètres. Il existe pour ce faire deux manières de procéder [5] :

- l'approche destructive : on part d'un réseau suffisamment complexe pour englober la solution et on le simplifie au cours de l'apprentissage.

- l'approche constructive : on part au contraire d'un réseau très simple que l'on complexifie au fur et à mesure.

Les méthodes destructives supposent que l'on soit en mesure de borner la complexité du système ce qui est une hypothèse assez forte. Elles reposent le plus généralement sur l'utilisation de la rétropropagation du gradient impliquant une présentation multiple des exemples contraire à nos hypothèses. Parmi ces approches nous pouvons citer par exemple *Optimal Brain Damage* [161] ou *Weight Decay* [76].

Les approches constructives ajoutent au fur et à mesure de l'apprentissage de nouvelles connections et de nouveaux neurones à un réseau à l'origine simple. Le problème de ces méthodes constructives vient du fait que les associations déterminant la sortie du réseau sont souvent distribuées sur plusieurs connections. L'ajout d'un nouvel élément ne doit pas perturber les associations déjà apprises et donc diminuer les performances du système. Un des moyens de limiter les effets perturbateurs est de subdiviser le réseau global en sous-réseaux entraînés indépendamment afin d'empêcher toute interaction entre eux et que l'on peut donc intégrer de manière incrémentale. Fahlman propose par exemple dans son approche appelée *Cascade Correlation* [57] de construire le réseau en ajoutant successivement de nouvelles couches. Chaque couche ajoutée est entraînée puis définitivement figée avant la création de la couche suivante. Jordan propose une division verticale en sous-réseaux appelés experts. Le vecteur d'entrée est présenté en parallèle à chacun des sous-réseaux. Les sorties respectives sont ensuite combinées par une somme pondérée pour obtenir le résultat final. Les coefficients

de pondération sont fournis par un autre réseau chargé d'arbitrer les sous-réseaux et donc de désigner lesquels sont experts pour la situation courante. Ces deux approches (combinaison d'experts et cascade corrélation) font appel à la rétro-propagation.

La distribution des associations peut également être évitée en assurant qu'un nombre limité de neurones (en général 1) soient activés pour chaque entrée. Cette stratégie conduit aux méthodes de compétition entre les neurones et aux mécanismes de subdivision de l'espace d'entrée en régions. On effectue ensuite sur chacune de ces régions une approximation locale (voir [14] par exemple) grâce à une valeur ou une fonction associée. L'avantage d'une telle approche face à une approche distribuée est la relative possibilité d'explication du système appris en analysant les différentes régions et donc de réduire l'effet "boite noire" à l'origine de nombreuses critiques envers les systèmes neuronaux.

Nous décrirons tout d'abord l'approche CMAC en raison de son importance historique. Nous nous intéresserons ensuite aux approches *Supervised Growing Cell Structures*, *Fuzzy Artmap* et *Grow and Learn*.

L'approche CMAC

L'approche CMAC[3] a été conçue initialement par Albus [4] comme un modèle de l'apprentissage d'actions moteurs dans le cortex cérébral. L'idée de base est de projeter un espace de grande taille sur un espace de taille plus réduite par l'intermédiaire d'une fonction de "hashing". Chaque élément de l'espace réduit correspond à une région de l'espace initial. Le hashing est une fonction associant à un point x le numéro de la région à laquelle il appartient.

Un des risques classiques de l'utilisation de réduction de dimension d'un espace est celui de *collision*, où deux points A et B différents de l'espace initial et tels que $f(A) \neq f(B)$ peuvent se retrouver projetés en un même point de l'espace réduit. Afin d'éviter ce problème et de faciliter le mécanisme de généralisation, on utilise m fonctions de hashing $Map_i(x)$ et donc m partitions différentes de l'espace en cellules. A chaque point d'entrée est associé m cellules mémoires de l'espace réduit. La structure du réseau est représentée figure 7.4. Nous avons considéré dans cet exemple afin de simplifier la représentation que l'espace d'entrée X est discret et que chaque cellule de la couche d'entrée représente un point de cet espace. Dans ce cas précis, chaque point de l'espace d'entrée est affecté à deux cellules mémoires ($m = 2$). Dans le cas où X est continu, il faut tout d'abord réaliser m partitions différentes de X, les fonctions $Map_i(x)$ implémentant alors un hashing entre la région à laquelle appartient x et les n cellules mémoires de l'espace réduit.

La sortie $o(x)$ du réseau CMAC est calculée par la moyenne des valeurs asso-

3. Cerebellar Model Articulation Controller

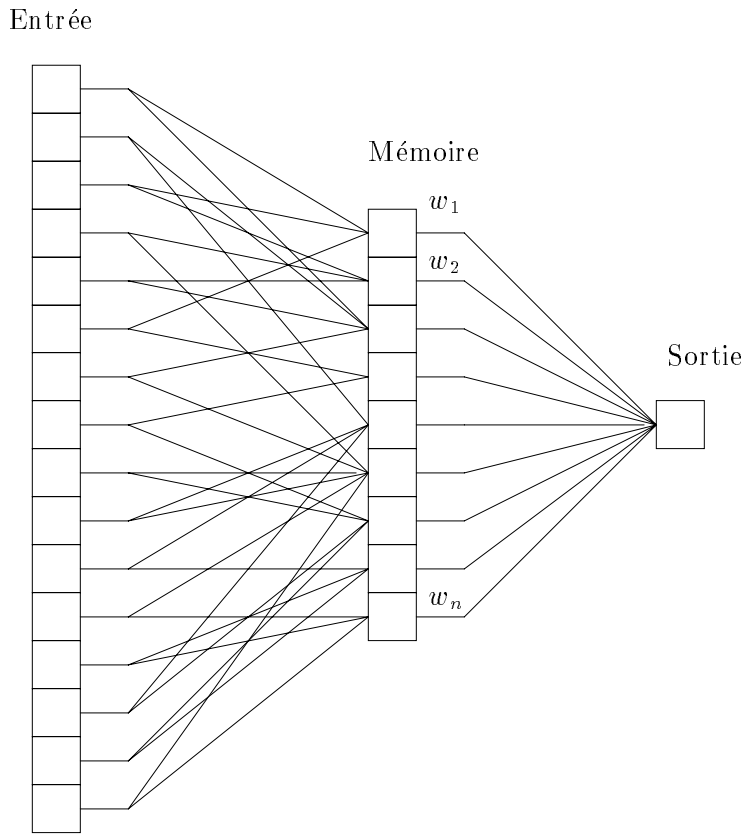


FIG. 7.4 - : *Structure d'un réseau CMAC. Chaque neurone de la couche d'entrée représente un point de l'espace X*

ciées aux cellules mémoires actives :

$$o(x) = \frac{1}{m} \sum_{i=1}^{n} w_k \ / \ k = Map_i(x)$$

L'apprentissage est réalisé en ajustant les différents paramètres w_i par une loi s'inspirant de la descente de gradient (on ne peut pas parler de gradient la fonction réalisée par CMAC n'étant pas continue et donc pas différentiable) :

$$w_i^{t+1} = w_i^t + \beta \epsilon_t(x_t) \phi_i(x_t)$$

avec :

$$\phi_i(x) = \begin{cases} 1 & \text{si } \exists j,\ 1 \leq j \leq m \wedge Map_j(x) = i \\ 0 & \text{sinon} \end{cases}$$

L'idée intuitive de cette règle d'apprentissage est très simple : seul le paramètre des cellules mémoires ayant servi au calcul de la fonction est mis à jour dans la direction de l'erreur. Chaque élément x de l'espace d'entrée X détermine m régions se recoupant (une région par partition différente de l'espace d'entrée). Ces m régions vont être projetées par Map_i sur m cellules mémoires distinctes servant à coder la valeur de f associée à x mais également aux x' "voisins" de x. Un élément x' proche de x a en effet toutes les chances d'être associé aux mêmes m cellules mémoires. Plus x' est éloigné de x et plus ce nombre d'éléments communs risque de diminuer. Lorsque l'on met à jour la valeur $y = f(x)$ on met à jour également la valeur y des voisins de x de manière plus ou moins importante selon la proximité de ses voisins et donc selon le nombre d'éléments mémoires partagés et donc corrigés. Cette propriété très intéressante pour la généralisation impose néanmoins une limitation sur les variations de courbure de la fonction que l'on peut approcher. Celle-ci doit être relativement régulière car CMAC ne possède une capacité mémoire limitée pour la représenter sur la totalité de son domaine.

Le système CMAC tel qu'il vient d'être décrit implémente un approximateur à modèle fixe : nombre de cellules mémoires et donc nombre de paramètres figés. De manière à pouvoir gérer le compromis entre le temps d'apprentissage et la qualité de l'apprentissage CMAC a été utilisé au sein d'une approche multi-résolution pouvant réaliser ainsi un système à modèle non fixe. Le principe est le suivant :

- On réalise une partition grossière de l'espace d'entrée (régions peu nombreuses et donc de grande taille). On entraîne CMAC sur cette partition créant une première approximation y_1 de la fonction f. Le nombre de régions étant faible l'apprentissage sera très rapide mais le résultat sera de mauvaise qualité.

- De manière à raffiner cette première approximation on crée une deuxième partition de l'espace X plus fine. Le nouveau réseau CMAC associé est alors entraîné sur la fonction $f - y_1$. L'apprentissage sera un peu plus long mais $y_2 + y_1$ sera une meilleure représentation de f que y_1

– L'étape précédente peut être répétée jusqu'à obtenir le degré de précision souhaité.

Cette approche multi-résolution permet de complexifier le réseau total en fonction du degré d'approximation souhaité. Il faut remarquer néanmoins qu'il est nécessaire d'attendre la convergence du niveau n avant d'ajouter le niveau $n+1$. Cela signifie qu'il faut présenter à plusieurs reprises les données. L'approche n'est pas incrémentale.

Une version incrémentale de l'approche multi-résolution consiste à entraîner simultanément tous les réseaux CMAC en réduisant la vitesse d'apprentissage pour les partitions les plus fines de l'espace. MaisΓon retrouve à ce moment là un approximateur à modèle fixe.

Growing Cell Structure et Supervised Growing Cell Structures

Avant de détailler l'algorithme d'apprentissage supervisé *Supervised Growing Cell Structure*Γnous allons nous intéresser à son noyau : l'approche *Growing Cell Structure*Γréalisant une classification non supervisée des données d'entrée.

L'approche *Growing Cell Structures* conçue par Bernd Fritzke [62Γ63] est une extension du modèle de Kohonen [100]. Le modèle de Kohonen permet la construction non supervisée d'une mise en correspondance d'un espace de dimension n vers un espace de cellules de dimension inférieure (en général de dimension 2). A chacune de ces cellules i est affectée une position dans l'espace initial $\Re^n$ par l'intermédiaire d'un vecteur de poids w_i. Lors de la présentation au système d'un vecteur d'entrée xΓune seule cellule est activée selon le principe du *winner takes all* basé sur une norme Euclidienne :

$$\forall i \in \{1, \ldots, n\} \; \|w_j - x\| \leq \|w_i - x\|$$

Les différents neurones du réseaux sont alors mis à jour lors d'une phase d'apprentissage :

$$\forall i \in \{1, \ldots, n\} \; w_i(t+1) = w_i(t) + h_{ji}(t)[x(t) - w_i(t)]$$

avec $h_{ji}(t)$ un ensemble de fonctions de t décroissantes et telles queΓà t fixéΓ $h_{jj}(t) = \max_i h_{ji}(t)$. De plusΓgénéralementΓ$h_{ji}(t) > h_{jk}(t)$ si la cellule k est plus éloignée de j que la cellule i. L'ensemble de ces fonctions permet de définir le voisinage de la cellule activée qui sera effectivement adaptée lors de la phase d'apprentissage ainsi que l'amplitude de ces modifications.

La mise en correspondance de $\Re^n$ vers l'ensemble des cellules apprises par l'approche de Kohonen présente deux propriétés fondamentales [100]:

– elle préserve la topologie des données. Deux vecteurs adjacents de $\Re^n$ seront associés à deux cellules adjacentes ou similaires. RéciproquementΓdeux cellules adjacentes ont des vecteurs poids similaires dans $\Re^n$.

– elle préserve la distribution des données. Soit $P(X)$ une distribution de probabilité à partir de laquelle on génère un ensemble de vecteurs exemples x_i de $\Re^n$. La mise en correspondance apprise préserve la distribution des données signifie que chaque cellule a une probabilité égale d'être activée par présentation des vecteurs x_i. Cela signifie que la densité des vecteurs positions associés aux cellules réalise une approximation de la densité de probabilité de $P(X)$.

Un réseau de Kohonen est initialement constitué de cellules dont les poids sont répartis aléatoirement dans l'espace $\Re^n$. Au fur et à mesure de l'apprentissageΓ ces différents poids vont se déplacer vers les zones possédant une forte densité de points exemplesΓne laissant que quelques poids dans les zones plus " pauvres ".

Le nombre de cellules choisi joue un rôle très important pour le succès de l'approche. Il doit être adapté à la complexité du problème. Le réseau proposé par Fritzke permet de compléter l'approche de Kohonen en ajoutant incrémentalement de nouvelles cellules. La structure de voisinage choisie est une structure en triangle : chaque cellule est le sommet d'un triangle dont les côtés représentent les connections vers d'autres cellules. A chacune de ces cellulesΓon associe de plus un compteur indiquant le nombre de fois où cette cellule a été activée. Celles-ci devant être activées de manière équiprobableΓla cellule i dont le compteur est le plus élevé représente un bon candidat pour accueillir dans son voisinage la création d'un nouvelle cellule k. Cette nouvelle cellule est insérée entre la cellule j et sa voisine la plus éloignée i. Son vecteur poids est initialisé à $\frac{w_j+w_i}{2}$. Le réseau est réarrangé de manière à garder une structure triangulaire. Le compteur de k et de ces voisins sont ajustés de manière à reproduire les valeurs qu'ils auraient eues si k avait été présent dès le départ. De mêmeΓles cellules trop peu sélectionnées sont éliminées et le réseau réorganisé.

Cette approche permet d'approcher la distribution des exemples $P(X)$ de manière incrémentale et sans avoir recours à un modèle fixe. Nous allons maintenant voir son application dans le cadre de l'apprentissage supervisé.

Supervised Growing Cell Structure est un modèle d'auto-organisation neuronale pour l'apprentissage supervisé basé sur *Growing Cell Structure*. Il propose une extension incrémentale des réseaux de type *Radial Basis Function* (voir sous-section 7.2.1) en réalisant un apprentissage simultané du nombre et de la répartition des champs récepteurs ainsi que des paramètres w_i de combinaison. Les champs récepteurs sont bien entendu associés aux cellules apprises. Le centre de la gaussienne correspond au vecteur poids w de la cellule. Le paramètre σ (rayon du champ) est fixé à la valeur moyenne des distances séparant la cellule de ses voisines immédiates. L'ajout d'une nouvelle cellule est toujours motivé par la valeur prise par le compteur. Mais au lieu d'incrémenter le compteur de 1 lorsque la cellule est activeΓon lui ajoute l'erreur quadratique commise par le système en ce point. La répartition des cellules ne sera plus guidée par la distribution des entrées mais par l'erreur de sortieΓmettant plus de cellules en jeu là où l'erreur

est plus grande.

Ce dernier point est très important et nous allons y revenir un instant. Apprendre la densité de probabilité de présentation des exepmles peut être un avantage dans certaines applications mais représente un handicap dans le cadre de notre problèmeΓcomme l'ont remarqué par exemple Pomerleau et Kröse [134Γ 107]. L'apprentissage supervisé se déroule classiquement de la manière suivante :

- Un opérateur humain téléopère le robot afin de lui faire effectuer la tâche désirée. L'ensemble des couples de points *(entrée, sortie)* générés durant cette phase est stocké.

- Les couples mémorisés sont présentés à l'approximateur retenu afin de créer une fonction satisfaisant les exemples.

L'opérateur humain étant un expertΓla séquence enregistrée contient une grande proportion de situations *normales* et très peu de situations *anormales* (obstacle très proche par exemple). Un réseau apprenant la distribution des données d'entrée aura tendance à allouer une grande partie de ses ressources pour les cas standards (le robot par exemple poursuit sa route loin de tout obstacle) et très peu ou pas de ressources pour les situations d'exceptionΓen général dangereusesΓoù la réponse du systèmeΓque l'on souhaiterait rapide et correcteΓa une très grande importance.

L'approche Supervised Gowing Cell Structure propose un apprentissage supervisé d'une fonction. Elle impose néanmoins le choix d'une dimension (appelée dimension inhérente) pour l'espace des cellules et dont la détermination peut nécessiter la réalisation de plusieurs essais. Cette dimension peut être vue comme une restriction dans le choix des fonctions pouvant être approchées.

Fuzzy Artmap

L'approche *Fuzzy Artmap* a été conçue par Carpenter et Grossberg [33] pour permettre l'apprentissage incrémental d'une fonction multi-dimensionnelle à partir d'exemples présentés. Elle s'appuie sur l'utilisation conjointe de deux réseaux de type *Fuzzy ART*.

L'approche *Fuzzy ART* ou *ART 2* [72] est la variante réelle du réseau binaire *ART* [32] mettant en jeu des opérateurs inspirés de ceux de la logique floue. Le réseau est constitué de trois couches (figure 7.5).

Les entrées x sont des vecteurs de dimension n dont les composantes sont comprises entre 0 et 1. Ces vecteurs sont tout d'abord normalisés grâce à un codage complémentaire avant d'être présentés à la couche d'entrée :

$$I = (x, x^c) \;\; \forall i \in \{1, \ldots, n\} \; x_i^c = 1 - x_i$$

La couche de sortie est constituée d'un ensemble de neurones y_j associés à un vecteur poids w_j. Ces neurones correspondent aux différentes classes créées

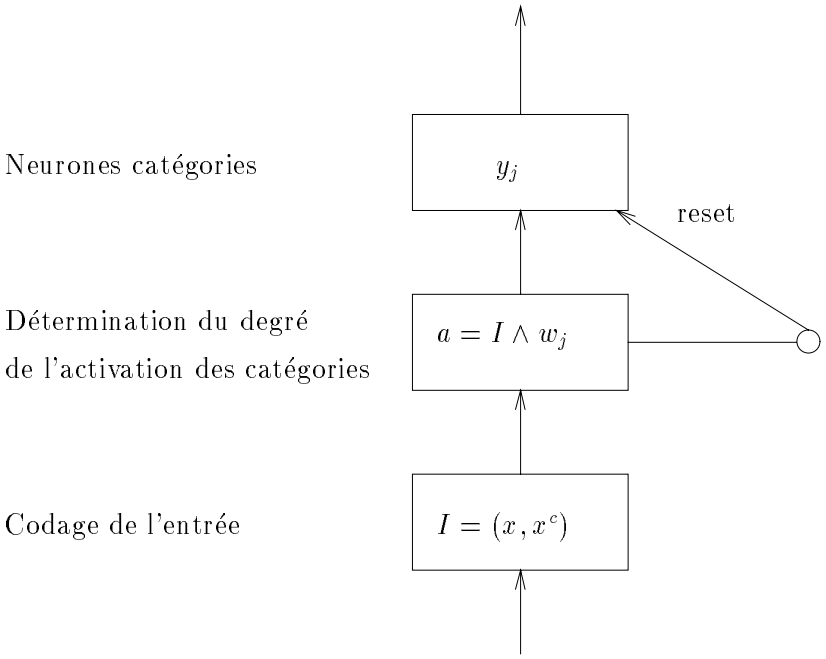


FIG. 7.5 - : *Structure du réseau Fuzzy Art*

à partir des données d'entrée. Lors de la présentation d'un nouvel exemple, la classe y_j activée est choisie selon le principe du *winner takes all* en appliquant l'opérateur ET de la logique floue ($(a \wedge b) = \min(a, b)$) . Si plusieurs neurones ont un degré T_k égal, le neurone dont l'indice est le plus faible est le neurone gagnant.

$$
\begin{aligned}
&T_k(I) = \tfrac{I \wedge w_k}{\alpha + |w_j|} \\
&T_j = \max_{j \in [1,n]}(T_k) \\
&activation(y_k) = \begin{cases} 1 & \text{si } k = j \\ 0 & \text{sinon} \end{cases}
\end{aligned}
$$

L'apprentissage du réseau est réglé par l'activité $a = I \wedge w_j$ de la couche cachée. Ce terme a indique la similarité de l'entrée x avec la classe y_j activée. Plus a est proche de 1 et plus x est semblable au représentant w_j de la classe et plus x est adapté à la classe. L'apprentissage s'effectue en ajustant le poids w_j dans le sens de la nouvelle entrée :

$$w_j^{new} = \beta(I \wedge x_j^{old}) + (1 - \beta) w_j^{old}$$

Si l'activité a est inférieure à un seuil fixé ρ appelé *vigilance*, on considère qu'aucune classe n'est adaptée à la nouvelle entrée. Une nouvelle classe est créée, centrée sur x. Ce paramètre ρ permet de régler le degré de généralité des classes et donc la finesse de description de l'espace de points.

Le réseau *Fuzzy Artmap* est constitué de deux réseaux *Art* F_{input} et F_{output} reliés par une mémoire associative. Ces deux réseaux ont pour rôle de réaliser une classification indépendante des points présents dans l'espace d'entrée et dans l'espace de sortie. La mémoire associative réalise quant à elle le lien entre les catégories d'entrée et les catégories de sortie.

Remarque : La création d'une nouvelle catégorie pour le réseau Art F_{output} n'est motivée que par les données de sortie. La création d'une nouvelle catégorie pour le réseau F_{input} est motivée soit par les données d'entrée, soit par un contre-exemple. Un contre exemple est obtenu lorsque l'on présente deux couples de points (x, y) et (x_1, y_1) avec :

- x et x_1 classés par F_{input} dans la même catégorie.
- y et y_1 classés par F_{output} dans deux catégories différentes.

Le réseau F_{input} crée alors une nouvelle classe centrée sur x_1 afin de prendre en compte ce cas particulier. Un réseau de type *Artmap* va apprendre en essayant de généraliser le plus possible les relations entre entrées et sorties (cette généralisation étant fixée par le degré de vigilance). Lors de l'apparition d'un cas inconnu ou d'un contre-exemple, ce dernier signifiant que la relation est trop générale, le système va créer une nouvelle association pour traiter ce cas particulier et donc raffiner sa connaissance. Ce cas particulier sera alors lui aussi susceptible d'être généralisé et ainsi de suite.

Un système de type *Artmap* apprend en une seule présentation d'exemples et de manière totalement incrémentale. De plus, en mode exploitation, le réseau est capable d'indiquer s'il ne connaît pas la réponse pour une situation particulière (le vecteur x d'entrée n'active aucune catégorie de F_{input} suffisamment) ce qui est un atout de sécurité en robotique mobile (voir remarque section 5.3.5). Enfin, un réseau de type Fuzzy Artmap n'apprend pas la distribution de probabilité des exemples, ce qui est recommandé pour l'application qui nous intéresse (voir remarque lors de la présentation de l'approche *Supervised Growing Cell Structure*).

Toutes ces propriétés font de Fuzzy Artmap un système potentiellement très intéressant pour un système robotique. Nous avons néanmoins choisi de nous tourner vers les systèmes de type *Gal* pour des raisons que nous allons préciser lors de la prochaine sous-section.

Grow And Learn (GAL)

L'approche *Grow And Learn* a été développée par Alpaydin [5] afin de réaliser un apprentissage supervisé incrémental de catégories. Elle peut également être utilisée afin d'apprendre une fonction en réalisant une approximation continue par morceau.

La structure du réseau est donnée figure 7.6. La première couche contient les neurones d'entrée. La deuxième couche est formée par les neurones exemples.

Chacun de ces neurones est caractérisé par un vecteur w_i représentant de la région de l'espace d'entrée associée. Ces neurones i sont finalement connectés à une ou plusieurs classes j par un lien $T_{i,j}$:

$$T_{i,j} = \begin{cases} 1 & \text{si le neurone exemple } i \text{ est connecté à la classe } j \\ 0 & \text{sinon} \end{cases}$$

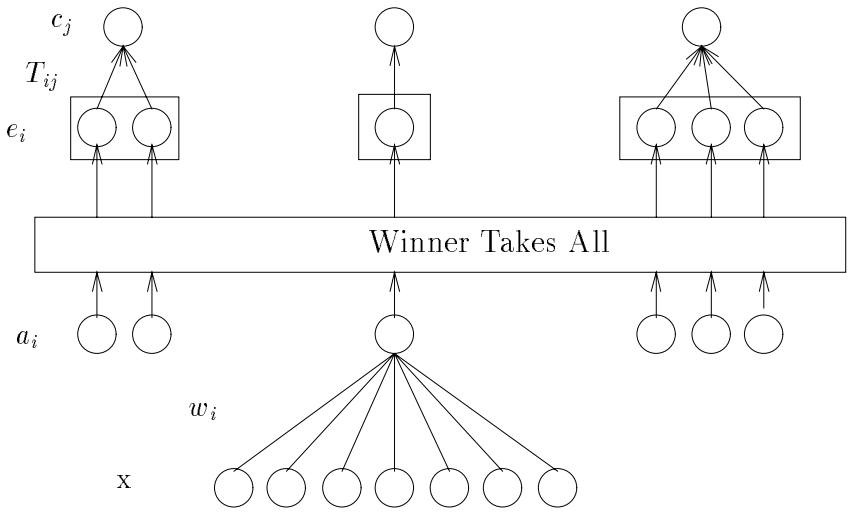


FIG. 7.6 - : *Structure d'un réseau de type* GAL

Le calcul de la sortie du réseau pour une entrée x donnée est réalisé de la manière suivante : on détermine tout d'abord le neurone exemple le plus proche de x par une mesure de distance de type Euclidienne. On détermine ensuite la classe associée à ce neurone. La sélection du neurone le plus proche implémente une non-linéarité de type *winner takes all*. Ces deux étapes sont résumées par les équations ci-dessous :

$$\begin{aligned} &\forall i \in \{1, \ldots, N\}\ a_i = \|x - w_i\| \\ &e_i = \begin{cases} 1 & \text{si } a_i = \min_i(a_i) \\ 0 & \text{sinon} \end{cases} \\ &c_j = \textstyle\sum_i e_i T_{i,j} \end{aligned} \tag{7.5}$$

L'algorithme de sélection du neurone exemple le plus proche réalise une subdivision de l'espace d'entrée en cellules de Voronoï bornées par des hyperplans. La zone de sélection d'une classe de sortie j correspond à l'union des zones d'activation des neurones exemples associés à j et peut donc prendre n'importe qu'elle forme. Cette forme sera approchée par un ensemble d'hyperplan.

L'apprentissage pour un réseau GAL consiste à déterminer d'une part les neurones exemples et d'autre part les liaisons entre ces neurones et les classes de sortie. Le principe est le suivant : on présente un nouvel exemple (x, c_j) au réseau. La sortie c associée à x est calculée grâce aux équations 7.5. Si $c = c_j$Γla classe obtenue est correcte. Aucune adaptation n'est nécessaire. Si $c \neq c_j$Γun nouveau neurone exemple k est créé et initialisé à x. Ce neurone est ensuite associé à la classe c_j :

$$w_k = x$$

$$\forall i \; T_{ki} = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{sinon} \end{cases}$$

L'apprentissage est donc réalisé en enregistrant les exemples contredisant la sortie courante. Par conséquentΓle réseau va stocker les exemples situés à proximité de la frontière séparant le choix d'une classe i de celui d'une classe j. Suivant l'ordre de présentation des donnéesΓle nombre de neurones exemples stockés peut donc fortement varier. Dans l'exemple ci-dessous (figure 7.7)Γle point A et le point C associés à deux classes de sortie différentes sont présentés. Le point BΓassocié à la même sortie que A est ensuite présenté. B étant plus près de C que de AΓon obtient une contradiction et B devient un neurone exemple. A étant dans la zone d'activation de B et étant associé à la même sortie devient inutile pour le réseau et n'aurait pas été créé s'il avait été présenté après B.

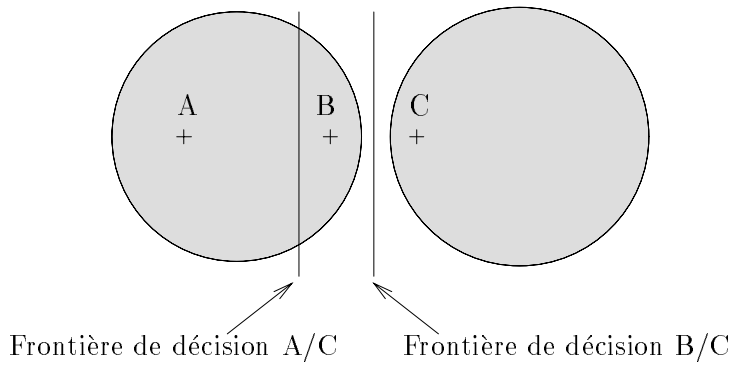


FIG. 7.7 - : *Création de neurones inutiles. La présentation des exemples A, B, C conduit à la création de trois neurones exemples. La présentation de B, C, A n'en crée que* 2

Afin de détecter ces neurones inutilesΓle système a recours à une phase de raffinement hors-ligne. Un neurone i est choisi au hasard. On détermine son voisin le plus proche et on compare leurs classes respectives. Si elles sont identiquesΓon efface le neurone i. Cette stratégie est susceptible de faire croître l'erreur de classification car la zone d'activation de i n'est pas forcément totalement incluse

dans la zone d'activation de son voisin. Mais elle conduit à une simplification du système.

L'apprentissage tel qu'il vient d'être décrit ne présente pas une grande résistance au bruit dans les entrées ou aux erreurs du superviseur (fournissant la sortie associée à une entrée). Ce problème peut être contourné en adaptant les neurones exemples proches de la frontière de décision par un processus de type LVQ[4]. Lorsque le neurone i le plus proche de l'entrée x et son plus proche voisin appartiennent à deux classes différentes, on adapte le poids de i en direction de x :

$$\Delta w_i = \alpha(t)(x - w_i)$$

Le facteur d'apprentissage α est diminué après chaque adaptation de manière à stabiliser le système. Cette adaptation réduit l'influence du bruit sur les performances du système. La comparaison de ce réseau avec un réseau de type Fuzzy Artmap conduit à plusieurs remarques :

- Les deux systèmes apprennent sur présentation d'un contre-exemple. Alors que le réseau Artmap ne crée qu'une association limitée à ce seul exemple, GAL va mettre en place un neurone exemple valide sur une zone allant jusqu'à ces voisins les plus proches. Artmap adopte en quelque sorte une approche plus sûre en se limitant à ce qui est acquis (l'exemple) et en ne faisant aucune hypothèse sur l'étendu de celui-ci. GAL adopte en revanche une stratégie plus risquée en généralisant immédiatement. Si cette généralisation s'avère correcte, l'apprentissage est alors plus rapide. Ce phénomène de généralisation conduit à la seconde remarque.

- GAL peut fournir une réponse quel que soit le point de l'espace d'entrée sans que l'on soit en mesure de savoir si cette réponse est due à une généralisation " abusive " ou à une situation déjà rencontrée. Le réseau n'est pas en mesure, comme dans le cas de ARTMAP, de reconnaître son incompétence à juger une situation. Il existe néanmoins dans le cas de GAL un mécanisme optionnel de rejet, rejetant une sortie si l'entrée est située trop près d'une frontière de décision (donnée par un seuil fixé par l'utilisateur). Mais ce mécanisme ne résout pas le problème de " sur-généralisation ".

- Un réseau ARTMAP nécessite pour sa mise en œuvre le choix de 6 paramètres ($\rho_{input}, \rho_{output}, \beta_{input}, \beta_{output}, \alpha_{input}, \alpha_{output}$). Un réseau GAL en revanche, dans sa forme élémentaire (pas de rejets ni d'adaptation LVQ) est très simple à mettre en œuvre (algorithme simple et nombre limité d'opérations à chaque étape) et ne nécessite la détermination d'aucun paramètre. Pour une utilisation en approximation de fonction, il faut néanmoins noter que dans le cas d'un réseau ARTMAP, on présente directement le couple

4. Learning Vector Quantization

(x, y) au système. Le réseau f_{output} se charge alors de réaliser la discrétisation des sortiesΓcette discrétisation étant réglée par le facteur de vigilance. Dans le cas d'un réseau GALΓcette discrétisation doit être réalisée avant présentation des exemples. Les différents paramètres de F_{output}Γs'il ne sont pas directement présents dans le réseauΓle sont éventuellement dans le module de catégorisation des valeurs y de sortie.

– Si en phase d'apprentissage on présente directement le couple (x, y) au réseauΓen phase exploitationΓla regénération de y à partir de la catégorie activée est un problème délicat. On peut montrer [33] que les catégories créées par un réseau ART sont des hyper-rectangles dont le prototype associé est un vecteur de dimension $2n$. Les premières n coordonnées correspondent à un coin du rectangleΓles deuxièmes n coordonnées au coin opposé. Ces points sont situés en frontières de catégories et ne sont donc pas très représentatifs. Les catégories pouvant se recouvrir et ne sont donc pas convexes. Le choix du point central du rectangle ne constitue pas une solution correcte.

La facilité de mise en œuvre de GAL ainsi que sa plus grande facilité d'utilisation en approximation de fonction nous ont conduit à le préférer à une solution de type ARTMAP. On peut également noter que la structure du réseau est suffisamment ouverte pour pouvoirΓsi la " sur généralisation " pose un problèmeΓ rajouter un mécanisme similaire à la vigilanceΓprovoquant l'absence de réponse du réseau si la distance séparant l'entrée du neurone exemple est trop importante.

Avant de décrire le système que nous avons réaliséΓ nous allons présenter quelques applications de l'apprentissage supervisé à la robotiqueΓ reprenant quelques-uns des réseaux présentés précédemment.

7.3 Exemples d'applications en robotique

L'apprentissage supervisé fut une des premières formes de programmation des robots de manipulation (dans l'industrie automobile en particulier). L'opérateur fait exécuter le mouvement une première fois au robot en le téléopérant. L'ensemble des coordonnées codeurs des différentes articulations est stocké dans une table. Cette table peut être ensuite relue par le système de manière à reproduire le mouvement. Les tâches devant être exécutées par de tels robots sont des tâches répétitivesΓne faisant appel à aucune décision basée sur des mesures provenant de capteurs extéroceptifs. Aucune capacité de généralisation n'est nécessaire à un tel système ou l'apprentissage est limité à sa plus simple expression de stockage séquentiel de toutes les données présentées. Nous nous intéressons au cours de ce rapport à des robots possédant un degré d'autonomie supérieur.

Il existe dans le domaine de la robotique mobile autonome plusieurs approches

basées sur l'apprentissage supervisé que l'on peut classer selon le type d'architecture de réseau et donc de loi d'apprentissage utilisée.

7.3.1 Apprentissage par rétro-propagation du gradient

Les réseaux de neurones à propagation unilatérale sont parmi les réseaux les plus fréquemment utilisés dans les applications neuronales quel que soit la discipline. Dans le cadre de la navigation réactive nous allons nous intéresser à deux exemples de pilotage de voiture.

Le système ALVINN

Le système ALVINN[5] a été développé par Dean Pomerleau dans le cadre de la conduite automatique d'un véhicule de type voiture sur une route [134 136 135]. Le principe du système est de laisser un opérateur humain conduire le véhicule pendant quelques minutes. Durant cette phase d'entraînement on présente au réseau de neurones les images i provenant de la caméra située sur le véhicule et regardant la route. La réponse r générée par le système (direction du volant) est comparée à celle r' fournie par le conducteur. Si ces deux réponses diffèrent le réseau est entraîné pour l'association (i, r'). Une nouvelle image est ensuite acquise et le cycle continue. Le réseau utilisé est un réseau à propagation unilatérale comportant une couche cachée. La couche d'entrée est une rétine de 30×32 neurones connectés chacun à un pixel de l'image fournie par le module de pré-traitement chargé de réduire la dimension des images vidéo. La couche cachée contient 4 neurones. Ce nombre a été fixé après de nombreux essais. Finalement la couche de sortie est constituée de 30 neurones chacun d'entre eux étant associé à une direction particulière du volant.

L'utilisation du schéma d'apprentissage tel qu'il a été décrit présente deux problèmes majeurs :

- absence de situations dangereuses dans les données d'entrée.
- phénomène d'oubli du réseau lors de la présentation de séquences identiques.

Le premier problème a déjà été évoqué lors de la présentation de l'approche *Supervised Growing Cell Structure* (sous-section 7.2.2). Durant la phase d'entraînement le conducteur humain conduit la voiture " correctement " en restant bien au centre de sa voie. Il ne fournit pas d'exemples permettant au système de savoir comment réagir si la voiture n'est malencontreusement plus alignée avec le centre de la route. La solution consistant pour le conducteur à faire zigzaguer la voiture durant la phase d'entraînement n'est pas satisfaisante car tout d'abord cela présente un caractère dangereux. De plus il est nécessaire dans un tel cas de débrancher l'apprentissage lorsque le conducteur éloigne la voiture du centre

5. Autonomous Land Vehicle In a Neural Network

de la route pour le rebrancher lorsqu'il rétablit la situation (sinonΓle réseau va également apprendre à faire zigzaguer la voiture sans raisons apparentes). Dean Pomerleau propose pour résoudre ce problème de générer à partir de chaque image perçue l'image que la caméra verrait si l'orientation du véhicule était modifiée de θ degrés vers la gauche et vers la droite. Les commandes devant être générées pour ces deux nouvelles images sont ensuite fournies par une procédure de type *poursuite pure* [180]. Le système peut ainsi apprendre à réagir lorsque la voiture n'est plus dans l'alignement de la route.

Le second problème se manifeste par exemple lorsque la voiture suit une longue ligne droite. La présentation de situations répétitives peut amener le réseau à oublier ce qu'il a appris dans d'autres situationsΓcomme par exemple prendre un virage. Ce phénomène d'oubli a déjà été présenté sous-section 7.2.1 et en annexe C. Il est dû à la loi d'apprentissage utilisée ne présentant pas un caractère incrémental. Il est nécessaire pour contourner ce problème qu'à chaque cycle l'apprentissage ne soit pas uniquement basé sur la dernière image perçue. Pour résoudre ce problèmeΓchaque nouvelle association *(image, commande)* est stockée dans un buffer servant de base de données pour l'apprentissage. Le stockage de l'ensemble de ces couples étant impossible (coût mémoire et temps de calcul prohibitif)Γla taille du buffer est volontairement limitée à 200 entrées. De manière à ce que la généralisation soit correcteΓil est nécessaire que ces données soient représentatives de la tâche à accomplir. La suppression d'un ancien exemple lors de la parution d'un nouveau est donc un problème délicat. Plusieurs algorithmes ont été expérimentés :

- l'introduction d'un nouvel exemple dans le buffer entraîne la suppression de l'exemple stocké le plus vieux. Le buffer contient alors un historique des situations rencontrées récemment. Si les situations ne sont pas diversifiées (longue ligne droite)Γles images contenues dans le buffer ne le seront également pas et l'apprentissage ne sera pas correct.

- l'introduction d'un nouvel exemple entraîne la suppression d'un exemple choisi aléatoirement parmi les anciens. Ceci ne résout pas le problème de manque de diversité des exemples en cas de présence prolongée de situations semblables.

- on supprime l'exemple le mieux appris. L'inconvénient d'une telle approche est que si le conducteur fait une faute (mauvaise direction du volant)Γcet exemple ne pourra être éliminé de la base qu'à condition que le réseau est appris à reproduire cette fauteΓce qui n'est pas souhaitable.

- résoudre le problème précédent en supprimant l'exemple le mieux appris ainsi que l'exemple le moins bien appris. On risque cependant de supprimer un exemple non appris non pas parce qu'il est faux mais par ce qu'il correspond à une situation nouvelle.

- supprimer aléatoirement un exemple avec une probabilité plus forte pour l'exemple le mieux appris. Cette approche a donné de bons résultats.

- garder la diversité de la base en imposant une direction moyenne des exemples proche de 0 degré (ligne droite). Ainsi l'ajout d'un exemple demandant un virage à gauche entraînera la suppression de l'exemple demandant de tourner à droite tel que la nouvelle moyenne sera la plus proche de 0. Cette approche a donné également de bons résultats.

Les neurones situés sur la couche cachée d'un réseau à propagation unilatérale ont pour rôle de coder les aspects importants présents dans les données d'entrée permettant le calcul de la sortie. Afin de permettre la navigation dans de nombreuses situations perceptives différentes (navigation sur une route à une voie sur une route à plusieurs voies franchissement de carrefours ...) Pomerleau a développé la méthode *IRRE*[6] mettant en œuvre un ensemble de réseaux chacun étant spécialisé pour un type de route. Les réseaux utilisés sont similaires à ceux décrits précédemment mais possède en plus sur la couche de sortie un ensemble de cellules similaires à la rétine d'entrée (voir figure 7.8).

Les réseaux sont entraînés de manière à générer la commande et à reconstruire l'image présentée en entrée. Une reconstruction correcte de l'image signifie que le réseau a été capable de la coder correctement et que cette image est donc dans sa zone de compétence. En utilisation les images d'entrée sont présentées en parallèle à l'ensemble des experts. L'erreur de reconstruction est utilisée afin de permettre l'arbitrage entre les différentes sorties proposées. Cette erreur permet également de réduire la vitesse du robot si aucun des réseaux n'est capable de coder l'image et donc si la situation ne correspond à aucune des situations pour lesquels ils sont prévus.

L'approche proposée par Pomerleau a permis d'obtenir des résultats très spectaculaires. Elle fait apparaître en revanche les problèmes posés par l'utilisation d'une approche non incrémentale permettant ainsi de justifier le choix de notre réseau.

Dans cette approche les données d'apprentissage sont fournies par un opérateur humain expert dans la tâche à réaliser. Nous allons maintenant présenter une approche où les données sont fournies par un autre programme.

Le système de Freisleben

Freisleben [61] se pose le problème de concevoir un contrôleur capable de conduire de manière réactive une voiture le long d'une piste de course non nécessairement connue au préalable. Le contrôleur dispose comme entrée de la vitesse du véhicule ainsi que d'informations de distances et d'angles sur les obstacles

6. Input Reconstruction Reliability Estimation

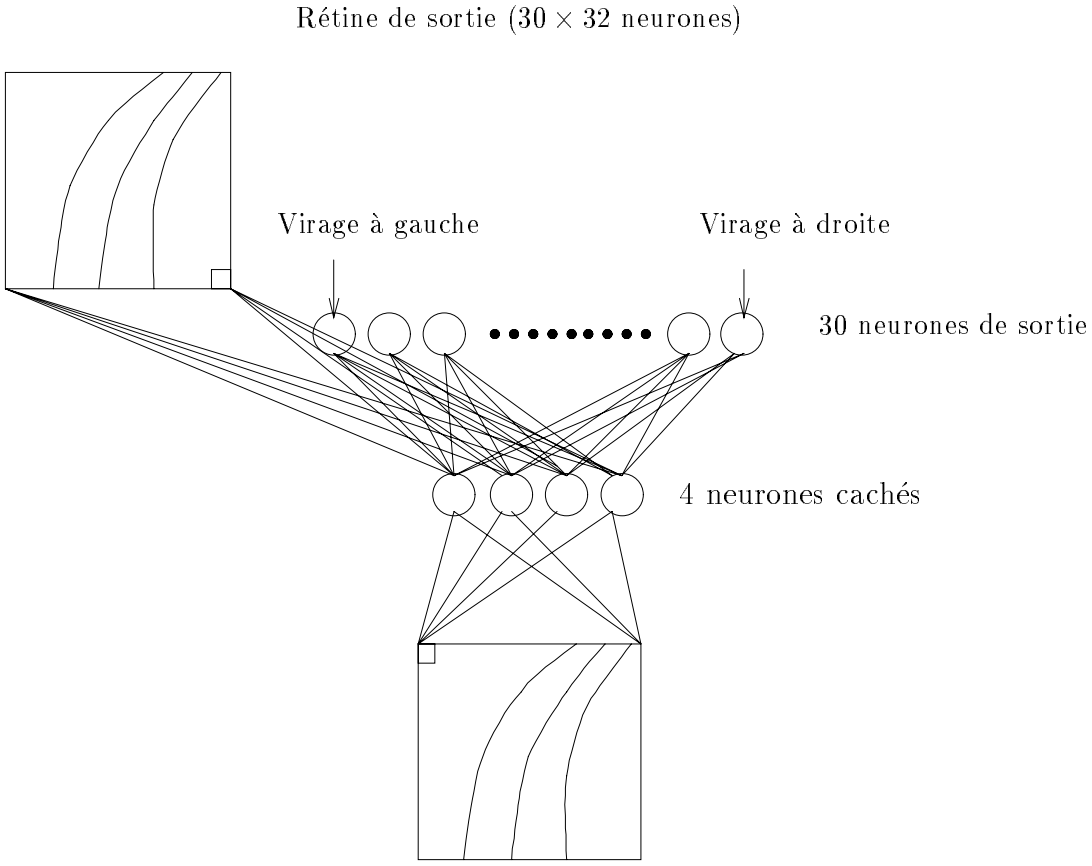


FIG. 7.8 - : *Architecture d'un réseau expert dans le cadre de l'approche* IRRE

proches (autres véhiculesΓbord de la route). Trois types de contrôleurs différents ont été expérimentés :

- un contrôleur flou.
- un réseau de neurones à propagation unilatérale entraîné à partir d'une base d'exemples fournie par un expert.
- un réseau de neurones à propagation unilatérale entraîné à partir d'une base d'exemples fournie par le contrôleur flou précédent.

Nous allons revenir sur le troisième point. Freisleben a conçu tout d'abord un contrôleur flou capable de conduire le véhicule sur une piste d'entraînement. Son but ensuite est de chercher à transférer la " compétence " de ce premier système à un réseau de neurones à propagation unilatérale. Une telle approche ne peut se justifier que si le nouveau contrôleur obtenu présente des performances supérieures au premierΓsoit en termes de temps de calculΓsoit en termes de compétence. L'apprentissage est réalisé en laissant le système flou conduire sur la piste d'entraînement et en notant l'ensemble des couples *(entrée, sortie)* générés. Le réseau apprend ensuite à reproduire l'ensemble de ces données.

Les performances respectives des trois contrôleurs (le contrôleur flou et les deux contrôleurs neuronaux) ont été ensuite comparées lors d'une série de tests. Chaque test consiste en une course entre deux véhicule équipés de deux contrôleurs différents. On note ensuite pour chaque véhicule le temps réaliséΓle nombre de courses gagnées ainsi que le nombre de collisions contre l'autre véhicule ou le bord de la route.

Les premières expériences ont été réalisées sur la piste d'entraînement. Les résultats obtenus sont alors assez similaires entre les trois solutions (même nombre de courses gagnées et perdues). Le réseau entraîné par un humain pilote la voiture plus rapidement mais cause plus d'accidents. Le contrôleur flou a causé un nombre faible d'accidents tandis que le réseau entraîné par ce contrôleur n'en a causé aucun.

Les expériences suivantes se sont déroulées sur une piste jamais rencontrée par aucun des contrôleurs. Le réseau entraîné par un humain a perdu toutes les courses et a causé un grand nombre d'accidents mettant en évidence la faible capacité de généralisation du système. Le contrôleur flou a causé beaucoup plus d'accidents que le réseau qu'il a entraîné et a perdu toutes les courses contre lui sauf une.

Ces deux séries d'expériences ont permis de mettre en évidence que les capacités de généralisation des réseaux sont susceptibles de générer des systèmes dont les performances sont supérieures à leur maître. Ce résultatΓbien que n'étant pas généralΓest encourageant quant à l'utilisation des approches neuronales. Dans le cas de cette approcheΓles réseaux utilisés sont des réseaux à modèle fixe. Le nombre de neurones sur la couche cachée a été déterminé par essais successifs.

Dans le cas de notre problème, l'utilisation d'un réseau à modèle non fixe nous permettra d'éviter ces différents essais.

Les approches basées sur les réseaux à propagation unilatérale et la rétro-propagation sont parmi les plus fréquemment rencontrées en robotique mobile (voir [101, 58] pour d'autres exemples). Nous allons maintenant nous intéresser à des approches ayant recours à une loi d'apprentissage basée sur des méthodes locales.

7.3.2 Apprentissage par méthodes locales

Comme nous l'avons précisé au cours de la sous-section 7.2.2, les méthodes locales partitionnent l'espace d'entrée en sous-régions et cherchent à approcher la fonction recherchée sur chacune de ces sous-régions. Nous allons nous intéresser plus particulièrement à l'approche proposée par Heikkonen [78] pouvant être décomposée en deux parties :

1. le robot apprend à se déplacer le long d'obstacles, tels que des murs, grâce à une série d'exemples présentés.

2. le robot apprend à rejoindre un but en évitant les obstacles grâce à un raffinement en-ligne de sa loi de commande.

La seconde partie s'inscrit dans un schéma d'apprentissage renforcé, objet du chapitre 8. Nous nous intéresserons donc plus particulièrement à la première partie. Le but est d'apprendre à réaliser un comportement particulier de la navigation dans un couloir après présentation d'exemples par un opérateur humain. Parmi ces comportements, nous pouvons citer *aller tout droit*, *tourner à gauche à une intersection* et *tourner à droite à une intersection*. Le comportement *tourner à gauche* par exemple est appris pour une intersection particulière mais doit pouvoir être utilisé pour tous les types d'intersection à gauche. Il faut que le système soit en mesure de généraliser correctement. Heikkonen associe à chaque comportement un réseau de type Kohonen chargé de l'implémenter. Durant l'apprentissage, le maître montre différents exemples de mouvements correspondant au comportement concerné. Les mesures provenant des 8 capteurs ainsi que le changement de direction sont alors enregistrés sous forme de vecteurs de dimension 9 :

$$\vec{x_t} = (d_{t,1}, \ldots, d_{t,8}, a_t)$$

Cette liste de vecteurs est ensuite présentée au réseau de Kohonen dont le rôle est de répartir les éléments dans différentes classes de manière non supervisée. Remarque : la cellule gagnante est sélectionnée à partir des 8 premières coordonnées. Seules les informations perceptives sont utilisées. Par contre, dans la phase adaptation, la totalité des coordonnées sont mises à jour. De même, en mode utilisation, seules les données perceptives sont utilisées pour le choix de la

cellule. La commande associée à cette cellule ($9^{ème}$ coordonnée) est alors envoyée au robot.

Les expériences réalisées en simulation montrent une capacité de généralisation intéressante de la part du système. En revanche, le problème du choix du nombre de cellules dans le réseau de Kohonen n'a pas été évoqué. Ce nombre doit néanmoins être choisi en fonction de la tâche à accomplir.

Nous allons maintenant présenter au cours de la prochaine section le système que nous avons réalisé dans le cadre de l'apprentissage supervisé de comportements [143]. Nous présenterons tout d'abord les résultats obtenus lors d'une mise en relation directe des données capteurs brutes avec les commandes pour nous intéresser par la suite à la mise en place d'un prétraitement permettant de faciliter cet apprentissage.

7.4 Apprentissage direct de la loi de commande

Comme nous l'avons précisé au cours de la section 7.2, le réseau utilisé est un réseau de type *GAL*. Nous avons choisi de faire apprendre au robot la tâche de suivi d'un mur à partir des données provenant des capteurs ultrasons. Le robot ne comporte pas de buts à atteindre en terme de points de coordonnées (x, y) dans l'environnement. Cette tâche de suivi de mur est une tâche élémentaire d'évitement d'obstacles.

7.4.1 Acquisition des données

De même que pour les approches précédemment décrites dans ce rapport, le module de navigation résultat de l'apprentissage est situé entre les données capteurs et le contrôleur de véhicule (voir figure 2.2). Les commandes générées sont des commandes de type *move* et *turn*.

le processus utilisé pour l'apprentissage est le suivant :

1. l'utilisateur téléopère le véhicule le long d'un mur. Il dispose pour cela de quatre boutons permettant d'indiquer au véhicule d'augmenter ou de diminuer sa vitesse linéaire ainsi que sa vitesse angulaire (ordre *move* et *turn* spécifiant la vitesse). Les données capteurs enregistrées lors de la téléopération ainsi que les commandes générées sont stockées dans un fichier. De manière similaire à celle proposée par Pomerleau [134], on augmente la vitesse d'apprentissage en générant pour chaque situation la situation miroir (voir figure 7.9).

2. Le fichier de données est présenté hors-ligne au réseau GAL pour apprentissage.

3. Le réseau GAL remplace alors l'opérateur humain dans la boucle de commande. Les performances obtenues sont évaluées.

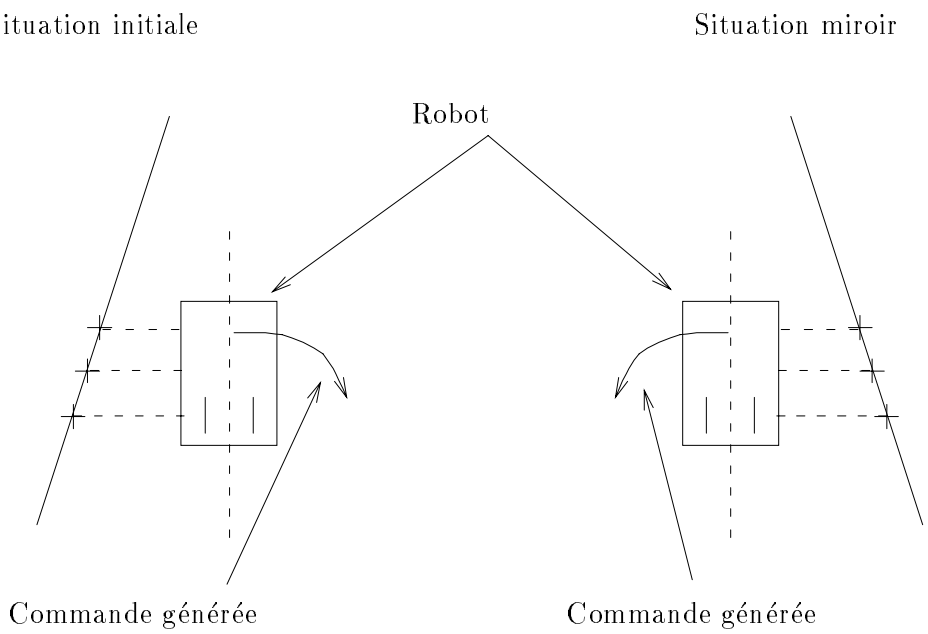


FIG. 7.9 - : *Situation miroir d'une situation perçue*

4. Si les performances sont jugées insuffisantesΓsi le robot n'estΓpar exempleΓ pas en mesure de franchir une situation particulièreΓl'utilisateur reprend le contrôleΓtéléopère le robot pour franchir cette situationΓcréant ainsi un nouveau fichier de données.

5. Ce nouveau fichier est présenté au réseau GAL courant. Comme il ne contient que les dernières données recueilliesΓla propriété d'incrémentalité du réseau est primordiale. Cette nouvelle phase d'apprentissage est destinée à raffiner le système courant et ne doit pas remettre en cause les capacités déjà acquises.

6. Les étapes 4 et 5 sont répétées jusqu'à obtenir un comportement du véhicule compatible avec la tâche qu'il doit accomplir.

Nous allons maintenant présenter les résultats obtenus. Ces expériences ont été réalisées en simulation.

7.4.2 Apprentissage incrémental

Le but de cette première expérience est de montrer sur un exemple l'incrémentalité du système. Elle a été réalisée sans la génération de situations miroires.

On apprend dans un premier temps au robot à tourner à gauche en lui présentant un coin convexe de 90 degrés. Le véhicule est téléopéré une seule fois

le long de ce coinΓgénérant ainsi un fichier de 183 associations *(données capteurs,commandes)*. Ce fichier est ensuite présenté au réseau GALΓprovoquant la création de 9 neurones exemples. La figure 7.10 représente la trajectoire suivie par le véhicule piloté par le réseau résultant.

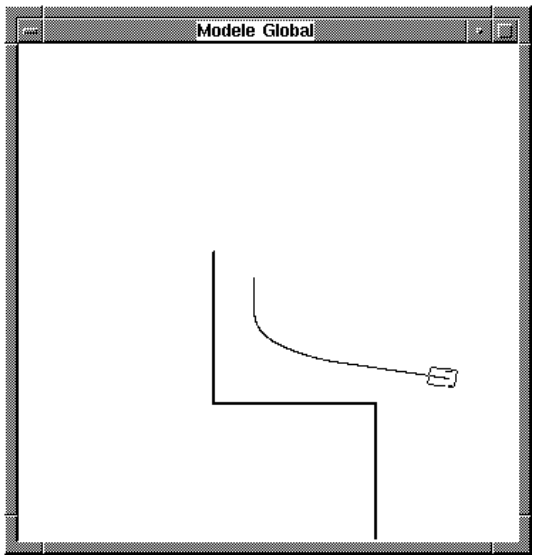


FIG. 7.10 - : *Le robot apprend à tourner à gauche*

Au cours d'une deuxième phaseΓon apprend au véhicule à tourner à droite en le téléopérant le long d'un coin concave de 90 degrés également. Cette nouvelle téléopération génère un fichier de 294 nouvelles situationsΓprésenté au réseau précédent. 7 nouveaux neurones sont créés pour un total de 16 neurones. L'apprentissage étant effectué hors-ligneΓle temps nécessaire n'est pas primordial. Mais à titre indicatifΓil ne faut au réseau GAL que 0.5 seconde au total sur une Sparc 10 pour apprendre le premier fichier de pointsΓpuis le second fichier. La trajectoire suivie par le véhicule après le second apprentissage est montrée 7.11. Comme on peut le constaterΓle robot est toujours capable d'effectuer de manière similaire le virage à gauche. Ses capacités n'ont pas été altérées par l'apprentissage du virage à droite. Le système est bien incrémental.

Nous allons illustré l'intérêt de l'apprentissage incrémental et du raffinement successif du comportement au cours de cette seconde expérience. La tâche a accomplir est conçue de telle sorte que la difficulté augmente au fur et à mesure que le robot avance. Le véhicule est dans un premier temps téléopéré le long du premier triangle comme indiqué figure 7.12. La trajectoire est constituée de 401 échantillonsΓgénérant un réseau GAL comportant 40 neurones exemples. La figure 7.12 montre les performances du robot sur la totalité du mur après ce premier apprentissage.

La généralisation des données apprises permet au véhicule de suivre le mur au-delà de l'intervalle d'entraînement. Le résultat est néanmoins jugé insuffisant

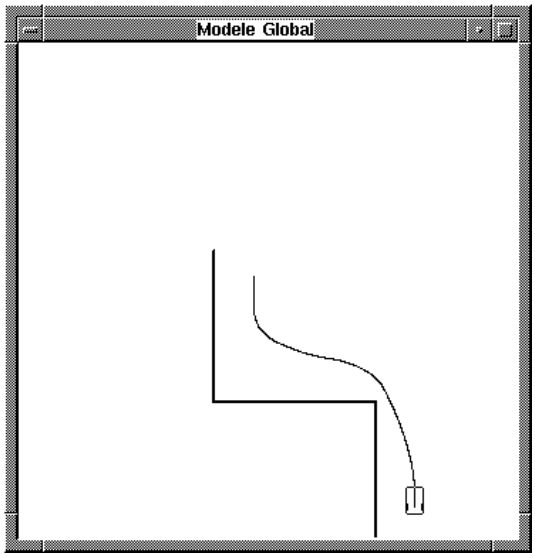


FIG. 7.11 - : *Le robot apprend à tourner à droite. Ses capacités à tourner à gauche ne sont pas altérées: l'apprentissage est incrémental*

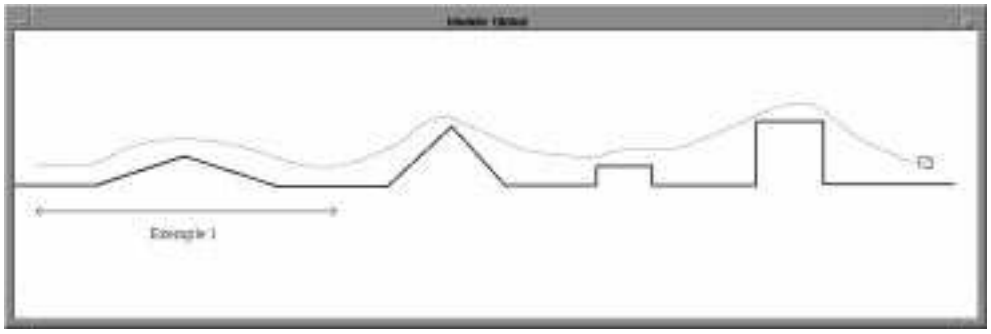


FIG. 7.12 - : *Comportement du robot après apprentissage du premier exemple*

entre les deux obstacles rectangulaires. Le robot est alors téléopéré le long de cette portion de mur Γ générant 249 nouveaux exemples. Le réseau GAL passe alors de 40 à 66 neurones. Le comportement résultant est montré figure 7.13.

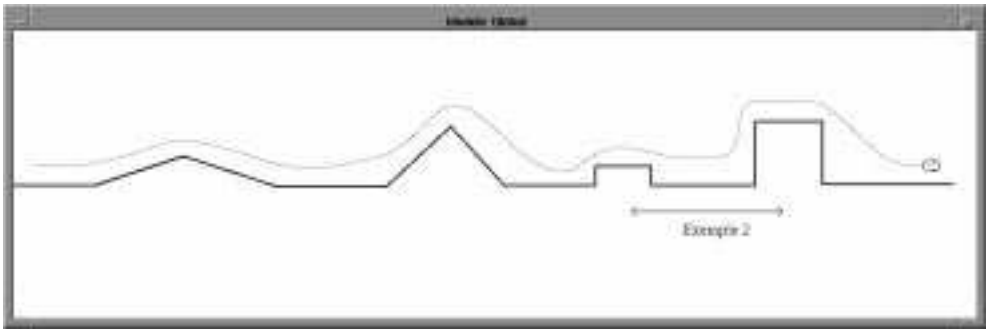


FIG. 7.13 - : *Comportement du robot après apprentissage du deuxième exemple*

Après apprentissage du second exemple Γ le robot est toujours capable de se déplacer le long de la totalité du mur et de manière plus satisfaisante. Ceci montre une fois de plus l'incrémentalité du système.

Il faut remarquer que cette seconde expérience est plus complexe que la première. Dans le cas de la première expérience Γ le premier groupe d'exemples (coin concave) et le second groupe d'exemples (coin convexe) délimitent deux zones disjointes de l'espace perceptif. L'apprentissage du second groupe d'exemples ne provoque donc aucune modification des neurones mis en place par le premier groupe. Dans le cas de la seconde expérience Γ les zones perceptives des deux groupes d'exemples se recouvrent. Cela signifie que les exemples du second groupe sont susceptibles de modifier le comportement mis en place par le premier. On peut d'ailleurs constater ce phénomène figure 7.13 où le franchissement du second triangle qui Γ tout en restant correct Γ ne s'opère plus de la même manière que figure 7.12. Ce recouvrement de situations perceptives facilite la généralisation des commandes apprises et permet au robot après les premiers exemples de franchir les deux obstacles rectangulaires. Mais ce recouvrement s'avère en revanche être nuisible pour l'incrémentalité si deux situations présentes dans le premier et le second groupe d'exemples sont perçues de manière identique et appellent des réponses différentes (aliasing perceptif). L'apprentissage du second groupe modifiera de manière erronée les connaissances apportées par le premier et le robot perdra des compétences déjà acquises. Le rôle d'un pré-traitement pourrait être de faire apparaître dans l'espace perceptif les zones concernées par le premier et le second groupe d'exemples comme deux zones disjointes. On obtiendrait alors bien un apprentissage incrémental Γ mais au détriment de la généralisation qui disparaîtrait. Un compromis doit être trouvé.

Nous allons maintenant nous intéresser aux capacités de généralisation du système.

7.4.3 Généralisation

Nous allons au cours de cette troisième expérience apprendre au robot à tourner à gauche lorsqu'il rencontre une porte dans un couloir. La téléopération fournie 341 points créant un réseau GAL de 12 neurones exemples. Le comportement du robot face à une situation identique est montré dans la reproduction d'écran située en haut à gauche figure 7.14. On évalue ensuite le comportement du robot en lui présentant un environnement comportant un carrefour (en haut à droite)Γ ainsi que deux environnements comportant des embranchements avec un angle différent de 90 degrés (en bas à gauche et à droite). Ces expériences sont similaires à celles réalisées par heikkonen [78]. Elles montrent une capacité de généralisation de la part du réseau GAL aussi importante que celle obtenue avec un réseau de Kohonen.

L'analyse du réseau obtenu fait apparaître la présence d'un neurone spécialisé dans la détection d'un espace libre sur la gauche. La figure 7.15 représente l'exemple stocké par ce neurone ainsi qu'une situation perceptive ayant permis de le déclencher.

La généralisation est possible dans les environnements où les données sensorielles sont capables de déclencher le neurone associé à l'embranchement. La mise en place d'un système d'extraction d'indices particulièrement sensible aux embranchements favorisera donc la généralisation.

7.4.4 Conclusion

Les différentes expériences réalisées mettent en évidence un apprentissage incrémental possédant d'intéressantes facultés de généralisation. Cet apprentissage peut être perturbé par le phénomène d'aliasing perceptifΓfaisant percevoir de manière très similaire ou identique deux situations devant être traitées différemment. Cet aliasing peut venir perturber les compétences précédemment acquises par le robot et donc ralentir son apprentissage. Ce phénomène peut être réduit par la mise en place d'un processus de filtrage des données d'entrée dont le rôle est de faire ressortir les caractéristiques " importantes " de ces données facilitant ainsi la discrimination de situations différentes.

7.5 Pré-traitement des données sensorielles

Le pré-traitement des données sensorielles a pour but de transformer les données d'entrée afin de faciliter le travail du module d'apprentissage. Ce travail peut être facilité :

- en réduisant la dimension de l'espace d'entrée. Le nombre d'exemples nécessaires pour l'apprentissage croît de manière exponentielle avec la dimension de l'espace d'entrée [146]

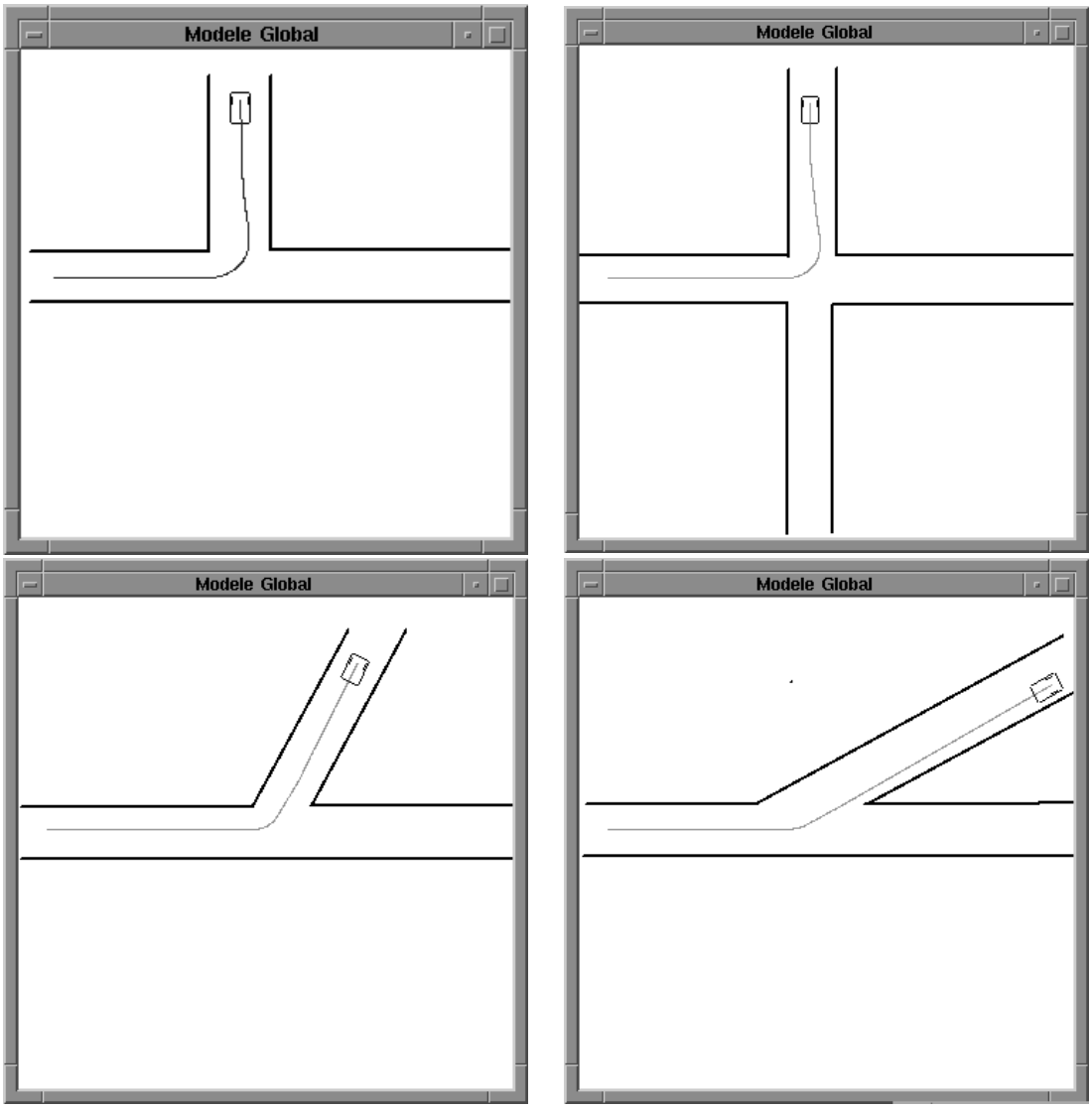


FIG. 7.14 - : *Généralisation du comportement de franchissement d'une porte située à gauche. La reproduction d'écran située en haut à gauche représente l'environnement utilisé pour l'apprentissage*

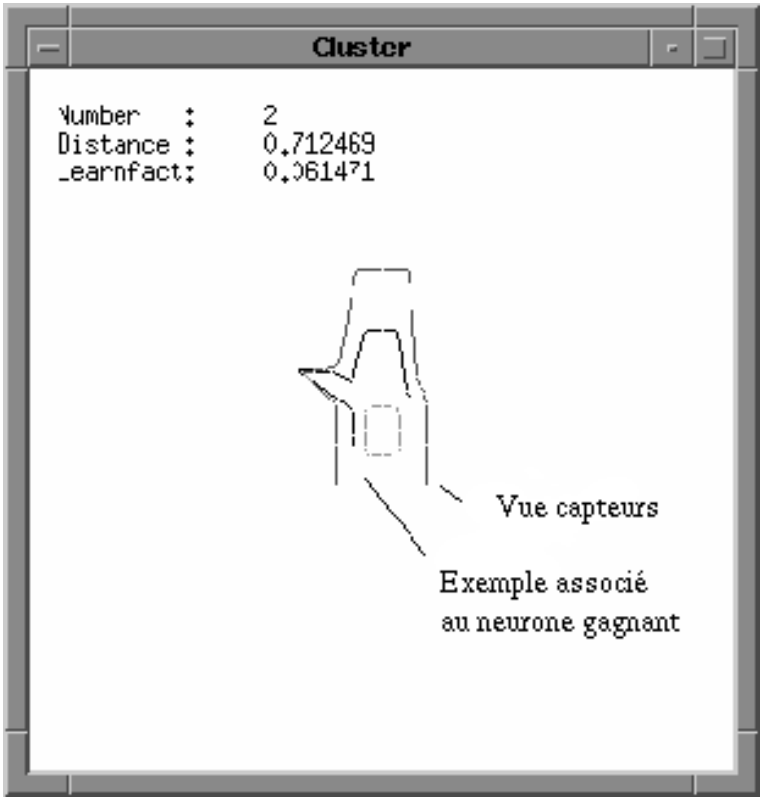


FIG. 7.15 - : *Situation perceptive et exemple stocké par le neurone gagnant lors du passage d'un embranchement*

– en réduisant l'aliasing perceptifΓprovoquant l'apparition d'exemples contradictoires.

Le phénomène d'aliasing perceptif peut avoir deux causes distinctes :

– les capteurs utilisés possèdent une résolution insuffisante ne permettant pas ou mal de discriminer deux situations différentes nécessitant deux réactions différentes (voir figure 7.16).

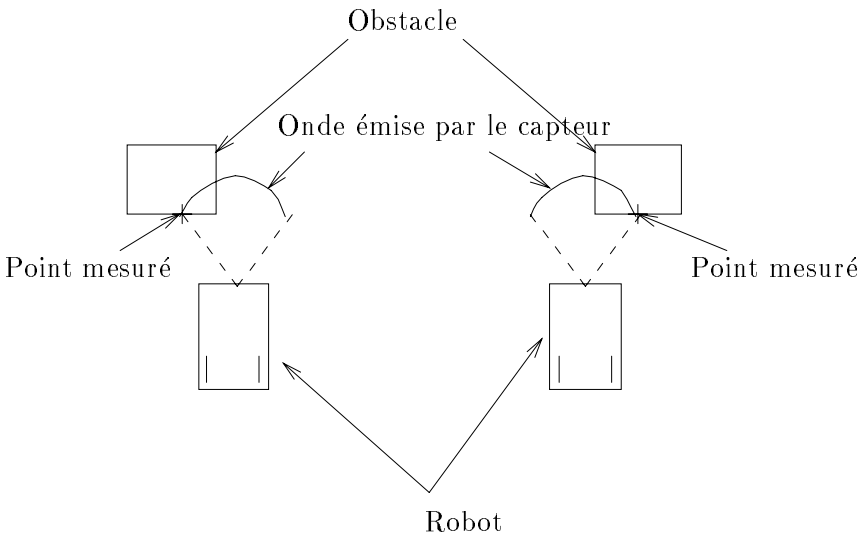


FIG. 7.16 - : *La figure de gauche et la figure de droite représentent deux situations différentes perçues de manière semblable*

– l'utilisation de données aberrantes générées par les capteurs (réflexions multiples par exemple pour les capteurs ultrasons). Ce problème est illustré figure 7.17.

Nous avons vu au cours de la sous-section 2.3.2 diverses méthodes permettant de filtrer les données aberrantes et de réduire l'incertitude sur les mesures. Ces méthodes peuvent être basées sur une modélisation paramétrique ou sous forme de grille de l'environnement. Nous nous intéresserons ici plus particulièrement au problème d'extraction d'indices permettant à la fois une réduction de la dimension des entrées et une meilleure séparation des différentes situations.

L'extraction d'indice a pour but de fournir une représentation des données dans un espace de dimension moindre et de fournir l'information de manière plus dense et plus robuste. On utilise par exemple en vision par ordinateur des indices tels que les points caractéristiquesΓles contours ou les régions. Le choix de ces indices est en général guidé par la tâche à accomplir ou par les types de données que l'on est susceptible de rencontrer (par exempleΓextraction de lignes

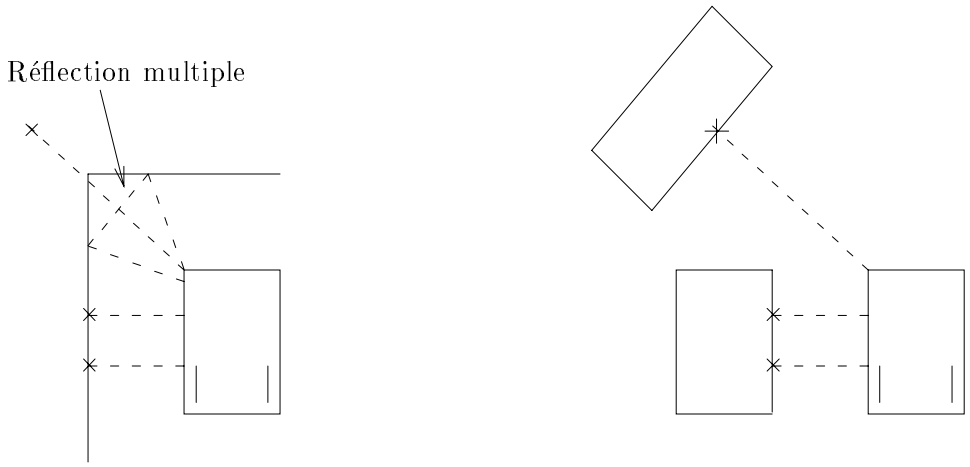


FIG. 7.17 - : *Similarité de situations perçues due à une reflexion multiple*

verticales dans une image pour un robot naviguant à l'intérieur d'un bâtiment). De manière à rester le plus général possible et essayer d'obtenir ainsi un codage fiable dans un grand nombre de situations différentesΓnous nous sommes tournés vers une approche réduisant la dimension des données d'entrée tout en conservant le maximum d'informations.

Nous allons tout d'abord présenter le principe de la réduction de dimension par extraction des vecteurs propres avant de nous intéresser à l'application réalisée dans le cadre de notre problème.

7.5.1 Réduction de dimension par extraction de vecteurs propres

La réduction par extraction de vecteurs propresΓappelée aussi analyse en composantes principalesΓcherche à fournir une représentation y de dimension inférieure d'un vecteur de donnée x en conservant le maximum d'informations. Notons que cette propriété de conservation d'entropie est supposée être un principe fondamental du traitement d'informations dans le cortex visuel humain [113].

Les variables $\{X_1, \ldots, X_n\}$ d'entrée sont supposées être issues d'un processus aléatoire statique. Le but est de trouver une description non redondante de ce processus $\{Y_1, \ldots, Y_m\}$ de plus petite dimension. Les variables Y_i sont appelées les *facteurs principaux*.

La figure 7.18 représente un ensemble d'observations d'un processus aléatoire décrit par les variables X_1 et X_2. La variable Y_1Γobtenue par combinaison linéaire de X_1 et X_2 fournit un codage non redondant de ce processus.

Dans le cas généralΓon essaie de décrire les observations du processus dans

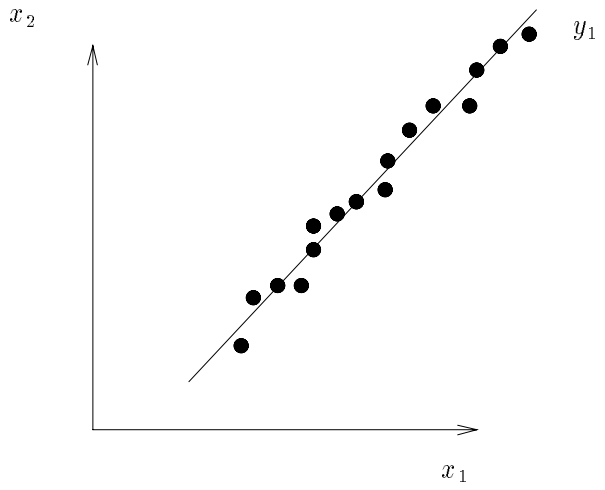


FIG. 7.18 - : *Distribution de points dans un espace à 2 dimensions et transformation dans un espace à 1 dimension.*

un espace de dimension plus petite en appliquant une transformation f :

$$f : \Re^n \mapsto \Re^m$$

Cette transformation f peut être linéaire ou non linéaire.

La réduction linéaire

Si on suppose qu'il existe une relation linéaire entre un vecteur d'observation $X =< x1, \ldots, x_n >$ et les m facteurs principaux y_iΓon peut alors écrire :

$$\forall i \in \{1, \ldots, m\}\ y_i = {A_i}^t (X - M)$$

où A_i est un vecteur de dimension n et M le vecteur moyen des observations. Connaissant la forme réduite $Y =< y_1, \ldots, y_m >$Γon peut alors reconstruire l'entrée x en appliquant la transformation inverse :

$$\hat{X} - M = \sum_{i=1}^{m} y_i A_i + \sum_{i=m+1}^{n} b_i A_i$$

L'erreur de reconstruction est minimale si et seulement si les vecteurs A_i correspondent au vecteurs propres dans l'ordre des valeurs propres décroissantes de la matrice de covariance C (voir équation 7.6) et que les composantes supprimées par la réduction de dimension soient remplacées par des constantes b_iΓmoyenne de y_i sur l'ensemble des observations [65]. Soient $\{X^1, \ldots, X^p\}$ un ensemble de

p vecteurs observation et M le vecteur moyen de ces vecteurs. La matrice de covariance C est alors définie par :

$$C = (X^1 - M, \ldots, X^p - M)(X^1 - M, \ldots, X^p - M)^t \tag{7.6}$$

Si la dimension de la matrice de covariance devient trop élevéeΓun calcul direct de diagonalisation peut être délicat. Il est alors possible d'avoir recours à des méthodes neuronales. Par exempleΓun réseau à une seule couche possédant n neurones d'entrée et m neurones de sortie et dont les poids des connections sont mis à jour par la règle de Oja voit ses poids converger vers les coordonnées des m premiers vecteurs propres ordonnés selon les valeurs propres [155]. De mêmeΓBaldi et Hornik [16] ont démontré qu'un réseau auto-associatif (les exemples présentés ont pour forme (x, x)) possédant moins de neurones sur sa couche cachée que sur sa couche d'entrée apprend une représentation proportionnelle aux vecteurs propres sur les connections entre la couche d'entrée et la couche cachée.

La méthode de décomposition en vecteurs propres a donné lieu à de nombreuses applications. Elle a été utilisée par exemple pour la surveillance d'un réacteur chimique [125]. 18 mesures du processus sont projetées sur un espace de dimension 2 et classées en catégories correspondant au mode de fonctionnement du réacteur. Cette approche a également été utilisée avec succès comme pré-traitement pour la reconnaissance de visages [177].

Le critère de conservation d'informations ne garantit pas l'extraction d'indices optimaux pour la tâche de classification [65].

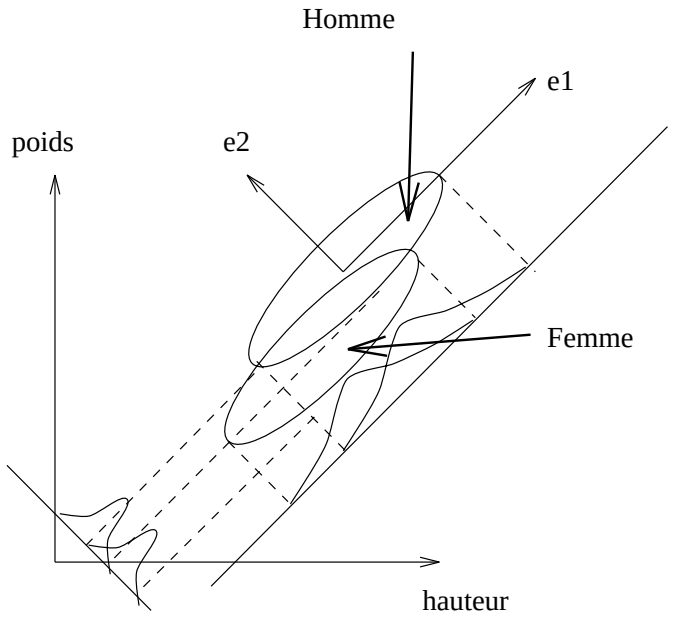


FIG. 7.19 - : *Distribution des variables aléatoires* poids *et* taille *pour les classes* homme *et* femme

La figure 7.19 représente la distribution de deux variables aléatoires *poids* et *taille* pour les classes *homme* et *femme*. La description des données en dimension 1 amène le choix du vecteur e_1 comme axe du nouveau repère. La représentation dans ce nouveau repère permet une reconstruction avec une erreur minimaleΓ conservant donc au maximum l'information présente dans les données. Si la tâche consiste à séparer les deux classes *homme* et *femme*Γle vecteur e_2 est en revanche mieux adaptéΓla projection des données étant plus discriminatoire que celle sur e_1. Il est néanmoins nécessaire de connaître la distribution des exemples dans l'ensemble des classes afin d'obtenir une classification optimale pour la compression [65]Γce qui semble peu réalisable dans le cadre de notre problème et contraire à notre hypothèse d'incrémentalité.

La réduction non linéaire

Dans la plupart des casΓon ne peut pas supposer l'existence d'une transformation linéaire $f : \Re^n \mapsto \Re^m$ valide sur la totalité de l'espace d'entrée. Oja [129] a démontré qu'un réseau auto-associatif à trois couches cachées peut apprendre une telle réduction non linéaire. L'activité de la couche cachée centrale représente des points dans un repère curviligne L'apprentissage d'un tel réseau par l'algorithme de rétro-propagation peut nécessiter beaucoup de temps et est susceptible de tomber dans des minima locaux.

Kambhatla et Leen [92] ont proposé une approche pour la compression linéaire par morceau basée sur une subdivision automatique de $\Re^n$ en cellules. L'algorithme associe ensuite à chacune de ces cellules une transformation linéaire telle qu'elle a été définie auparavant. Le découpage de l'espace en cellule peut se faire selon une distance euclidienne ou selon l'erreur de reconstruction totale dans une région de l'espace :

$$\begin{aligned} d_{euclidienne}(x, r_c) &= (x - rc)^T (x - r_c) \\ d_{reconstruction}(x, r_c) &= (x - r_c)^T A_c^T A_c (x - r_c) \end{aligned} \tag{7.7}$$

r_c correspondant au centre de la cellule et A_c à la matrice des vecteurs propres correspondant à la transformation linéaire associée à la cellule. L'algorithme se décompose en trois étapes :

1. subdiviser l'espace $\Re^n$ en q cellules de Voronoï par un apprentissage compétitif grâce à une des deux mesures de distance définies équation 7.7. L'utilisation de la distance euclidienne permet d'obtenir un algorithme simple mais ne tient pas compte de l'erreur de reconstruction et ne permet donc pas de la minimiser. Il faut pour ceci avoir recours à l'erreur reconstruction. L'erreur de reconstruction obtenue après minimisation dépend bien évidemment du nombre de classes choisi.

2. calculer la décomposition en composantes principales pour chacune des cellules en calculant la matrice de covariance relative à r_c et en déterminant

les différents vecteurs propres $(e_1^c, \ldots, e_m^c)$.

3. déterminer la dimension de réduction et projeter les entrées x sur les m premiers vecteurs propres.

L'application de cette démarche dans le domaine de la compression d'images a permis d'obtenir de meilleurs résultats (en terme d'erreur de reconstruction et de temps d'apprentissage) que ceux fournis par un réseau comprenant 5 couches cachées. L'erreur de reconstruction est également moindre par rapport à une simple transformation linéaire. Le temps d'apprentissage est par contre bien supérieur.

Nous allons maintenant présenter les résultats obtenus dans le cadre de la réduction linéaire dans un premier tempsΓpuis dans le cadre de la réduction linéaire par morceau.

7.6 Pré-traitement des données sensorielles par réduction linéaire

La transformation des données capteurs que l'on cherche à déterminer a pour but de réaliser un capteur virtuel. Les données utilisées pour l'extraction de vecteurs propres sont les données provenant directement des mesures capteurs.

Le calcul des vecteurs propres est réalisé en utilisant l'approche directe classique décrite dans la sous-section précédente. La matrice de covariance C est formée à partir d'un ensemble de mesures capteurs représentatives de la tâche considérée. Cette matrice C est ensuite triangularisée par la méthode de HouseholderΓpermettant de calculer les valeurs propres λ_i comme étant les racines du polynôme caractéristique de cette nouvelle matrice. Les vecteurs propres $\vec{e_i}$ sont finalement calculés grâce à l'équivalence :

$$C\vec{e_i} \approx \lambda_i \vec{e_i} \;-\; (C - \begin{pmatrix} \lambda_1 & & \\ & \backslash & \\ & & \lambda_n \end{pmatrix})\vec{e_i} \approx 0$$

Chaque vecteur propre ainsi calculé détermine un capteur virtuel dont la mesure retournée est fournie par la relation :

$$d_i{}^{virtuel} = \vec{e_i}\vec{x}$$

où $\vec{x}$ représente les mesures fournies par la totalité des capteurs utilisés.

7.6.1 Evaluation des indices extraits

La première expérience que nous avons réalisée a pour but d'essayer d'interpréter dans un cas simple les indices extraits automatiquement par la réduction

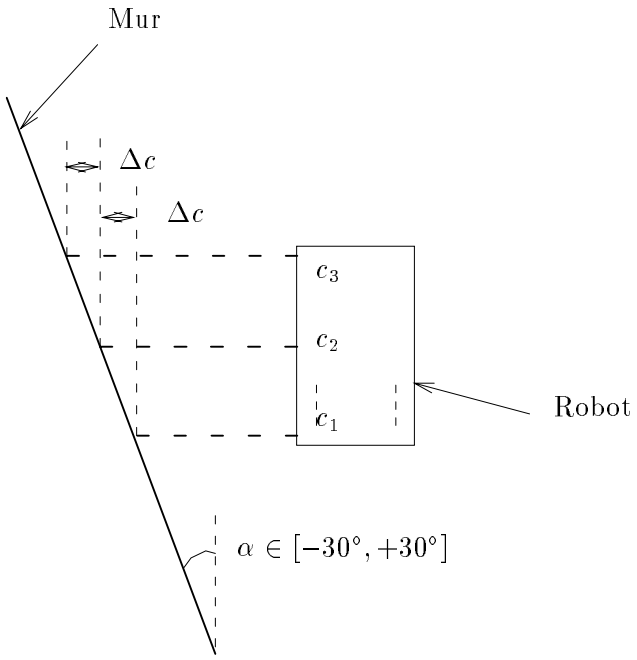


FIG. 7.20 - : *Configuration du robot et du mur pour l'acquisition des données*

linéaire et voir ainsi s'ils sont bien adaptés à notre problème. On positionne en simulation le robot le long d'un mur dont on fait varier l'angle de $-30°$ à $+30°$. On collecte les mesures provenant des capteurs c_1Γc_2 et c_3 (voir figure 7.20).

Les vecteurs propres obtenus sont les suivants :

$$\vec{e_1} = \begin{pmatrix} -0.580 \\ -0.580 \\ -0.580 \end{pmatrix}$$

$$\vec{e_2} = \begin{pmatrix} 0.707 \\ 0.0 \\ -0.707 \end{pmatrix}$$

$$\vec{e_3} = \begin{pmatrix} 0.408 \\ -0.816 \\ 0.408 \end{pmatrix}$$

La matrice de covariance étant une matrice réelle symétriqueΓles vecteurs $\vec{e_1}$Γ$\vec{e_2}$ et $\vec{e_3}$ sont orthonormaux. Ils sont classés dans l'ordre des valeurs propres croissantes.

Soit (c_1, c_2, c_3) une situation mesurée :

- Le premier capteur virtuel associé à $\vec{e_1}$ va fournir comme mesure :

$$d_1 = \begin{pmatrix} -0.580 \\ -0.580 \\ -0.580 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = 0.580(c_1 + c_2 + c_3)$$

$$= 0.580(c_2 + c_2 + \Delta c + c_2 - \Delta c)$$
$$= k * c_2$$

  Cette mesure est donc proportionnelle à la distance séparant le robot du mur.

- le deuxième capteur virtuel associé à $\vec{e_2}$ va fournir comme mesure :

$$d_2 = \begin{pmatrix} 0.707 \\ 0.0 \\ -0.707 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = 0.707(c_1 - c_3)$$
$$= 0.707(c_2 + \Delta c - c_2 + \Delta c)$$

$$= 1.414\Delta c \approx k * sin(\alpha)$$

  pour des valeurs de α peut importantes. La deuxième mesure est donc une fonction de l'angle du mur par rapport à la direction du robot.

- le troisième capteur virtuel associé à $\vec{e_3}$ va fournir comme mesure :

$$d_3 = \begin{pmatrix} 0.408 \\ -0.816 \\ 0.408 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = 0.408(c_1 - 2c_2 + c_3)$$

$$= 0.408(c_2 + \Delta c - 2c_2 + c_2 - \Delta c) = 0$$

  Cette troisième mesure est nulle lorsque le robot longe un mur. Elle ne l'est plus si les relations $c_1 = c + \Delta c$ et $c_2 = c - \Delta c$ sont fausses et si donc on se trouve par exemple en présence d'un coin.

Dans cette situation très précise (robot le long d'un mur et capteur équidistants)Γl'espace de dimension 3 formé par les distances retournées par $c_1 \Gamma c_2$ et c_3 peut être réduit en un espace de dimension 2 correspondant à la distance et à l'angle du mur par rapport au robot. Cette représentation permet d'identifier de manière unique la situation dans laquelle on se trouve et peut permettre donc la navigation. L'utilisation du troisième vecteur $\vec{e_3}$ permet de détecter un changement de situation et d'agir en conséquence. L'utilisation des trois capteurs virtuels n'est alors pas motivée par la réduction de dimension de l'espace perceptif (les deux sont de dimension 3) mais plutôt par le fait qu'elle propose une représentation de l'environnement mieux adaptée.

7.6.2 Choix de la dimension de représentation

Les résultats expérimentaux ci-dessous ont été réalisés à partir d'une base d'exemples obtenue en téléopérant le robot le long du mur représenté figure 7.12 et en conservant les mesures des 5 capteurs sur le côté droit et avant du véhicule.

Nous nous sommes plus particulièrement intéressés à la perte d'information lors de la projection d'un vecteur de données capteurs $\vec{d}$ sur la base de vecteurs propres. Cette perte est évaluée par le carré de la distance entre le vecteur initial $\vec{d}$ et le vecteur $\vec{d'}$ reconstruit à partir du vecteur projeté de $\vec{d}$. La figure 7.21 représente la moyenne et l'écart type de la perte sur l'ensemble de la base d'exemples en fonction de la dimension de la base de vecteurs propres. Conformément à ce

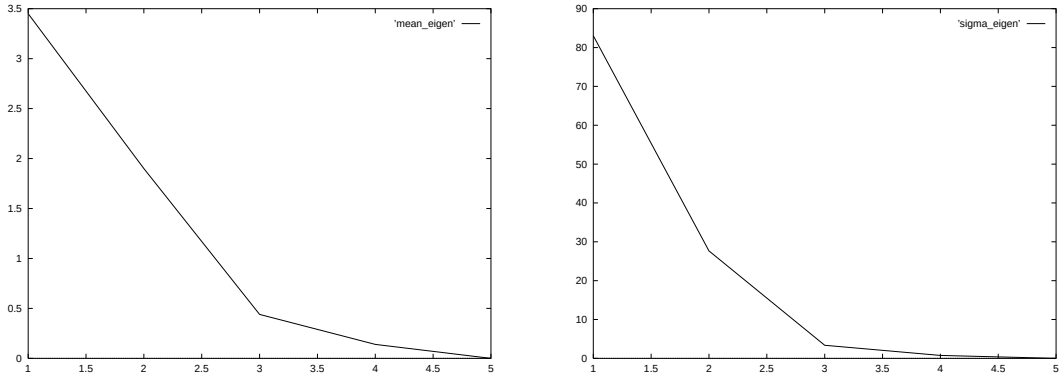


FIG. 7.21 - : *Moyenne (figure de gauche) et écart type (figure de droite) de l'erreur commise à la reconstruction en fonction de la dimension de la base de vecteurs propres.*

que l'on peut prévoirΓces deux grandeurs sont décroissantes en fonction de la dimension et s'annulent lorsque $n = 5$ (la dimension de l'espace d'arrivée est identique à la dimension de l'espace initial). Cette décroissance est au départ assez forte permettant de sélectionner une valeur de n faible et donc une forte compression de l'espace d'entrée.

les quatre copies d'écran figure 7.22 représente la trajectoire suivie par le robot avec respectivement :

- aucune compression (utilisation directe des 5 mesures capteurs).
- compression dans un espace de dimension 1.
- compression dans un espace de dimension 2.
- compression dans un espace de dimension 3.

Le robot est en mesure quelque soit la dimension choisie de se déplacer le long du mur et d'atteindre un point plus éloigné que celui atteint par le réseau exploitant directement les mesures brutes. On peut remarquer que la qualité de la trajectoire obtenue augmente avec la dimension de la base réduite.

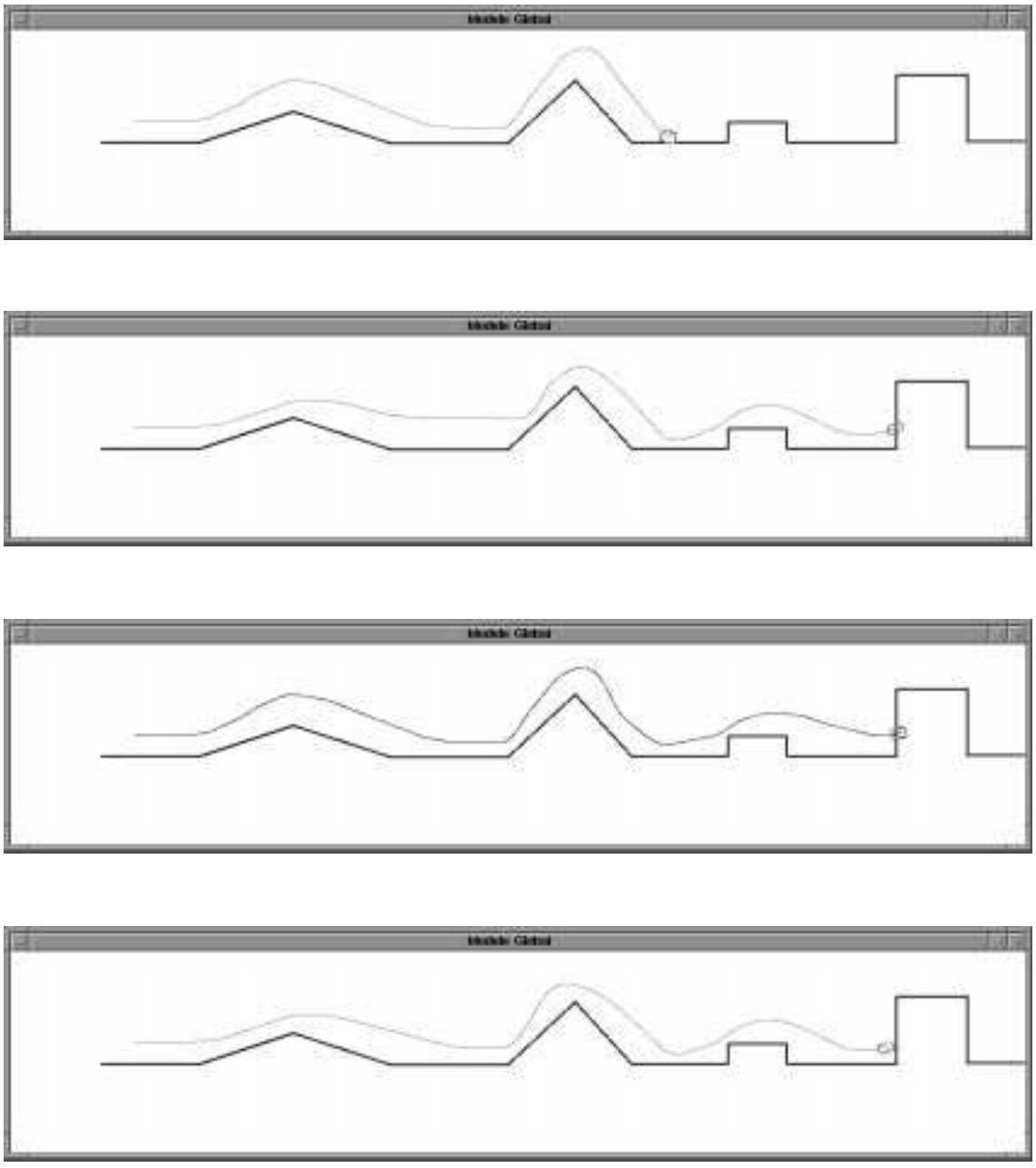


FIG. 7.22 - : *Trajectoires réalisées par le robot en utilisant respectivement les données brutes puis la compression dans un espace de dimension* 1, 2 *et* 3.

La figure 7.23 représente l'erreur commise sur l'ensemble des vecteurs ayant servi à la construction de la base. La figure en haut à gauche est associée à une base de dimension 1Γla centrale à une base de dimension 2 et la figure de droite à une base de dimension 3. La figure située en dessous est une superposition des trois précédentes. On peut constater que les différentes bases calculées sont adaptées

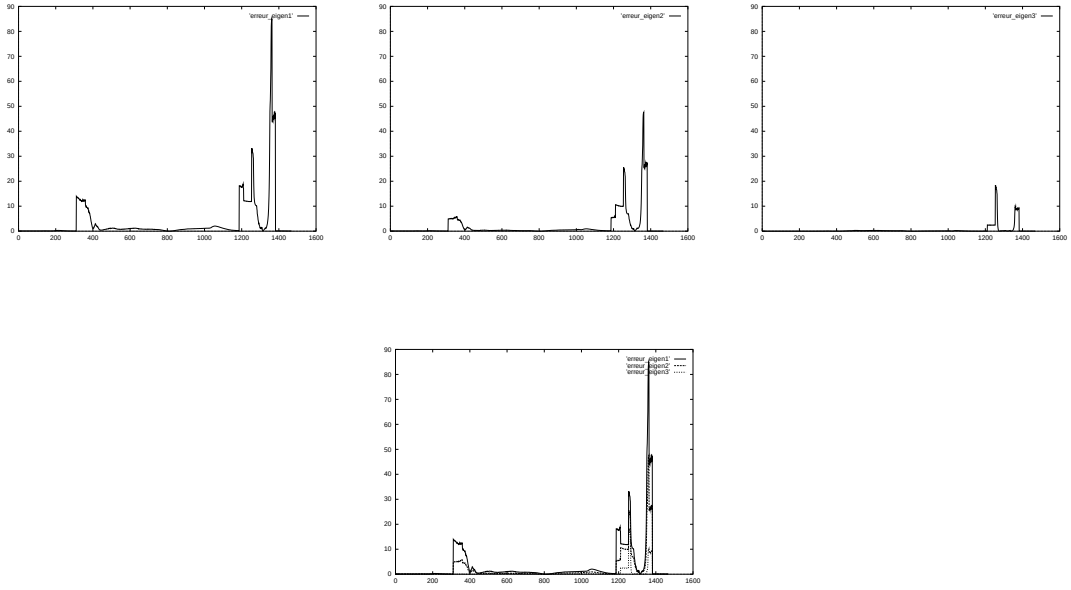


FIG. 7.23 - : *Erreur commise sur l'ensemble des points ayant servi à la construction de la base. Les figures en haut de gauche à droite sont associées respectivement à des bases de dimension* 1, 2 *et* 3. *La figure en dessous est une superposition des trois précédentes.*

à certaines zones de l'espace (faible erreur) et ne le sont pas pour d'autres (forte erreur). L'augmentation de la dimension (et donc la réduction de la compression) a pour effet d'améliorer les performances dans ces secondes zones. Cet exemple permet d'illustrer l'intérêt de la décomposition par utilisation d'un ensemble de bases de vecteurs propresΓremplaçant l'augmentation de la dimension par l'ajout d'une nouvelle base attachée à une zone de forte erreur. Ce principeΓillustré dans la prochaine sectionΓpermet ainsi de garantir une dimension faible pour chaque base et donc un bon taux de compression.

7.7 Pré-traitement des données sensorielles par réduction non linéaire

Nous allons reprendre au cours de cette section l'exemple précédent et l'appliquer au cas de la décomposition de l'espace d'entrée en un ensemble de bases de vecteurs propres. Rappelons que cette décomposition est guidée par l'erreur

commise à la reconstruction (une nouvelle base est créée si aucune des bases existantes n'est en mesure de coder correctement la donnée courante).

La figure 7.24 représente la moyenne et l'écart type de l'erreur de reconstruction sur l'ensemble de la base d'exemples en fonction de la dimension des bases de vecteurs propres. La moyenne et l'écart type sont également deux fonctions

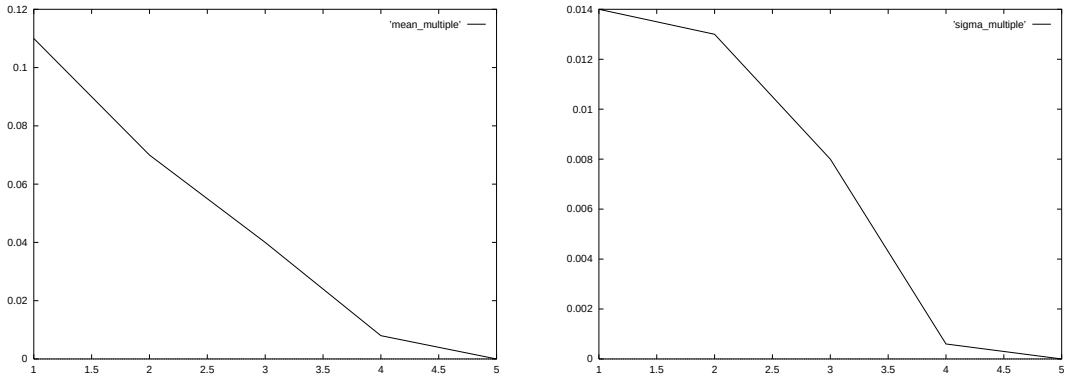


FIG. 7.24 - : *Moyenne (figure de gauche) et écart type (figure de droite) de l'erreur commise à la reconstruction en fonction de la dimension des bases de vecteurs propres.*

décroissantes vers 0 dont les valeurs sont bien inférieures à celles obtenues par une décomposition simple (voir figure 7.21). La figure 7.25 illustre l'erreur commise sur chaque point de l'ensemble d'exemples. Le nombre total de bases de vecteurs créées est de 3. Chacune de ces bases est de dimension 2. Il faut noter que par

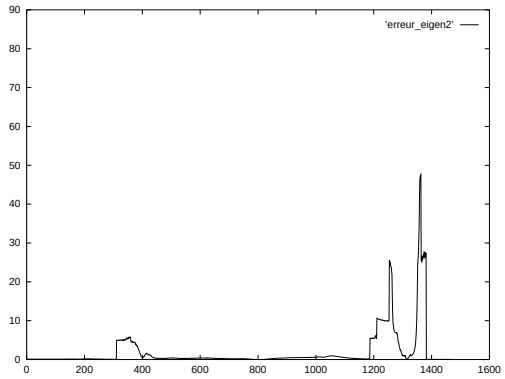


FIG. 7.25 - : *Erreur commise sur l'ensemble des points ayant servi à la construction des* 8 *bases.*

rapport à l'exemple figure 7.23Γl'erreur maximale commise peut être ajustée à la valeur souhaitée mais cela a pour conséquence d'augmenter le nombre total de bases (voir figure 7.26).

La figure 7.27 représente les trajectoires obtenues à l'aide de bases de vecteurs de dimension 1Γ2 et 3. Alors que le robot est en mesure d'accomplir sa tâche pour une dimension 2Γil n'est pas capable de la réaliser pour une dimension 3.

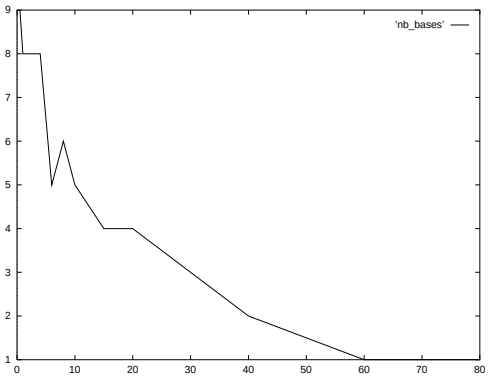


FIG. 7.26 - : *Nombre de bases créées en fonction de l'erreur maximale de reconstruction imposée par l'utilisateur.*

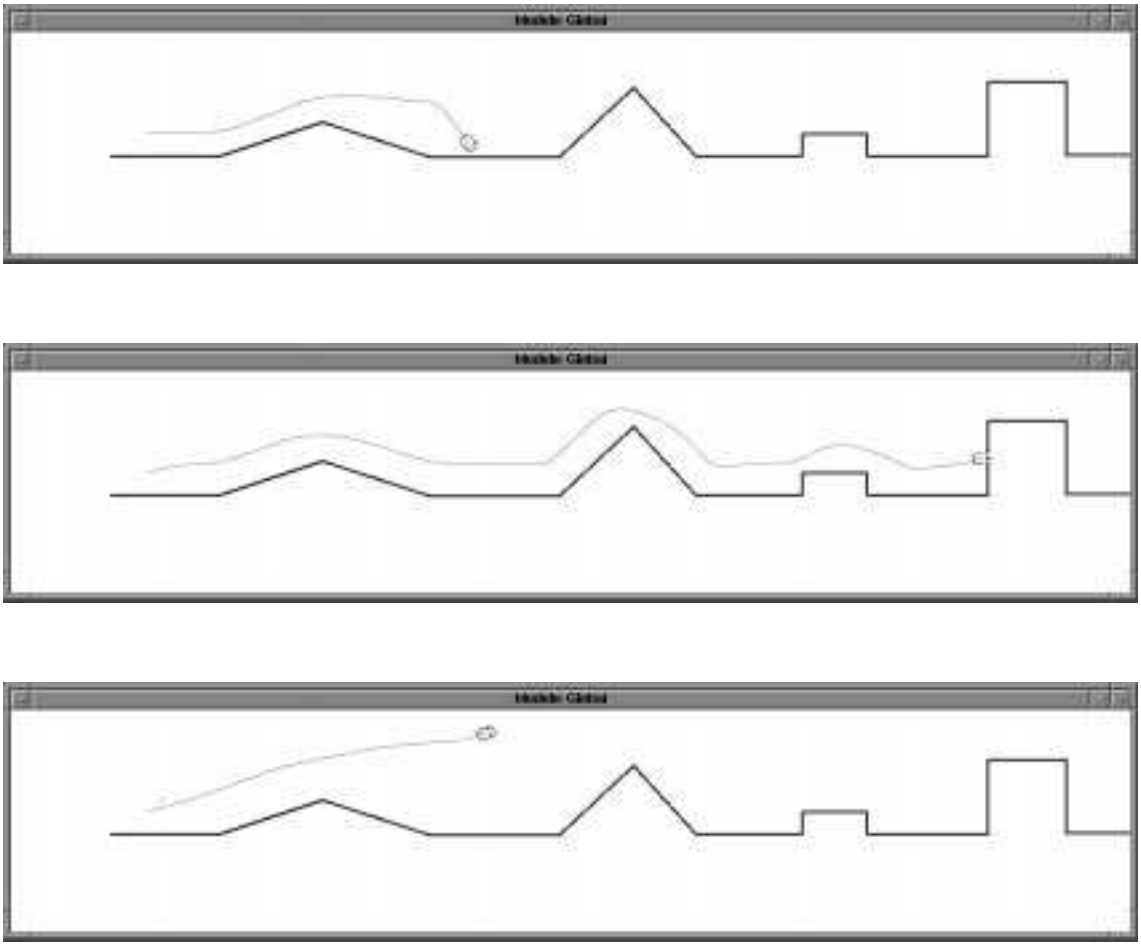


FIG. 7.27 - : *Trajectoires obtenues par le robot en utilisant des bases de vecteurs de dimension* 1, 2 *et* 3

Nous avons constaté que le même comportement était actif durant la montée et le franchissement de la pointe du triangle. Ce comportement est donc le plus spécialisé à ce type d'environnement mais n'est pas en mesure de générer une commande correcte.

Le découpage de l'espace d'entrée en un nombre variable de bases de vecteurs propres permet la création automatique d'experts attachés à des situations particulières. Le point que nous pensons être important est que ces situations ne sont pas fixées par des connaissances a priori de l'utilisateur (par exemple couloirsΓportes Ètc) mais sont basées sur les mesures capteurs du robot et devraient donc être adaptées aux capacités perceptives du système. Cette étude n'en est qu'à une phase très préliminaire et de nombreux travaux doivent être réalisésΓen particulier sur la caractérisation des comportements appris de manière à pouvoir diagnostiquer et corriger les problèmes tels que celui présenté lors de l'exemple précédent.

7.8 Conclusion

Nous nous sommes intéressés au cours de ce chapitre au problème de la détermination d'une transformation perception-action grâce à l'apprentissage supervisé. Nous avons montré en particulier comment un tel système pouvait être réalisé grâce à un réseau de type *Grow and Learn*. Le choix de ce réseau est justifié tout d'abord par sa propriété d'extensibilitéΓpermettant de se complexifier automatiquement si les données à apprendre l'exigent. Il est également justifiéΓ et nous pensons que ce point est très importantΓpar sa propriété d'apprentissage incrémental. Cette incrémentalité permet au robot d'améliorer ses performances en ne s'intéressant qu'aux nouveaux problèmes sans avoir à reprendre systématiquement les anciens pouvant être considérés comme acquis.

De manière à faciliter l'apprentissageΓnous nous sommes intéressés dans un deuxième temps au pré-traitement des données capteurs. Nous avons pour cela considéré un ensemble de capteurs virtuels obtenus par projection des données réelles sur une ou plusieurs bases de vecteurs. Ces vecteurs sont obtenus à partir d'une analyse en composante principale d'un ensemble d'exemples représentatifs. Ces travaux n'en sont qu'à leurs débuts.

Le robot apprend chaque fois qu'il n'est pas en mesure de franchir seul un obstacle. Cet apprentissage nécessite l'intervention d'un opérateur humain chargé d'indiquer la solution au problème. Afin d'augmenter l'autonomie du système et d'essayer de supprimer toute intervention extérieureΓnous allons aborder au cours du prochain chapitre le problème de l'apprentissage renforcé.

Chapitre 8

Naviguer par l'échec

Ce chapitre est consacré à l'étude de l'apprentissage renforcé dans le cadre de la navigation réactive. Nous présenterons tout d'abord les différents paradigmes mis en jeu avant de nous intéresser plus particulièrement au problème de la détermination automatique du critique prédictif pour l'apprentissage renforcé associatif.

8.1 La méthode des Différences Temporelles

Nous allons présenter au cours de cette section les travaux de Sutton dans le domaine de l'apprentissage pour prédire. Le lecteur pourra se reporter pour plus de détails à [163] dont cette présentation s'inspire. Ces travaux sont très étroitement liés aux développements de l'apprentissage renforcé en contrôle. De manière à clarifier l'exposéΓnous avons préféréΓcomme l'auteur le suggèreΓles sortir tout d'abord de ce contexte.

8.1.1 Apprendre à prédire

Le problème abordé au cours de cette section est le suivant : comment peut-on apprendre à prédire ou comment peut-on utiliser l'expérience passée afin de prédire le comportement futur d'un système dont le fonctionnement est partiellement connu. Le but d'une telle prédiction peut être par exemple :

- de déterminer siΓpour un adversaire donnéΓune position particulière aux échecs peut conduire à une victoire ou une défaiteΓ
- de déterminer s'il va pleuvoir ou non ce week-endΓ
- de déterminer si le cours de la bourse va augmenter ou diminuer lors de la prochaine session.

Les problèmes de prédiction peuvent être classés en deux grandes catégories :

1. les problèmes de prédiction à un pas. La totalité des informations permettant de juger si la prédiction est correcte ou non est révélée en une seule étape.

2. les problèmes de prédiction à pas multiples. L'exactitude d'une prédiction ne sera observée que plusieurs pas après qu'elle ait été formulée. Néanmoins, des informations partielles sur ce sujet sont révélées à *chacun* de ces pas. Dans le cas par exemple de la prédiction météorologique pour le week-end, l'observation du ciel au fur et à mesure de l'avancée des jours dans la semaine fournit de plus en plus d'informations relatives au temps du dimanche.

Les travaux que nous présentons ici concernent plus particulièrement le deuxième type de problèmes.

Nous considérerons que les données expérimentales sont fournies sous la forme $(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_m}, z)$. Le vecteur $\vec{x_i}$ correspond au vecteur d'observation obtenu au temps t_i et z un réel représentant le résultat final que l'on souhaite prédire. On associe à chaque observation $\vec{x_i}$ une prédiction P_i de la valeur finale z :

$$P_i = P(\vec{x_i}, \vec{w})$$

où $\vec{w}$ représente un ensemble de paramètres modifiables permettant de définir la fonction P (les poids synaptiques d'un réseau de neurones par exemple). Le but de tout algorithme d'apprentissage est de proposer un ensemble de règles permettant de mettre à jour ces paramètres :

$$\vec{w} \leftarrow \vec{w} + \sum_{t=1}^{m} \vec{\Delta} w_t \tag{8.1}$$

Le problème étant maintenant posé, nous allons nous intéresser à deux grandes familles d'approches permettant de le résoudre : l'apprentissage supervisé et la méthode des différences temporelles.

8.1.2 L'approche $TD(\lambda)$

Tout problème de prédiction peut être ramené à un problème d'apprentissage supervisé. Il suffit pour cela de présenter au système les exemples formés par les couples (x_i, z). La variation du vecteur de paramètres $\vec{w}$ au temps t dépend de l'erreur de prédiction commise et de l'influence de $\vec{w}$ sur cette prédiction :

$$\vec{\Delta} w_t = \alpha(z - P_t)\nabla_w P_t \tag{8.2}$$

α étant un paramètre compris entre 0 et 1 réglant la vitesse de l'apprentissage. Ce type d'approche présente deux grands problèmes :

1. il est nécessaire d'attendre le résultat z de manière à pouvoir commencer l'apprentissage : la totalité des exemples doit être stockée et la procédure n'est pas incrémentale. De plus, la charge de calcul induite par l'algorithme n'est pas uniformément répartie sur la totalité de l'expérience mais est en revanche concentrée à la réception de la réponse finale z.

2. La structure temporelle des observations n'est pas utilisée.

La famille d'algorithme $TD(\lambda)$ a été conçue par Sutton afin d'apporter une réponse à ces deux constatations. Reprenons l'équation 8.2. $(z - P_t)$ peut être réécrit en faisant apparaître les successions de prédictions :

$$z - P_t = \sum_{k=t}^{m} (P_{k+1} - Pk) \tag{8.3}$$

en posant $P_{m+1} = z$. En reportant les équations 8.3 et 8.2 dans l'équation 8.1, on obtient :

$$\begin{aligned}
&\vec{w} \leftarrow \vec{w} + \textstyle\sum_{t=1}^{m} \alpha(z - P_t)\nabla_w P_t \\
&= \vec{w} + \textstyle\sum_{t=1}^{m} \alpha \sum_{k=t}^{m} (P_{k+1} - P_k)\nabla_w P_t \\
&= \vec{w} + \textstyle\sum_{k=1}^{m} \alpha \sum_{t=1}^{k} (P_{k+1} - P_k)\nabla_w P_t \\
&= \vec{w} + \textstyle\sum_{t=1}^{m} \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \nabla_w P_k \\
&\Rightarrow \vec{\Delta} w_t = \alpha(P_{t+1} - P_t) \textstyle\sum_{k=1}^{t} \nabla_w P_k
\end{aligned}$$

La formule ci-dessus de calcul de $\vec{\Delta} w_t$ est appelée *TD(1)* pour une raison apparaissant ci-après. Elle fait référence explicitement à la succession dans le temps des différentes prédictions et permet de réaliser un calcul incrémental (la valeur z n'apparaît que pour le calcul de $\vec{\Delta} w_m$). Il n'est de plus pas nécessaire de stocker les différents exemples ni même les différents gradients (il suffit de conserver uniquement la somme).

La modification des paramètres $\vec{w}$, et donc l'apprentissage, intervient lorsque $P_{t+1} \neq P_t$, c'est-à dire lorsque l'évolution de la situation entre ces deux instants a apporté suffisamment d'éléments nouveaux permettant de remettre en question la valeur z précédemment prédite. Cette différence de prédiction est propagée sur toutes les situations antérieures (somme des gradients dans la formule).

Les valeurs des paramètres fournies par $TD(1)$ et par la formule 8.2 sont identiques. A partir de $TD(1)$, une famille complète d'algorithmes peut être générée en atténuant la propagation sur les situations antérieures de l'erreur de prédiction. L'atténuation proposée par Sutton est de forme exponentielle. Elle est réglée par le paramètre λ :

$$TD(\lambda) \ : \ \vec{\Delta} w_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k \tag{8.4}$$

Nous allons revenir sur les conséquences de cette équation dans la mise à jour des paramètres au travers de deux exemples simples.

8.1.3 Exemples

De manière à mieux comprendre la différence existant entre une approche supervisée et une approche de type $TD(\lambda)$ avec $\lambda \neq 1$Γnous allons reprendre l'exemple de la prévision météorologique. On utilise une fonction P linéaire par rapport à $\vec{w}$ et par rapport à $\vec{x}$: $P_t = \vec{w}^t x_t$. Cette fonction a pour rôle à partir d'un vecteur d'observation d'indiquer la probabilité qu'il pleuve le dimanche. De manière à simplifierΓle vecteur observation ne contient que le jour de réalisation de la prédiction et ne contient donc pas d'information basée par exemple sur l'observation du ciel. Il comporte 7 composantes x_i telles que :

$$x_i = \begin{cases} 0 & \text{pour tous les jours sauf le jour } i \\ 1 & \text{le jour } i \end{cases}$$

Le vecteur de paramètres $\vec{w}$ est un vecteur de 7 composantes également. Chaque composante w_i représente la probabilité au jour i qu'il pleuve le dimanche.

Plaçons-nous tout d'abord dans le cas de l'apprentissage supervisé. P étant linéaire par rapport à $\vec{w}$Γon obtient directement $\nabla_w P_t = \vec{x}_t$. En reprenant l'équation 8.2Γon obtient (on pose $\alpha = 1$) :

$$\vec{\Delta} w_t = (z - \vec{w}^t \vec{x}_t)\vec{x}_t$$
$$\Rightarrow \forall i \in \{1, \ldots, 7\},\ w_i \leftarrow z$$

La mise à jour des coefficients est illustrée figure 8.1. Le dimancheΓl'estimation associée à chaque journée est mise à jour à la valeur du temps du dimanche.

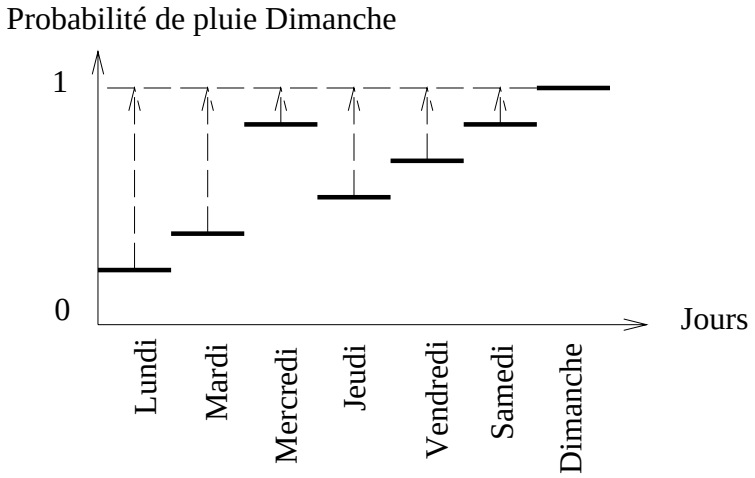


FIG. 8.1 - : *Mise à jour par apprentissage supervisé de l'estimation du temps pour le dimanche*

Plaçons-nous maintenant dans le cas de l'apprentissage $TD(0)$. En reprenant

l'équation 8.4Γon obtient :

$$\vec{\Delta} w_t = (P_{t+1} - P_t)\vec{x}_t$$
$$\Rightarrow \forall i \in \{1, \ldots, 7\},\ w_i \leftarrow w_{i+1}$$

La mise à jour des coefficients est illustrée figure 8.2. Elle ne s'effectue plus uniquement le dimanche mais au fur et à mesure de la semaine. Au jour $i+1$Γ l'estimation du jour i est réajustée à l'estimation du jour $i+1$. Cette démarche peut être justifiée intuitivement par le fait que l'on dispose en $i+1$ de plus d'informations qu'en i permettant de réaliser une meilleure prédiction. Dans le cas d'un algorithme $TD(\lambda)$ avec $\lambda \neq 0$Γla remise à jour des estimations ne se limite pas uniquement à la veille mais également aux jours précédents.

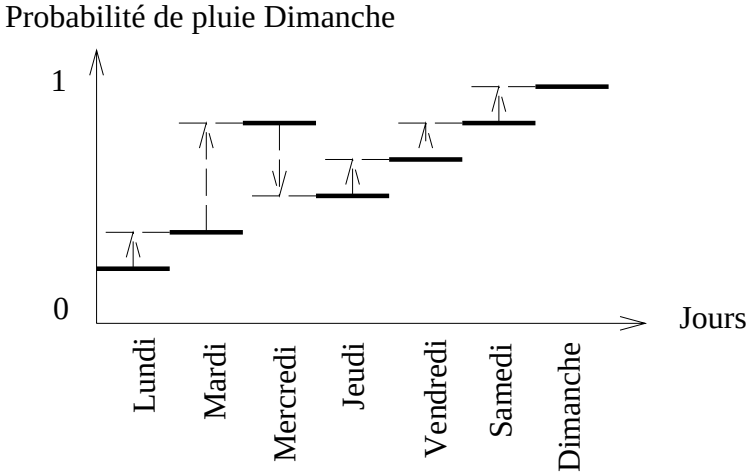


FIG. 8.2 - : *Mise à jour par algorithme $TD(0)$ de l'estimation du temps pour le dimanche.*

L'exemple suivant a pour objectif d'essayer de montrer intuitivement que l'apprentissage par différence temporelle peut être plus efficace que l'apprentissage supervisé. Nous nous plaçons dans le cas d'un jeu à deux joueurs. Supposons qu'il existe une configuration que vous ayez apprise comme étant mauvaise pour vous : dans 90% des cas la partie se termine par la victoire de l'adversaire et dans seulement 10% des cas par votre victoire (voir figure 8.3). Au cours d'une partieΓvous vous retrouvez dans une nouvelle configuration encore jamais explorée. Vous vous retrouvez au prochain coups dans la situation étiquetée *mauvaise* et vous gagnez ensuite la partie. Quelle étiquette doit-être attachée à cette nouvelle configuration ?

- l'apprentissage supervisé va réaliser directement l'association (nouvel état $\rightarrow$ victoire) et associera l'étiquette *bonne*.
- l'algorithme $TD(\lambda)$ en revanche va associer le nouvel état et l'état suivant et attribuera l'étiquette *mauvais*.

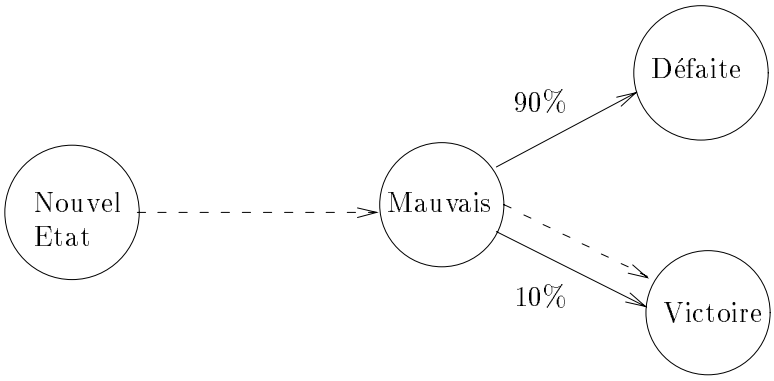


FIG. 8.3 - : *Lorsque la configuration du jeu atteint l'état étiqueté* mauvais, *la partie se termine dans* 90% *des cas par votre perte et dans* 10% *des cas par votre victoire. Question : comment faut-il étiqueter le nouvel état en cas de victoire après son exploration?*

L'algorithme $TD(\lambda)$ fournira immédiatement la réponse correcte. L'apprentissage supervisé finira également par apprendre que la situation est mauvaise après plusieurs passages se terminant par une défaite. Mais cela nécessitera du temps.

8.1.4 Extensions de l'algorithme

L'utilisation des algorithmes $TD(\lambda)$ ne se limite pas à la prédiction de la valeur finale d'une succession d'observations. Ils peuvent également être utilisés afin de prédire une grandeur s'accumulant au cours des différents pas restant à accomplir. Par exemple dans le cas du contrôle d'un pendule inversé cela peut être le temps restant avant la chute du balancier. Dans le cas d'un jeu cela peut être le nombre de coups restant avant la victoire d'un des joueurs.

La quantité P_t que l'on cherche à prédire à l'instant t n'est donc plus une valeur constante z mais une grandeur dépendant du temps :

$$z_t = \sum_{k=t}^{m} c_{k+1}$$

L'erreur de prédiction est $P_t - z_t$. En remplaçant z_t par son expression et en redéveloppant des calculs similaires à ceux décris précédemment on obtient :

$$\vec{\Delta} w_t = \alpha(c_{t+1} + P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k \tag{8.5}$$

Le calcul de la somme des coûts c_t tel que nous venons de le décrire précédemment nécessite que le processus que l'on observe ait un nombre de cycles fini m. Cette hypothèse peut être relâchée en remplaçant la prédiction de la somme

complète par la prédiction de la somme "amortie" :

$$z_t = \sum_{k=0}^{+\infty} \gamma^k c_{t+k+1},\ \gamma \in [0,1]$$

Remarque : si la fonction de prédiction est correcte, on obtient alors :

$$\begin{aligned} & P_t = z_t \\ \Rightarrow\ & P_t = \textstyle\sum_{k=0}^{+\infty} \gamma^k c_{t+k+1} \\ & \quad = c_{t+1} + \gamma \textstyle\sum_{k=0}^{+\infty} \gamma^k c_{t+k+2} \\ & \quad = c_{t+1} + \gamma P_{t+1} \end{aligned}$$

l'écart entre la valeur P_t et $c_{t+1} + \gamma P_{t+1}$ correspond à l'erreur de prédiction. La relation de mise à jour des paramètres $\vec{w}$ s'écrie :

$$\vec{\Delta} w_t = \alpha(c_{t+1} + \gamma P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k \tag{8.6}$$

Après cette présentation générale de la famille d'algorithmes $TD(\lambda)$, nous allons maintenant revenir sur l'apprentissage renforcé, sujet principal de ce chapitre, en montrant la complémentarité de ces deux approches.

8.2 L'apprentissage renforcé

Le paradigme utilisé au cours de ce chapitre est celui de l'apprentissage renforcé tel qu'il a été défini brièvement au cours de la section 6.3.2. Rappelons que dans le cas de l'apprentissage supervisé, le contrôleur reçoit pour toute situation perçue la commande correspondante fournie par un agent externe. Dans le cas de l'apprentissage renforcé, le contrôleur reçoit pour chaque situation perçue l'évaluation du résultat de la commande qu'il a proposée. Cette fonction d'évaluation du comportement du robot est fournie par l'utilisateur et permet de coder les objectifs devant être atteints par le système sans indiquer comment les atteindre. La recherche de la solution au problème posé n'est plus réalisée par le concepteur du système mais par l'ordinateur.

Par nature, l'apprentissage renforcé est basé sur la découverte par le système lui-même de son propre fonctionnement et de ses interactions avec son environnement. Cette découverte est réalisée au travers d'essais multiples en tenant compte des expériences passées. Il n'y a donc pas de séparation entre la phase *apprentissage* et la phase *exploitation*.

8.2.1 Algorithme général

L'apprentissage renforcé cherche à déterminer une fonction $e(i,s)$ où i représente les données perçues. Cette fonction est extraite d'une famille générale de

fonctions par détermination des valeurs d'un ensemble de paramètres s (appelés aussi état). Ces paramètres peuvent être les coefficients d'une expression polynomialeΓles poids synaptiques d'un réseau de neurones ou bien encore les différentes règles d'un système flou.

Il existe de nombreux travaux relatifs à l'apprentissage renforcé. Les différents algorithmes existants peuvent être regroupés dans l'algorithme général suivant [91] :

```
s := s0
loop
    i := input
    a := e(s,i)
    output a
    r := reinforcement
    s := u(s,i,a,r)
end loop
```

Cet algorithme est représenté de manière graphique figure 8.4. L'état s de f est initialisé à s_0. s_0 peut être aléatoire ou représenter une connaissance initiale fournie par l'utilisateur. L'algorithme boucle ensuite à l'infini. Une nouvelle acquisition i de la situation est réalisée permettant de calculer l'action a correspondante grâce à la fonction e. Cette action est ensuite exécutée provoquant un changement d'état du monde et la génération d'un renforcement r associé. L'analyse par la fonction u de l'association perception - action (i, a) et de l'effet obtenu r permet de déduire la nouvelle valeur des différents paramètres sΓmodifiant ainsi la transformation f.

Le cœur de l'algorithmeΓfaisant sa spécificité et permettant l'apprentissage ou non du comportement $e(i, s)$ répondant aux objectifsΓprovient de la fonction u.

8.2.2 Evaluation de la transformation apprise

Tout au long de l'apprentissageΓle comportement du robot va évoluer grâce à la fonction u introduite au cours de la section précédente. De manière à pouvoir juger les performances et à guider l'algorithme d'apprentissageΓil faut être en mesure d'évaluer le comportement obtenu. Pour plus de détails sur les deux critères énoncés ci-dessousΓle lecteur pourra se reporter à [91].

Critère local

Nous allons tout d'abord préciser la notion de renforcement attendu erΓl'agent étant dans la situation i et effectuant l'action a.

$$er(i, a) = \sum_{\acute{i} \in I} r(\acute{i}) \acute{W}(i, a)(\acute{i})$$

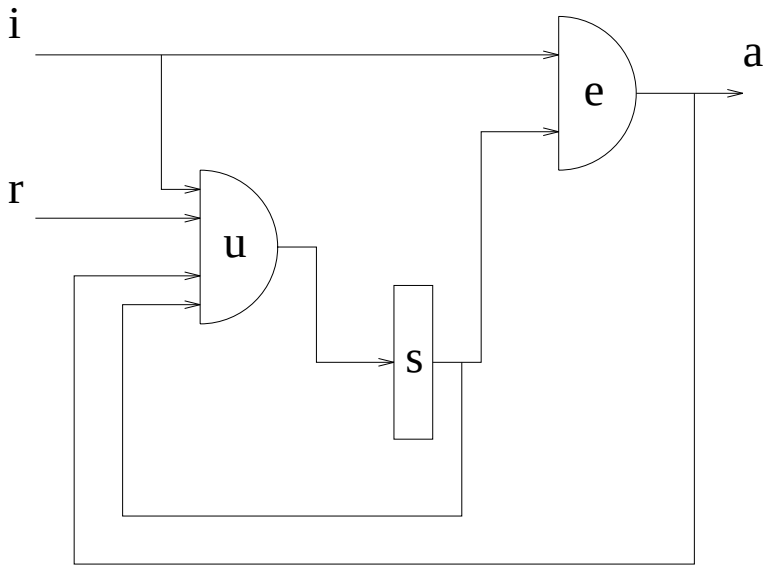


FIG. 8.4 - : *Algorithme général pour l'apprentissage renforcé*

où I représente l'ensemble des états pouvant être perçus et où $\acute{W}(i,a)(i')$ exprime la probabilité que l'environnement passe de l'état i à l'état $\acute{i}$ si le système exécute l'action a. L'utilisation de probabilité de transitions entre les différents états dans le modèle de l'environnement permet de prendre en compte la présence de capteurs et d'effecteurs bruités ou défectueux au niveau du système évoluant dans ce milieu [91].

Une action a sera dite optimale pour une situation i si cette action maximise le renforcement attendu :

$$\forall i \in I,\ \forall a \in A,\ Optimal(i,a) - \ \forall a' \in A,\ er(i,a) > er(i,a') \qquad (8.7)$$

où A représente l'ensemble des actions possibles.

Un tel critère n'est pas satisfaisant car il conduit à l'apprentissage de comportements sacrifiant un renforcement futur important pour un renforcement immédiat même faible. Considérons le cas d'un robot mobile devant atteindre un but tout en évitant les obstacles. Le renforcement choisi peut être une fonction croissante de la distance au but et une fonction décroissante de la distance aux obstacles. Si l'algorithme d'apprentissage cherche à générer une transformation e maximisant le renforcement au prochain cycleΓil ne sera pas en mesure de créer un comportement permettant au robot de contourner un mur (voir figure 8.5). Un tel comportement nécessite en effet pour le robot d'être momentanément puni en s'éloignant du but pour être finalement fortement récompensé en l'atteignant.

Un comportement appris e respectant le critère 8.7 réalise une montée du gradient de la fonction de renforcement. Dans le cas de notre exempleΓcette fonction de renforcement peut être vue comme l'inverse d'une fonction de potentiel (voir

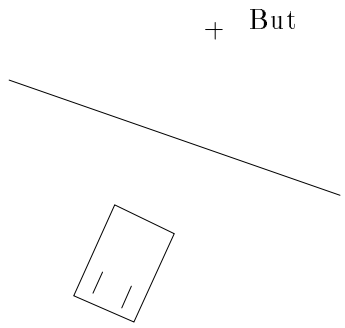


FIG. 8.5 - : *Le robot ne peut contourner le mur que s'il accepte d'être momentanément puni avant d'être récompensé*

chapitre 4). e correspond donc aux comportements obtenus par les méthodes de type champs de potentiels et présente en particulier des zones de minima locaux comme celle représentée figure 8.5.

Il est nécessaire de définir un critère plus global.

Critère global

L'algorithme d'apprentissage doit être guidé par une évaluation du comportement sur un horizon plus vaste que le résultat de l'action choisie. Le critère retenu n'est plus le prochain renforcement mais la somme des renforcements futurs. De manière à privilégier néanmoins un renforcement proche à un renforcement trop éloignéΓles différentes valeurs sont amorties par une fonction exponentielle.

L'utilisation d'un critère global prenant en compte les renforcements futurs permet :

1. d'apprendre à choisir une action non pas pour son renforcement immédiat (critère local) mais parce qu'elle fait partie d'une chaîne d'actions conduisant à une situation de fort renforcementΓ

2. d'associer une action et son renforcement dans le cas où il existe un délai de plusieurs cycles entre les deux événements [1].

Le second point est très important dans le cadre d'une application robotiqueΓen particulier si un renforcement n'est disponible que lorsque le but est atteint ou que le robot est entré en collision avec un obstacle.

On appelle $er_\gamma(i,a)$ le renforcement total amorti attendu lors de l'exécution de l'action a pour la perception iΓen supposant qu'après avoir exécuté cette actionΓ

1. Problème connu sous le nom de *Temporal Credit Assignment*

le robot se comporte de manière optimale :

$$er_\gamma(i,a) = er(i,a) + \gamma \sum_{\acute{i} \in I} \acute{W}(i,a)(\acute{i}) er_\gamma(\acute{i}, Opt_\gamma(\acute{i})) \tag{8.8}$$

L'action optimale $Opt_\gamma(\acute{i})$ étant définie par :

$$\begin{array}{l} \forall i \in I,\ a = Opt_\gamma(i) \\ Optimal_\gamma(i,a) - \ \forall a' \in A,\ er_\gamma(i,a) > er_\gamma(i,a') \end{array} \tag{8.9}$$

Le rôle de l'apprentissage renforcé est de générer un paramètre $s_{optimal}$ tel que la transformation e réponde au critère d'optimalité :

$$e(i, s_{optimal}) = Opt_\gamma(i)$$

Il existe à notre connaissance deux grandes familles d'approches dans ce domaine :

- l'apprentissage renforcé par utilisation d'un *Critique Heuristique Adaptatif*[2]Γ
- le *Q-Learning.*

8.2.3 Le Critique Heuristique Adaptatif

Les algorithmes classiques de renforcement tels qu'ils ont été présentés section 8.2.1 ont pour tâche de modifier à chaque étape la fonction e de manière à tenir compte de l'action exécutée et de ses conséquences immédiates. Ce mode de fonctionnement permet de gérer directement le critère d'optimalité local mais n'est pas en mesure de travailler avec un critère global (seul le triplet courant *(état, action, renforcement résultant)* est connu à chaque cycle). De manière à contourner ce problèmeΓSutton a proposé de transformer le renforcement $r(i,a)$ reçu de l'environnement en $R(i,a) = er_\gamma(i,a)$ correspondant au renforcement total amorti attendu. La maximisation de ce nouveau renforcement R par un algorithme classique d'apprentissage renforcé appliquant un critère local permettra de respecter le critère global.

Le nouveau renforcement R est la somme pondérée des futurs renforcements reçus par le robot en suivant sa loi de commande actuelle. Il réalise une prédiction du comportement futur du robot. En remplaçant le coût c_i de l'équation 8.6 par le renforcement r provenant de l'environnementΓR peut être calculé grâce à un algorithme de la famille $TD(\lambda)$ (voir section 8.1.2).

Le principe d'un algorithme d'apprentissage renforcé par utilisation d'un critique adaptatif (ou apprentissage renforcé associatif) est le suivant (voir figure 8.6) :

- le robot perçoit son environnement et choisit une actionΓ

2. En anglais : Adaptive Heuristic Critic ou AHC

- l'exécution de cette action produit le renforcement r de la part de l'environnement et R de la part du critique adaptatifΓ
- le renforcement r est utilisé pour l'apprentissage du critique adaptatifΓ
- la fonction de commande e est modifiée de manière à maximiser à chaque pas la valeur R.

Le critique adaptatif et la loi de commande sont appris simultanément au cours de l'exécution du système.

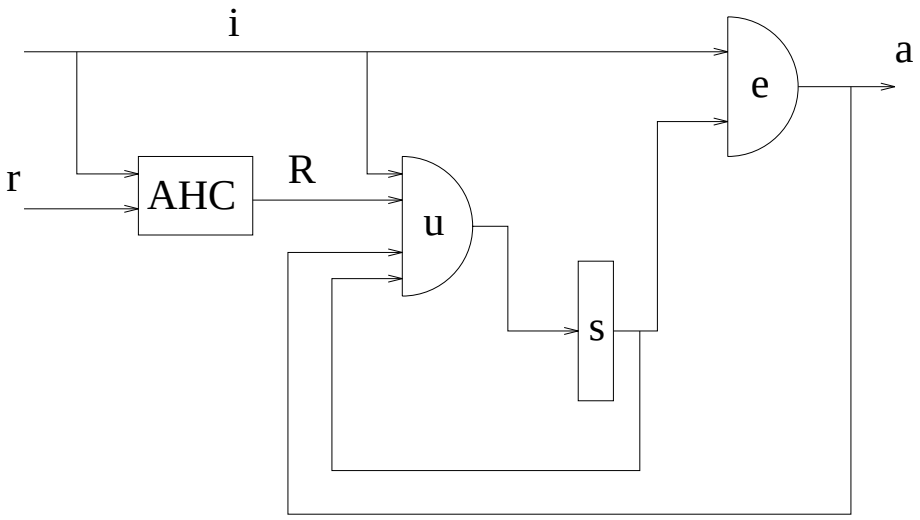


FIG. 8.6 - : *Algorithme d'apprentissage renforcé avec utilisation d'un Critique Heuristique Adaptatif.*

Remarque : conceptuellementΠ'apprentissage renforcé associatif peut être rapproché de la méthode de planification variationnelle dans les approches de type potentiels (voir section 4.2.5) :

- en apprentissage renforcéΓle principe est de générer une loi de commande dont l'exécution permettra de maximiser la somme des renforcements reçusΓ
- en planification variationnelleΓle principe est de générer un chemin tel que la somme des potentiels le long de ce chemin soit minimum.

8.2.4 Le *Q-Learning*

La loi de commande optimale que l'on cherche est solution des équations récursives 8.8 et 8.9. Elle peut être obtenue par l'utilisation d'une approche de type programmation dynamique. Les algorithmes mis en jeu ne sont néanmoins pas adaptés aux contraintes d'apprentissage "en-ligne" propre aux types de problèmes traités. De manière à contourner cette difficultéΓWatkins a proposé un algorithme

qualifié de *programmation dynamique incrémentale* (voir [185] par exemple). Cet algorithme s'appelle *Q-Learning*.

Soit $V^e(i)$ la valeur d'un état exprimée en termes de renforcement total amorti reçu en suivant la loi de commande e :

$$V^e(i) = er(i, e(i)) + \gamma \sum_{\acute{i} \in I} \acute{W}(i, e(i)) V^e(\acute{i})$$

Soit $Q^e(i, a)$ la fonction estimant le renforcement total amorti attendu en exécutant la commande a à partir de la situation i et en suivant ensuite la loi de commande e :

$$Q^e(i, a) = er(i, a) + \gamma \sum_{\acute{i} \in I} \acute{W}(i, a)(\acute{i}) V^e(\acute{i})$$

On considérera une représentation discrétisée de l'espace d'entrée (nombre fini de configurations i perçues) ainsi qu'un nombre fini d'actions a possibles. La fonction Q peut alors être représentée à l'aide d'un tableau à double entrées $s[i, a]$.

A chaque cycle, la loi de commande e choisit l'action suivante à effectuer grâce à un tirage aléatoire dont la densité est une gaussienne centrée sur l'action a_{max} de plus grande utilité :

$$a_{max} = max_{a \in A} s[i, a]$$

Le tirage aléatoire est justifié par le besoin de ne pas uniquement suivre l'action la plus utile mais de s'en écarter éventuellement un peu de manière à pouvoir explorer d'autres solutions.

L'exécution de la commande a va provoquer un changement d'état $i \mapsto \acute{i}$ du système et l'émission d'un renforcement r. La fonction d'utilité de i est alors remise à jour :

$$s[i, a] \leftarrow (1 - \alpha) s[i, a] + \alpha (r + \gamma \max_b (s[\acute{i}, b]))$$

Le terme $s[i, a]$ dans la partie droite représente l'évaluation courante de l'utilité. Le terme $r + \gamma \max_b (s[\acute{i}, b])$ représente la nouvelle évaluation de cette utilité sachant que le changement de situation de i à $\acute{i}$ a généré un renforcement r.

Au fur et à mesure que le robot explore son environnement, il construit sa fonction Q d'évaluation de l'utilité d'une commande. Cette même fonction au cours de sa construction joue le rôle d'heuristique de manière à guider le robot dans son exploration. Contrairement à l'apprentissage renforcé associatif, l'algorithme Q-Learning utilise une seule et même structure pour coder à la fois la fonction d'évaluation et la sélection de la prochaine action.

Ces deux schémas de base étant maintenant présentés, nous allons revenir sur l'un des problèmes principaux de l'apprentissage renforcé : le temps de convergence.

8.3 Réduction du temps d'apprentissage

Un des problèmes principaux posé par l'apprentissage renforcé est le temps de convergence trop important [186] vers une solution acceptable. Ce phénomène peut avoir plusieurs causes comme une mauvaise gestion du conflit exploration exploitation (voir section 8.3.1) ou la réalisation d'explorations inutiles (voir section 8.3.4).

De nombreuses approches ont été proposées de manière à réduire ce temps d'apprentissage. Nous allons revenir sur quelques unes d'entre elles.

8.3.1 Exploration et exploitation

Les algorithmes d'apprentissage renforcé ne proposent pas de séparation entre la phase apprentissage et la phase exploitation. De part leur nature ils sont confrontés en permanence à un conflit entre deux objectifs contradictoires :

1. l'exploration
2. l'exploitation.

L'exploration a pour but de renseigner le robot sur son environnement et ses capacités d'action sur celui-ci. Son objectif est de *minimiser le temps d'apprentissage*. L'exploitation a pour but de rendre le robot opérationnel en lui imposant de respecter les contraintes fixées par la fonction de renforcement. Son objectif est de *minimiser le coût d'exécution*. Ces deux objectifs sont contradictoires. Les besoins d'exploration peuvent amener le robot à exécuter une action de manière à découvrir son effet cette action pouvant conduire à un renforcement négatif.

Thrun rapporte deux grandes catégories d'exploration [172 171]:

- l'exploration non dirigée.
- l'exploration dirigée.

l'exploration non dirigée consiste à choisir aléatoirement la prochaine commande à exécuter ce tirage aléatoire étant généralement centré sur la proposition de l'exploitation (maximisant le renforcement total prédit). L'exploration dirigée choisit une action à exécuter de manière à explorer le plus efficacement possible. Par exemple :

- l'action peut être choisie de manière à amener le système dans un état peu visité *(counter-based exploration)*.
- l'action peut être choisie de manière à amener le système dans un état dans lequel il a peu de connaissances [175 174 173] *(error-based exploration)*. Ceci peut être fait en maintenant un modèle du fonctionnement du "monde" (état $\times$ action $\mapsto$ état) ainsi qu'un modèle de la validité de ce

modèle (évaluation de l'erreur commise sur l'état prédit). Le robot est alors conduit vers les zones de faible validité de manière à compléter et corriger sa connaissance.

La gestion du conflit entre l'exploitation et l'exploration peut être réalisée en choisissant une action maximisant une combinaison linéaire de l'apport dans les deux domaines de l'exécution de cette action. On parle alors d'exploration guidée. Cette combinaison linéaire peut être fixe ou variable (mécanisme d'attention sélectiveΓvoir [172] par exemple).

Thrun rapporte que sous certaines hypothèsesΓl'apprentissage par exploration aléatoire (non guidée) est moins efficace (en terme de temps) que l'exploitation pureΓelle-même moins efficace que l'exploration guidée.

8.3.2 Utilisation d'expériences mentales

Sutton [165Γ164] propose grâce à l'architecture Dyna d'accélérer le processus d'apprentissage en intégrant une phase de planification. Il maintient pour cela un modèle du monde dont la construction se déroule parallèlement à l'apprentissage de la loi de commande. Ce modèle est une fonction associant à un état et une action une prédiction du renforcement reçu et du nouvel état obtenu. Il permet de générer des expériences hypothétiques (cf planification). Les résultats de ces expériences sont utilisés au niveau de l'apprentissage renforcé au même titre que les résultats provenant d'expériences réellesΓréduisant ainsi dans des proportions significatives le temps nécessaire à la construction de la loi de commande. L'algorithme général est le suivant :

1. observer l'état courant i et choisir une action $a\Gamma$
2. exécuter l'action et observer le renforcement r et l'état suivant $\acute{i}\Gamma$
3. appliquer le schéma d'apprentissage renforcé à l'expérience $(i, \acute{i}, a, r)\Gamma$
4. mettre à jour le modèle du monde grâce à $(i, \acute{i}, a, r)\Gamma$
5. répéter k fois :
   - sélectionner un état i et une action a hypothétiquesΓ
   - déterminer grâce au modèle du monde l'état $\acute{i}$ et le renforcement r résultantsΓ
   - appliquer le schéma d'apprentissage renforcé à l'expérience hypothétique $(i, \acute{i}, a, r)\Gamma$
6. reboucler sur l'étape 1.

8.3.3 Apprentissage par explication

Ces travaux se situent dans le cadre de l'apprentissage par Q-Learning de différentes tâches dans un environnement identique [123Γ174]. Le principe est d'exploiter des connaissances du domaine invariantes pour les tâches permettant d'extraire des renseignements sur la forme de la fonction Q. Ces connaissances correspondent à un modèle de l'interaction du robot dans son environnement (cf l'architecture Dyna de Sutton).

Le système peut se déplacer en choisissant une commande a parmi n possibles. A chaque commande a est associé un réseau de neurones N_a chargé de prédire son effet : $I \mapsto I$. Ces réseaux sont tout d'abord entraînés et figés avant de débuter l'entraînement des tâches.

Lors de l'apprentissage d'une tâcheΓle robot atteint son but après une succession d'actions $(a_1, \ldots, a_p)$ associées aux états respectifs $(i_1, \ldots, i_p)$. Il reçoit un renforcement r. Les modèles d'actions appris peuvent alors être utilisés de manière à expliquer et à analyser cet épisode. Expliquer signifie retourner de i_1 à i_p en appliquant à travers les modèles les actions a_i exécutées par le robot (voir figure 8.7). Analyser signifie évaluer la dérivée partielle de Q en i_j par rapport au

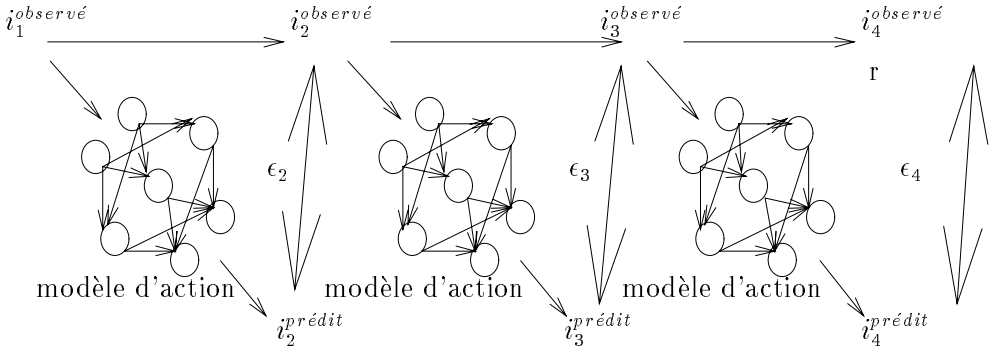


FIG. 8.7 - : *Utilisation du modèle d'action afin d'expliquer un épisode*

renforcement final r (et donc déterminer le rôle de chaque configuration perçue dans le résultat final). Ce calcul de dérivée est rendu possible par l'utilisation de modèles d'action différentiables. La figure 8.8 illustre le gain apporté par l'utilisation de la valeur de la dérivée dans l'apprentissage de la fonction Q.

Remarque : l'utilisation de la pente comme contrainte est pondérée par la précision de cette valeur. Cette précision dépend de la qualité du modèle d'actionΓ évaluée lors de la phase explication en mesurant la différence entre un état prédit et un état observé (voir figure 8.7).

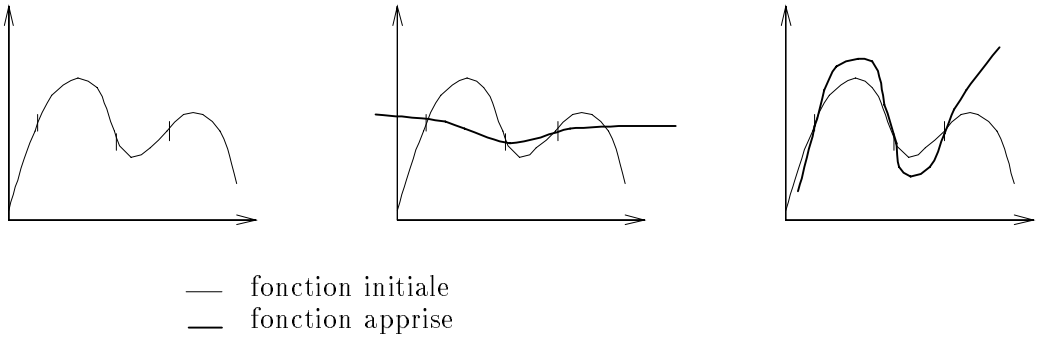


FIG. 8.8 - : *Utilisation de la dérivée afin d'augmenter la vitesse de l'apprentissage. La figure de gauche représente la fonction que l'on cherche à apprendre. La figure centrale correspond à la fonction apprise en utilisant uniquement la valeur en trois points. La figure de droite correspond à la fonction apprise en utilisant la valeur ainsi que la pente en ces points.*

8.3.4 Utilisation de connaissances initiales expertes

La dernière catégorie d'approches auxquelles nous allons nous intéresser a pour objectif de faciliter l'apprentissage en incorporant au système initial des connaissances expertes du domaine. Ces connaissances permettent entre autre de ne pas apprendre de zéro. On dispose immédiatement d'un système plus ou moins opérationnel ne se comportant pas de manière totalement aléatoire. Cela permet également d'assurer la sécurité du véhicule en évitant d'explorer le résultat d'actions que l'on sait de toutes façons dangereuses. Il est par exemple inutile d'essayer une commande de type marche avant si le robot se trouve en face d'un mur à faible distance.

Le système *TESEO*

Le système *TESEO* proposé par Millán est un système neuronal de navigation réactive pour robot mobile basé sur l'apprentissage renforcé [120Γ122]. Le but de ces travaux est de permettre à un robot d'apprendre un comportement réactif lui permettant de rejoindre un but à partir des données capteurs et de sa position relative par rapport à celui-ci. Ce comportement doit être efficace et doit donc tenir compte d'informations globales sur l'environnement.

Un tel comportement ne peut être obtenu de manière purement réactive. Une solution couramment utilisée consiste à construire une carte de la zone d'évolution afin de pouvoir réaliser une phase de planification avant exécution (voir chapitre 3). Millán propose en revanche de ne pas séparer le problème en deux phases et d'apprendre directement un comportement adaptéΓintégrant en quelque sorte une représentation interne du milieu.

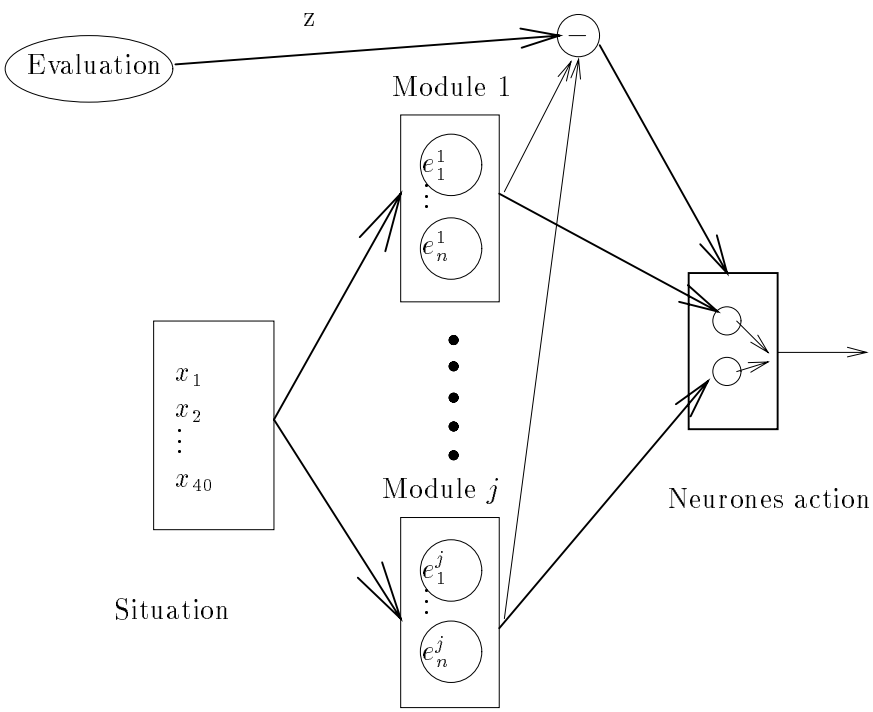


FIG. 8.9 - : *Architecture neuronale de TESEO*

L'architecture neuronale proposée par Millán permet de réaliser un codage de règles perception $\mapsto$ action (voir figure 8.9). Les neurones exemples ont pour tâche de classifier les données d'entrée. Ils sont regroupés par moduleΓchaque module étant composé d'un ensemble de neurones *exemples* traitant la même zone de l'espace d'entrée et proposant une réponse similaire.

Deux cas peuvent se produire lors de la présentation des mesures capteurs :

1. Un ou plusieurs modules possèdent des neurones exemples actifs. Cela signifie que la situation courante fait partie du domaine de compétence du réseau. Le module dont les neurones exemples sont les plus activés est sélectionné. La couche action fournit la commande résultante en perturbant la commande par défaut associée à ce module (exploration).

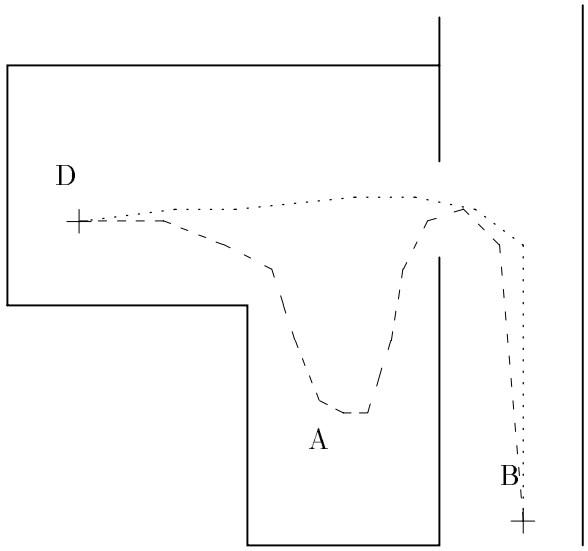
2. Aucun module n'est activé. On se trouve dans une situation nouvelle pour laquelle le réseau ne peut répondre. Le robot possède alors un certain nombre d'actions réflexes capables de fournir une commande appropriée à la situation immédiate. Une nouvelle règle associant la situation courante à l'action réflexe est créée. Si elle est suffisamment proche d'un ensemble

de règles existantes, elle est alors intégrée au module correspondant. Dans le cas contraire, un nouveau module est également créé.

Les différents paramètres associés au réseau et que nous n'avons pas détaillé ici sont mis à jour grâce aux algorithmes $TD(\lambda)$ et *Associative Search* de l'apprentissage renforcé.

Le principe général de cette approche est de partir d'un ensemble d'actions réflexes capables d'assurer la survie du robot et d'exécuter de manière très inefficace la tâche souhaitée. Ce module réflexe pilote initialement le robot en permanence. Au fur et à mesure que l'apprentissage progresse, il est de moins en moins activé, le relais étant pris par le système neuronal. Les connaissances contenues dans les actions réflexes sont transférées dans le réseau et améliorées par apprentissage.

L'environnement d'expérimentation est représenté figure 8.10. Après une dizaine d'essais à partir du point D, le robot apprend seul à rejoindre directement son but sans plus chercher à pénétrer dans la zone A. Les capacités de généralisation du contrôleur obtenu permettent également au robot de rejoindre son but à partir d'une très grande variété de points situés dans la pièce ou dans le couloir.



- - - Exemple de trajectoire avant apprentissage

····· Exemple de trajectoire après apprentissage

FIG. 8.10 - : *Environnement d'expérimentation. Le robot doit rejoindre son but à partir du point B. Au bout de quelques essais, il apprend à ne plus pénétrer dans la zone A.*

Le système *GARIC*

Le système *GARIC* proposé par Berenji [21Γ22] est un autre exemple d'utilisation de connaissances initiales dans un schéma d'apprentissage renforcé. La connaissance initiale est fournie aussi bien pour le contrôleur que pour le module de critique heuristique adaptatif. De manière à permettre l'apprentissageΓ ces deux systèmes flous sont tout d'abord transformés en réseaux de neurones équivalents (voir section 8.6).

Les approches présentées ci-dessus ne sont que quelques représentants d'un ensemble très riche de travaux dans ce domaine. Elles permettent néanmoins d'indiquer quelques grandes directions de recherche. Nous allons maintenant préciser l'axe que nous avons choisi et le problème particulier que nous avons étudié.

8.4 Choix de l'approche retenue

Comme nous l'avons indiqué au cours du chapitre 2Γun robot mobile est un système complexe et bruité pour lequel on ne possède pas de modèles fiables et dont l'apprentissage d'un tel modèle nous semble être délicat (Mitchell apprend un modèle de comportement de son robot mais le nombre possible d'actions est limité [123]). Comme le rappelle Gullapalli [75]Γil est préférable dans un tel contexte d'avoir recours à des techniques d'apprentissage renforcé direct.

De plusΓnous avons étudié au cours des chapitres 4 et 5 un certain nombre d'approches permettant au véhicule dans certaines situations d'atteindre son but tout en évitant les obstacles. Ces approches peuvent servir de solution initiale se renforçant grâce à l'apprentissage.

Nous nous sommes donc tournés vers une solution de type *connaissance initiale*. Pour des raisons de facilité de codage et d'édition de cette connaissance ainsi que pour des raisons de possibilités d'évolutionΓnous avons retenu comme support la logique floue. Un exemple d'apprentissage à partir de champs de potentiels peut être trouvé dans [41].

Un système d'apprentissage basé sur une méthode de type *Q-Learning* regroupe en une seule fonction l'utilité d'une action ainsi que la loi de commande. Nous avons préféré utiliser une solution de type apprentissage renforcé associatif permettant de séparer les deux problèmes. La loi de commande apprise $e(s,i)$ peut ainsi être directement extraite et utilisée dans un contexte hors apprentissage.

Ce type d'approche nécessite la mise en place de deux modules distincts :

- le module d'apprentissage du critique heuristique adaptatifΓ
- le module d'apprentissage de la loi de commande.

Nous nous sommes plus particulièrement intéressés au premier problème : l'apprentissage $TD(\lambda)$ appliqué à un système flou.

Remarque : il nous semble que l'utilisation de connaissances initiales pour la loi de commande ainsi que pour le critique nécessite la mise en place de précautions particulières lors de la mise en œuvre du système. Le rôle de l'apprentissage renforcé est de générer une loi de commande maximisant à chaque étape le renforcement prédit. Si la fonction de critique prédictif n'est pas "adaptée" à la loi de commande fournieΓelle aura pour effet de détruire celle-ci et donc de perdre la connaissance initiale. Il nous semble que ce phénomène peut être évité en interdisant dans un premier temps l'apprentissage de la loi de commande et en laissant le critique prédictif s'adapter de manière à prédire le comportement initial du robot. Une fois le prédictif stabiliséΓl'apprentissage peut être rétabli pour la loi de commande.

8.5 Formulation du problème

Nous supposerons que le robot est piloté par le système de navigation flou exposé section 5.3. Sa tâche est de rejoindre un ensemble de buts disposés de manière aléatoire. Les renforcements générés par l'environnement sont les suivants :

- $r = 1$ si le but est atteint (la contrainte est sur la position du robot et non son orientation pouvant être quelconque)Γ
- $r = -1$ si le robot entre en collision avec un obstacleΓ
- $r = -1$ si le robot est arrêté à un emplacement autre que le but (minimum local)Γ
- $r = -1$ si le robot oscille sur place (vitesse de translation nulle et vitesse de rotation changeant périodiquement de signe).

Le problème est de déterminer grâce à l'algorithme $TD(\lambda)$ un ensemble de règles floues permettant de prédire le renforcement total amorti reçu par le robot au cours de ses déplacements.

Revenons à l'équation 8.6 section 8.1.2 :

$$\vec{\Delta} w_t = \alpha(r_{t+1} + \gamma P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k$$

Cette équation a été obtenue en considérant la minimisation de l'erreur de prédiction fournie par une fonction P dépendant d'un ensemble de paramètres $\vec{w}$. Le principe général que l'on retrouve derrière cette équation et illustré par Sutton

est le suivant [166] :

- si la prédiction est correcteΓon obtient :

$$\begin{aligned} & P_t = z_t \\ \Rightarrow\ & P_t = \sum_{k=0}^{+\infty} \gamma^k r_{t+k+1} \\ & \quad = r_{t+1} + \gamma \sum_{k=0}^{+\infty} \gamma^k r_{t+k+2} \\ & \quad = r_{t+1} + \gamma P_{t+1} \end{aligned}$$

- l'erreur de prédiction vaut donc : $\epsilon = r_{t+1} + \gamma P_{t+1} - P_t$ (terme central de l'équation 8.6).

- cette erreur doit être appliquée à l'état courant mais doit également être propagée aux états passés. Cette propagation permet d'associer un état avec un renforcement reçu avec retard. De manière à limiter la remontée dans le tempsΓon associe à chaque état un degré d'activation décroissant de manière exponentielle : $\lambda^{(t-t_0)}\Gamma t_0$ correspondant à l'heure où l'état a été visité. Cette propagation dans le temps de l'erreur de prédiction et cette décroissance de l'activation des états passés se retrouve dans la somme des gradients de l'équation 8.6.

Ce principe a été illustré figure 8.2 dans le cas de $TD(0)$. L'algorithme général est le suivant. Soit f la fonction de prédiction réalisée grâce à un ensemble de règles floues. On maintient une liste $l_{états}$ d'états visités avec leur degré d'activation. A chaque cycle :

1. on décrémente le degré d'activation de chaque état de la liste $l_{état}$ en le multipliant par λ. On retire ceux dont le degré d'activation est inférieur à un seuil fixe.

2. on détermine l'état courant i_t ainsi que le renforcement associé r_t.

3. on détermine l'erreur de prédiction par rapport à l'état précédent grâce à la formule :
$$\epsilon = \alpha(r_t + \gamma f(i_t) - f(i_{t-1}))$$

4. pour chaque élément i de la liste $l_{état}$Γon modifie la fonction f de telle sorte que :
$$f(i) \leftarrow f(i) + activation(i) * \epsilon$$

L'apprentissage proprement dit est effectué au cours de l'étape 4 de l'algorithme. Il s'inscrit dans le cadre d'un apprentissage supervisé : la valeur de la fonction f au point i est fournie.

De part la nature de notre problème, l'algorithme retenu pour effectuer cette tâche dans le cas d'un système flou devra présenter les propriétés suivantes :

- être incrémental. Les différents exemples sont fournis au système au fur et à mesure que le robot se déplace et explore son espace d'entrée. Un apprentissage de type renforcé pouvant être actif durant la totalité de la vie du robot (de manière à adapter le comportement du véhicule à toute situation nouvelle), la totalité des exemples ne peut être stocké.

- être robuste face aux données incohérentes. De part le principe même de l'apprentissage du critique prédictif, la valeur de la fonction f en tout point i sera modifiée jusqu'à convergence du système. Les exemples présentés seront donc formés de couples (i, x_0) et (i, x_1) avec $x_0 \neq x_1$.

Ayant précisé les spécifications désirées pour l'algorithme d'apprentissage flou supervisé, nous allons maintenant revenir sur les principaux travaux existants dans ce domaine.

8.6 L'apprentissage en logique floue

Les premiers travaux dans le domaine de l'apprentissage pour systèmes flous ont été réalisés par Procyk et Mamdani [137] dans le cadre des contrôleurs flous auto-organisés[3]. Le principe général utilisé est celui de l'apprentissage renforcé. Le but est d'apprendre à un contrôleur flou à suivre une trajectoire de référence (ou atteindre un point). Le résultat de ses commandes est jugé grâce à une table de performance permettant d'indiquer les corrections nécessaires. Ces corrections sont alors répercutées sur les différentes règles en tenant compte de leur rôle pour le calcul de la commande finale.

Récemment, Glorennec [68] a proposé une adaptation de l'algorithme Q-Learning à un contrôleur flou. Le contrôleur est formé d'un ensemble d'agents acceptant tous les mêmes entrées. Pour chaque situation, la sélection de l'agent actif est réalisée grâce à la fonction d'utilité Q. Le renforcement reçu est alors distribué sur ses différentes règles en fonction de leur degré d'activation. Après apprentissage, il est possible de former un nouvel agent constitué des règles de chaque agent ayant reçu le plus de renforcement. Cet algorithme ne propose pas la modification ou la création de nouvelles règles mais de réaliser une sélection des règles les plus appropriées parmi une base existante.

Les deux approches précédentes ont pour principe commun d'essayer de construire une base de règles par apprentissage renforcé (le schéma de renforcement étant élémentaire dans le cas de Mamdani). Elles ne répondent pas à nos objectifs.

3. En anglais : Self Organizing Controller ou SOC

Comme le souligne Lee [111]Γil existe deux types d'apprentissage pour un système flou :

1. l'apprentissage de paramètres. Il permet de régler la valeur des différents paramètres attachés aux fonctions d'appartenance et (ou) aux opérateurs de combinaison.

2. l'apprentissage structurel. Son but est de déterminer le nombre de règles ainsi que les variables mises en jeu.

Nous allons revenir sur ces deux aspects.

8.6.1 L'apprentissage des paramètres

Le but de cet apprentissage est de déterminer la valeur des différents paramètres utilisés de manière à minimiser l'erreur commise par rapport aux exemples. Ces paramètres peuvent être attachés aux opérateurs de combinaison [69] ou aux fonctions d'appartenance des différentes données linguistiques. L'approche la plus simple consiste à positionner les différentes parties condition des règles manuellement ou grâce à un algorithme de catégorisation et d'ajuster uniquement la partie conclusion (voir [90] par exemple). Rappelons que dans le cadre d'un contrôleur de type Sugeno (voir section 5.2.4) avec une partie conclusion constanteΓla valeur de sortie est donnée par :

$$f = \frac{\sum_i w_i a_i}{\sum_i w_i}$$

La valeur de sortie est donc une combinaison linéaire des termes recherchés. La minimisation de l'erreur peut être obtenue directement par une méthode de type simplexe [66] ou descente de gradient ([83] par exemple).

La modification exclusive de la partie droite peut être insuffisante. Il faut alors avoir recours à une adaptation de l'ensemble des paramètres du système flou. Cela peut être réalisé grâce à différentes techniques d'optimisation telles que le recuit simulé [108] ou à nouveau la descente de gradient ([73] par exemple).

Le calcul du gradient de chaque paramètre intervenant dans la partie gauche d'une règle est un problème a priori non trivial. On peut constater néanmoins que les opérations réalisées dans les différentes étapes de l'évaluation d'une base de règles sont similaires aux opérations réalisées par certains neurones formels. Un système flou peut être transformé en un réseau à propagation unilatérale et être modifié par rétro-propagation du gradient.

Le rapprochement de la logique floue et des réseaux de neurones afin de combiner leurs avantages respectifs est une grande préoccupation des chercheurs des deux axes. Ces formalismes sont tous les deux utilisés pour résoudre des problèmes semblables caractérisés par une absence de modèle. Le contrôle flou permet la prise en compte de connaissances initiales mais reste figé. Les réseaux de neurones ne peuvent incorporer cette connaissance initiale mais sont capables d'adaptation. Le lecteur intéressé pourra se reporter à [126Γ67Γ69Γ86Γ127].

8.6.2 L'apprentissage structurel

Le but de cet apprentissage est de déterminer la structure d'un contrôleur décrivant une base d'exemples fournie. Les méthodes développées peuvent être séparées en deux catégories :

- celles basées sur l'analyse de la distribution des points exemplesΓ
- celles basées sur la recherche du "meilleur" contrôleur parmi l'ensemble des contrôleurs possibles.

Analyse de la distribution des points

Les exemples fournis pour l'apprentissage sont constitués d'un ensemble de couples de points *(entrée, sortie)*. Chacun de ces couples peut trivialement être considéré comme une règle élémentaire mais cela conduit rapidement à un contrôleur final trop complexe. Le principe de l'analyse de la distribution des points est de regrouper au maximum les exemples de manière à réduire le nombre final de règles.

Cette analyse des exemples peut avoir pour rôle de simplifier le contrôleur complet (une règle par exemple) ou au contraire de ne générer que les règles nécessaires. Dans le cadre de la première approcheΓon peut citer par exemple les travaux d'Arciniegas [10] basés sur les travaux de Chen [39]. Le principe est de considérer le système flou comme un réseau de neurones de type *Radial Basis Function* (voir section 7.2.1). A chaque exemple est associé une fonction de base. Ces fonctions sont ensuite orthogonalisées de manière à être en mesure de déterminer le rôle de chacune d'entres elles dans la détermination de la valeur finale. Celles dont le rôle est le plus faible sont supprimées Γréduisant ainsi le nombre de règles. Les résultats obtenus ne semblent pas très satisfaisants Γ Arcieniegas rapportant un nombre final de règles de l'ordre de 80% du nombre d'exemples.

Dans le cadre de la seconde approche (génération du nombre nécessaire de règles)Γle principe général est de réaliser un regroupement des exemples à l'aide d'un algorithme de classification[4]. Kosko [103] par exemple réalise une catégorisation de l'ensemble des exemples représentés comme des points dans l'espace produit *Entrée* $\times$ *Sortie*Γchaque catégorie créée représentant une règle. La plupart des autres méthodes considèrent le lien entre un système flou et un réseau de type *Radial Basis Function*. Katayama [94] part d'un ensemble de règles initiales traduites sous forme d'un réseau. Il modifie ensuite les différents paramètres (y compris la position des sites récepteurs et donc la zone d'activation des règles) grâce à un algorithme de descente de gradient. Lorsque les corrections calculées deviennent trop faiblesΓun nouveau site (et donc une nouvelle règle) est créée à partir de l'exemple le moins bien appris. Cette algorithme nécessite la connaissance initiale de l'ensemble des exemples et n'est donc pas incrémental.

4. En anglais : clustering

Wang [182] propose de réaliser une classification basée sur la distance entre les points dans l'espace d'entrée. Une nouvelle règle est créée si le nouvel exemple est situé trop loin des exemples précédents. Nie [128] propose plutôt de baser la classification sur le degré d'activation des règles de manière à n'adapter que la règle la plus active et de ne créer une nouvelle règle que si aucune parmi celles existantes n'est suffisamment activée. Cet algorithme impose par contre un support identique pour l'ensemble des données linguistiques en partie condition des règles et l'utilisation de nombres non flous en partie conclusion.

Sélection du contrôleur parmi un ensemble

Le principe général de ces méthodes est de réaliser une recherche au sein de l'espace formé par l'ensemble des contrôleurs possibles. A notre connaissanceΓla première approche proposée dans ce domaine est celle de Takagi et Sugeno [169]. Le but est de déterminer les variables intervenant dans le contrôleurΓles parties conditions et les parties conclusion des règles. Le principe de l'algorithme est le suivant :

1. un sous-ensemble de l'ensemble total des variables du problème est choisi grâce à une heuristique que nous ne détaillerons pas ici. A partir de cet ensembleΓla partie condition et conclusion optimales sont calculées grâce aux pas 2 et 3 de l'algorithme. Le contrôleur obtenu est évalué par rapport à l'ensemble des exemples (par un critère quadratique d'erreur). Le choix initial du sous-ensemble de variables est alors remis en cause de manière à réduire cette erreur.

2. étant donné un choix de variablesΓles parties conditions optimales peuvent être déterminées (en tenant compte des parties conclusions calculées par le pas 3 de l'algorithme).

3. étant donné un choix de variables et de parties conditionsΓles différentes conclusions sont générées afin de réduire l'erreur quadratique par rapport aux exemples.

La détermination d'un contrôleur nécessite donc de nombreuses itérations entre les trois étapes rendant l'algorithme peu adapté à un calcul en ligne. Il est de plus nécessaire de connaître initialement la totalité des exemples.

Le problème de recherche d'un élément parmi un ensemble vaste d'éléments possibles est un problème type des algorithmes génétiques. De nombreux travaux de sont intéressés à cette voie [112Γ46Γ45Γ70Γ80Γ81Γ79Γ34]. Herrera par exemple [79] code une règle floue au sein d'un chromosome destiné à évoluer. La qualité de chaque chromosomeΓpermettant de sélectionner les candidats au croisementΓest jugée selon 4 critères :

– le nombre d'exemples positifs. Ce sont les exemples de la base initiale en accord avec la règle.

- le degré d'activation moyen de la règle sur l'ensemble des exemples positifs.
- le nombre d'exemples négatifs. Ce sont les exemples en contradiction avec la règle.
- le degré de participation de la règle sur la totalité des exemples

Le principe de construction d'un contrôleur est le suivant :

1. Un ensemble de chromosomes initiaux sont fournis.
2. L'algorithme génétique est appliqué à partir de ces chromosomes et de la base de points exemples (cette base est nécessaire pour le calcul de la fonction d'évaluation).
3. le meilleur chromosome est sélectionné et la règle correspondante est ajoutée au contrôleur en construction.
4. les éléments de la base d'exemples couverts par la nouvelle règle sont supprimés.
5. Si la base d'exemple est vide l'algorithme est terminé. Sinon on retourne au pas 2.

Le problème posé par les approches génétiques est qu'elles ne semblent pas adaptées aux problèmes d'apprentissage en ligne incrémentaux.

La plupart des algorithmes proposés dans le cadre de l'apprentissage structurel ou de l'apprentissage de paramètres sont basées sur le principe suivant :

- le système flou est converti en un réseau de neurones équivalent au travers une phase de compilation
- le réseau est entraîné à partir des exemples
- après apprentissage il est reconverti en règles floues par une phase de décompilation.

Les architectures neuronales pouvant être directement déterminées à partir des règles sont des réseaux à propagation unilatérale mal adaptés à nos contraintes particulières d'incrémentalité et de structure non rigide.

Ces deux propriétés requises sont par contre une des caractéristiques des systèmes de type *ARTMAP* [33]. De plus la représentation de l'espace d'entrée et de sortie que ces réseaux proposent sous forme de catégories est assez similaire au découpage de l'espace par des données linguistiques. La comparaison s'arrête toutefois ici. Le principe général des réseaux ART est basé sur l'algorithme *Winner Takes All* opposé au contrôle flou où toutes les règles sont actives simultanément.

De manière à répondre à nos spécifications nous avons proposé un algorithme d'apprentissage pour systèmes flous s'inspirant des règles de modification des catégories dans un réseau *ART*. Nous allons maintenant le présenter.

8.7 Apprentissage supervisé incrémental de règles floues

L'algorithme d'apprentissage que nous avons développé [142] est basé sur une manipulation directe des règles floues et de leurs composantes. Il ne nécessite pas de transformations préliminaires du système en réseau de neuronesΓni de transformation inverse en fin d'apprentissage.

Le principe de l'algorithme est de considérer l'hyper-surface (si la dimension d'entrée du problème est 2Γon parle alors de surface) décrite par le système flou comme une surface élastique pouvant être déformée sous les diverses contraintes provenant des exemples. Soit f une fonction décrite par un ensemble de règles floues. Soit (x_0, y_1) un nouvel exemple. Si avant sa présentationΓon a $f(x_0) = y_0$ avec $y_1 \neq y_0$Γle système va devoir apprendre. Il va pour ce faire "appliquer une pression" sur la surface en x_0 de manière à ce qu'elle se déforme de y_0 vers y_1. L'élasticité de la surface détermine si cette déformation est de nature locale ou non.

Nous considérerons les hypothèses suivantes (se reporter section 5.2.4 pour plus de précisions) :

- le contrôleur utilisé est de type LarsenΓ
- le connectif *AND* est réalisé par l'opérateur *min*Γ
- le connectif *ALSO* est réalisé par l'opérateur *somme normalisée*Γ
- le module de codage transforme les données d'entrée en singletons flous (pas de "fuzzyfication")Γ
- le module de décodage utilise l'approche *center of area*Γ
- les données linguistiques utilisées sont codées uniquement grâce à des fonctions *exponentielles* ou *trapèze-exponentielles* (voir figure 8.11)Γ
- chaque règle ne possède qu'une et une seule conclusion. Si une règle nécessite m conclusions différentesΓon réécrit alors la règle en m règles distinctes.

Ces hypothèses sont très classiques et s'appliquent à une grande quantité de contrôleurs flous traditionnellement utilisés.

La méthode de décodage employée est l'approche *center of area*. La valeur rendue par un système flou comportant n règles est donc :

$$f = \frac{\sum_{i=1}^{n} \int x \mu_i(x) dx}{\sum_{i=1}^{n} \int \mu_i(x) dx} \tag{8.10}$$

où $\mu_i(x)$ représente la fonction d'appartenance de la partie conclusion de la règle i **après** évaluation. Le contrôleur utilisé étant de type LarsenΓl'opérateur d'inférence est le produit (rappelons que l'opérateur d'inférence permet de reporter en

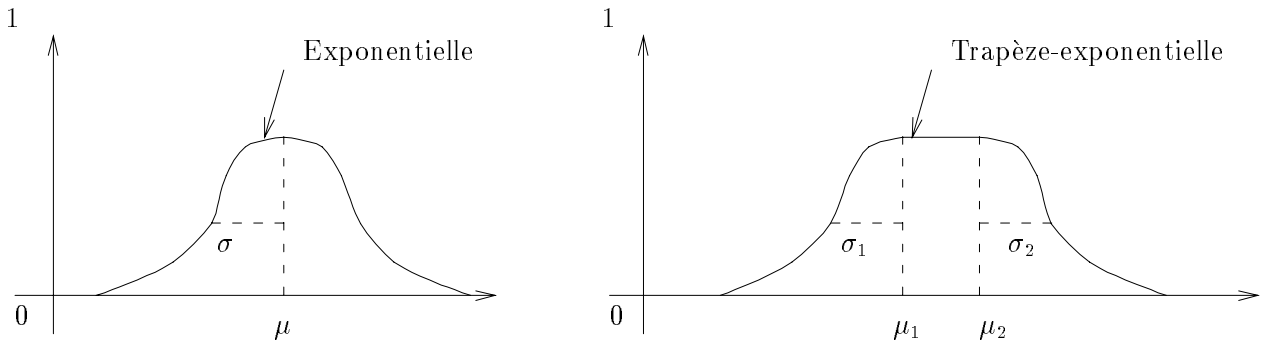


FIG. 8.11 - : *Définition des fonctions d'appartenance valides pour les données linguistiques.*

partie conclusion le degré d'activation de la partie condition de la règle). L'équation 8.10 peut se réécrire :

$$f = \frac{\sum_{i=1}^{n} w_i \int x \mu_{c_i}(x) dx}{\sum_{i=1}^{n} w_i \int \mu_{c_i}(x)} \tag{8.11}$$

où $\mu_{c_i}(x)$ représente la fonction d'appartenance de la partie droite **avant** évaluation et w_i le degré d'activation de la règle. Conformément aux hypothèses que nous avons énoncées précédemment μ_{ci} correspond soit à une fonction exponentielle soit à une fonction trapèze-exponentielle.

Si μ_{c_i} est une fonction exponentielle l'intégrale présente au dénominateur de l'équation 8.11 s'écrit :

$$\int_{-\infty}^{+\infty} e^{\frac{-(x-\mu)^2}{\sigma^2}} dx = \sqrt{\pi}\sigma$$

L'intégrale présente au numérateur s'écrit :

$$\begin{aligned} &\int_{-\infty}^{+\infty} x e^{\frac{-(x-\mu)^2}{\sigma^2}} dx \\ &= \int_{-\infty}^{+\infty} (y+\mu) e^{\frac{-y^2}{\sigma^2}} dy \\ &= \sqrt{\pi}\mu\sigma \end{aligned}$$

Si μ_{c_i} est une fonction trapèze-exponentielle l'intégrale présente au dénominateur de l'équation 8.11 s'écrit :

$$\begin{aligned} &\int_{-\infty}^{\mu_1} e^{\frac{-(x-\mu_1)^2}{{\sigma_1}^2}} dx + \int_{\mu_1}^{\mu_2} dx + \int_{\mu_2}^{+\infty} e^{\frac{-(x-\mu_2)^2}{{\sigma_2}^2}} dx \\ &= \frac{\sqrt{\pi}}{2}\sigma_1 + \frac{\sqrt{\pi}}{2} + \mu_2 - \mu_1 \end{aligned}$$

L'intégrale présente au numérateur s'écrit :

$$\begin{aligned}
&\int_{-\infty}^{\mu_1} x e^{\frac{-(x-\mu_1)^2}{{\sigma_1}^2}} dx + \int_{\mu_1}^{\mu_2} x dx + \int_{\mu_2}^{+\infty} x e^{\frac{-(x-\mu_2)^2}{{\sigma_2}^2}} dx \\
&= \int_{-\infty}^{0} (y+\mu_1) e^{\frac{-y^2}{{\sigma_1}^2}} dy + \frac{{\mu_2}^2}{2} - \frac{{\mu_1}^2}{2} + \int_{0}^{+\infty} (y+\mu_2) e^{\frac{-y^2}{{\sigma_2}^2}} dy \\
&= \mu_1\sigma_1\frac{\sqrt{\pi}}{2} + \mu_2\sigma_2\frac{\sqrt{\pi}}{2} + \frac{{\mu_2}^2}{2} - \frac{{\mu_1}^2}{2} - \frac{{\sigma_1}^2}{2}[e^{-\frac{y^2}{{\sigma_1}^2}}]_{-\infty}^{0} + \frac{{\sigma_2}^2}{2}[e^{-\frac{y^2}{{\sigma_2}^2}}]_{0}^{+\infty} \\
&= \mu_1\sigma_1\frac{\sqrt{\pi}}{2} + \mu_2\sigma_2\frac{\sqrt{\pi}}{2} + \frac{{\mu_2}^2}{2} - \frac{{\mu_1}^2}{2} - \frac{{\sigma_1}^2}{2} + \frac{{\sigma_2}^2}{2}
\end{aligned}$$

L'équation 8.11 peut se réécrire sans termes intégrales comme une fonction de σ et μ. Dans le cas par exemple où toutes les données linguistiques sont exprimées sous forme d'exponentiellesΓla fonction f représentée par les règles floues s'écrit :

$$f = \frac{\sum_{i=1}^{n} w_i \mu_i \sigma_i}{\sum_{i=1}^{n} w_i \sigma_i} \tag{8.12}$$

Reprenons le cas énoncé au début de cette section. Un nouvel exemple (x_0, y_1) est présenté au système alors que les règles floues actuelles réalisent $f(x_0) = y_0$ avec $y_0 \neq y_1$. L'équation 8.12 doit être modifiée afin de prendre en compte cette nouvelle donnée. Nous avons considéré trois possibilités :

1. un nouveau terme est ajouté au numérateur et au dénominateur de manière à modifier la sortie de y_0 en y_1Γ

2. aucun terme n'est ajouté mais un couple existant (μ_i, σ_i) est modifiéΓ

3. aucun terme n'est ajouté mais une valeur w_i existante est augmentée.

La première modification envisagée correspond à la création d'une nouvelle règle. La seconde modification consiste à adapter la partie conclusion d'une règle déjà existante. La troisième solution enfin revient à modifier le degré d'activation d'une règle afin de la rendre plus sensible à la situation courante (on accepte uniquement d'augmenter w_i et non de le diminuer afin d'éviter des problèmes d'oscillations). Cette modification est obtenue en augmentant la zone d'influence de la partie condition de la règle et en réalisant donc une généralisation.

Comme nous l'avons mentionné précédemmentΓun des buts principaux nous ayant conduit à la création de cet algorithme est de pouvoir disposer d'un système d'apprentissage incrémental. Chaque modification apportée au système ne doit pas mener à l'oubli de modifications précédentes et doit donc rester locale. Lorsqu'une nouvelle règle est crééeΓelle correspond à l'exemple présenté. Tant que cette règle reste localeΓelle peut alors être susceptible d'être adaptée (opération 2) ou d'être généralisée (opération 3). Au fur et à mesure que sa zone d'influence s'étendΓla seule opération restant possible est la généralisation.

Nous allons maintenant revenir plus en détails sur ces trois opérations.

8.7.1 Création d'une nouvelle règle

La création d'une nouvelle règle a pour but de prendre en compte un nouveau cas particulier (x_0, y_1) en contradiction avec les valeurs actuelles du système. Dans le cas d'un réseau de type ARTMAP, il s'agit de créer une nouvelle catégorie au sein de la couche d'entrée correspondant à x_0 et de l'associer à la catégorie de la couche de sortie contenant x_1. Dans le cadre de notre système, la création d'une nouvelle règle nécessite la détermination de sa partie condition, centrée sur x_0, et de sa partie conclusion.

Détermination de la partie condition

Ne possédant pas d'informations a priori quant au rôle de chacune des variables du système dans la détermination de la valeur y_1 au point x_0, on définit la partie condition de la règle comme la conjonction de l'ensemble des variables du problème : *if* X_1 *is* A_1 *and* X_2 *is* A_2 *and* ... *and* X_m *is* A_m. De manière à garder la propriété de localité, les différents termes linguistiques A_i seront décrits à l'aide de fonctions exponentielles (paramètres (μ_i, σ_i)). La nouvelle règle étant centrée sur l'exemple, on obtient directement :

$$\begin{cases} \mu_1 = x_1 \\ \mu_2 = x_2 \\ \vdots \\ \mu_m = x_m \end{cases}$$

Les paramètres σ_i sont responsables de la localité de la règle. Si on considère à nouveau l'analogie de la surface élastique, un paramètre σ important signifie que la règle aura une grande zone d'influence sur l'espace d'entrée et que la déformation de la surface sera donc importante. σ_i est obtenu grâce à la formule :

$$\sigma_i = \begin{cases} \gamma_{max}\Delta_l & \text{si } |y_1 - y_0| > \gamma_{max}\Delta_r \\ \gamma\Delta_l \frac{|y_1 - y_0|}{\Delta_r} & \text{autrement} \end{cases} \tag{8.13}$$

où

- Δ_l représente l'amplitude maximum des données décrites par A_i.
- Δ_r représente l'amplitude maximum des données décrites par la partie conclusion de la règle.
- γ_{max} représente le pourcentage maximum de Δ_l autorisé pour la zone d'influence de la règle créée.
- γ représente le pourcentage standard de Δ_l autorisé pour la zone d'influence de la règle créée.

La largeur de la donnée linguistique est donc proportionnelle à l'erreur commise ($|y_1 - y_0|$) ramenée entre 0 et 1 grâce au facteur Δ_r. Le coefficient de proportionnalité est un pourcentage fixe de la valeur maximum pouvant être prise (Δ_l). De manière à éviter une valeur trop importante$\Gamma\sigma_i$ est bornée à un pourcentage maximum de Δ_l : γ_{max}.

La partie condition étant maintenant totalement définie. La prochaine étape consiste à définir la partie conclusion.

Détermination de la partie conclusion

La partie conclusion de la règle est exprimée sous la forme *Y is B*. *B* est décrit grâce aux deux paramètres (μ_B, σ_B) vérifiant l'équation :

$$\frac{a + w\sigma_B\mu_B}{b + w\sigma_B} = y_1 \tag{8.14}$$

où a et b correspondent respectivement au numérateur et au dénominateur de l'équation 8.11 au point x_0 (on a donc la relation $\frac{a}{b} = y_0$). Plusieurs remarques peuvent être faites :

- Comme nous l'avons indiqué précédemmentΓla nouvelle règle créée est centrée sur le vecteur x_0. Son degré d'activation est maximum en ce point. w peut être remplacé par 1 dans l'équation 8.14.

- l'ensemble des valeurs du support de B sont susceptibles d'être la commande y résultat de l'évaluation de l'ensemble des règles. Ce support doit donc être inclus dans l'intervalle $[\mu_0, \mu_1]$ des valeurs permises pour y. Rappelons que le support de B est l'ensemble des points tels que leur degré d'appartenance à B est non nul.

- σ_B est un réel positif.

Les contraintes énoncées ci-dessus conduisent au système d'inéquations non-linéaires suivant dont les inconnues sont μ_B et σ_B :

$$\begin{cases} \mu_B + \alpha\sigma_B < \mu_1 \\ \mu_B - \alpha\sigma_B > \mu_0 \\ \sigma_B > 0 \\ \frac{a+\mu_B\sigma_B}{b+\sigma_B} = y_1 \end{cases} \tag{8.15}$$

avec $\alpha = \sqrt{\frac{-1}{\log\epsilon}}$ le réel tel que le support de B soit de dimension $2\alpha\sigma_B\Gamma$la fonction exponentielle étant considérée comme nulle si sa valeur est inférieure à ϵ. De manière à simplifier les notations$\Gamma\mu_B$ et σ_B seront dorénavant remplacées par μ et σ respectivement.

L'équation 8.14 peut être réécrite :

$$\mu = \frac{y_1(\sigma + b) - a}{\sigma} \tag{8.16}$$

En remplaçant μ par cette expression dans les deux premières inéquations du système 8.15Γon obtient :

$$\begin{cases} -\alpha\sigma^2 + \alpha(\mu_1 - y_1) + a - y_1 b \geq 0 \\ -\alpha\sigma^2 + \alpha(y_1 - \mu_0) + y_1 b - a \geq 0 \\ \sigma > 0 \end{cases}$$

Le coefficient des termes carrés est négatif. Ces deux inéquations ne pourront alors être positives pour certaines valeurs de σ que si et seulement si elles admettent deux racines (éventuellement confondues) . De plusΓces deux inéquations ne pourront être vérifiées simultanément que si les intervalles créés respectivement par les deux racines des deux expressions sont d'intersection non nulle (voir figure 8.12)

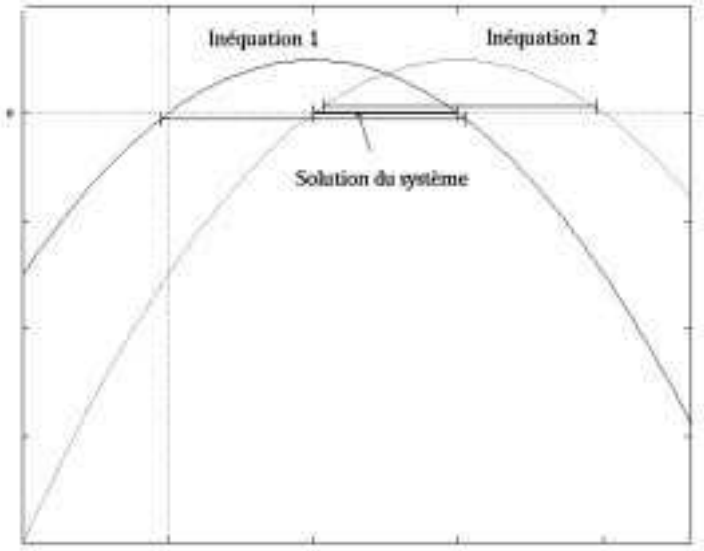


FIG. 8.12 - : *Détermination de la zone solution des deux inéquations*

Les deux expressions quadratiques possèdent deux racines si et seulement si :

$$\begin{cases} \Delta_1 = (\mu_1 - y_1)^2 + 4\alpha(a - y_1 b) \geq 0 \\ \Delta_2 = (y_1 - \mu_0)^2 + 4\alpha(y_1 b - a) \geq 0 \end{cases}$$

Il existe plusieurs possibilités :

1. $\Delta_0 < 0$ et $\Delta_1 < 0$. Les deux expressions n'admettent pas de solutions. Il est impossible de sélectionner un couple (μ, σ) tel que le support de B

soit compris dans l'intervalle $[\mu_0, \mu_1]$. On relâche donc cette condition tout en continuant à imposer $\mu \in [\mu_0, \mu_1]$ (μ représente la commande ayant le degré de possibilité le plus élevé et doit si possible être parmi les valeurs permises). La relation entre σ et μ est donnée par :

$$\sigma = \frac{y_1 b - a}{\mu - y_1}$$

σ devant être positifΓle signe de $(y_1 b - a)$ détermine si μ doit être choisi entre μ_0 et y_1 ou entre y_1 et μ_1.

2. $\Delta_0 \geq 0$ et $\Delta_1 < 0$ ou $\Delta_0 < 0$ et $\Delta_1 \geq 0$. Ces deux situations sont identiques à celle présentée ci-dessus et doivent être résolues de la même manière.

3. $\Delta_0 \geq 0$ et $\Delta_1 \geq 0$. Les deux expressions admettent une ou deux racines. Leur position respective doit être étudiée de manière à pouvoir conclure sur la valeur de σ. Nous les appellerons $\sigma_0^0 \Gamma \sigma_1^0 \Gamma \sigma_0^1$ et σ_1^1 :

$$\begin{cases} \sigma_0^0 = \frac{(y_1 - \mu_1) + \sqrt{\Delta_0}}{-2\alpha} \\ \sigma_1^0 = \frac{(y_1 - \mu_1) - \sqrt{\Delta_0}}{-2\alpha} \\ \sigma_0^1 = \frac{(\mu_0 - y_1) + \sqrt{\Delta_1}}{-2\alpha} \\ \sigma_1^1 = \frac{(\mu_0 - y_1) - \sqrt{\Delta_1}}{-2\alpha} \end{cases}$$

σ devant être positifΓnous étudierons tout d'abord la position de ces racines par rapport à 0. Rappelons que dans le cas d'une équation du second degré $ax^2 + bx + c\Gamma$la somme des racines vaut $-\frac{b}{a}$ et le produit $\frac{c}{a}$. On obtient :

$$\begin{cases} \sigma_0^0 \sigma_1^0 = \frac{y_1 b - a}{\alpha} \\ \sigma_0^0 + \sigma_1^0 = \frac{\mu_1 - y_1}{\alpha} \\ \sigma_0^1 \sigma_1^0 = \frac{a - y_1 b}{\alpha} \\ \sigma_0^1 + \sigma_1^0 = \frac{y_1 - \mu_0}{\alpha} \end{cases} \tag{8.17}$$

α est un réel positif. Seul le signe du numérateur est important. Si le système que l'on souhaite contrôler reste à l'intérieur de ses limites $(y_1 \in [\mu_0, \mu_1])\Gamma$ on peut conclure grâce aux équations 2 et 4 du système 8.17 que si les racines ont le même signeΓelles sont toutes les deux positives. Il existe deux possibilités :

(a) $y_1 b - a \geq 0$. σ_0^0 et σ_1^0 ont le même signe et sont positives. Appelons :

$$\begin{cases} \sigma_{inf}^0 = \sigma_0^0 \\ \sigma_{sup}^0 = \sigma_1^0 \end{cases}$$

On peut également conclure que σ_0^1 et σ_1^1 sont de signes opposés. Ne nous intéressant qu'aux racines positivesΓnous appellerons :

$$\begin{cases} \sigma_{inf}^1 = 0 \\ \sigma_{sup}^1 = \sigma_1^1 \end{cases}$$

(b) $y_1 b - a < 0$. Par un raisonnement similaire on obtient :

$$\begin{cases} \sigma_{inf}^0 = 0 \\ \sigma_{sup}^0 = \sigma_1^0 \\ \sigma_{inf}^1 = \sigma_0^1 \\ \sigma_{sup}^1 = \sigma_1^1 \end{cases}$$

Les deux intervalles $[\sigma_{inf}^0, \sigma_{sup}^0]$ et $[\sigma_{inf}^1, \sigma_{sup}^1]$ contiennent les valeurs de σ satisfaisant nos contraintes. Leur position respective doit être étudiée. Il y a 6 situations possibles :

(a) $\sigma_{inf}^0 < \sigma_{inf}^1 < \sigma_{sup}^1 < \sigma_{sup}^0$. On choisit $\sigma = \frac{\sigma_{inf}^1 + \sigma_{sup}^1}{2}$.

(b) $\sigma_{inf}^0 < \sigma_{inf}^1 < \sigma_{sup}^0 < \sigma_{sup}^1$. On choisit $\sigma = \frac{\sigma_{inf}^1 + \sigma_{sup}^0}{2}$.

(c) $\sigma_{inf}^1 < \sigma_{inf}^0 < \sigma_{sup}^1 < \sigma_{sup}^0$. On choisit $\sigma = \frac{\sigma_{inf}^0 + \sigma_{sup}^1}{2}$.

(d) $\sigma_{inf}^0 < \sigma_{sup}^0 < \sigma_{inf}^1 < \sigma_{sup}^1$. Les deux intervalles sont disjoints. Il n'y a pas de solution. Le couple (μ, σ) est choisi de manière similaire à la situation 1 ($\Delta_0 < 0$ et $\Delta_1 < 0$).

(e) $\sigma_{inf}^1 < \sigma_{sup}^1 < \sigma_{inf}^0 < \sigma_{sup}^0$. Les deux intervalles sont disjoints. Il n'y a donc pas de solution. Le couple (μ, σ) est donc choisi de manière similaire qu'à la situation 1 ($\Delta_0 < 0$ et $\Delta_1 < 0$).

(f) $\sigma_{inf}^1 < \sigma_{inf}^0 < \sigma_{sup}^0 < \sigma_{sup}^1$. On choisit $\sigma = \frac{\sigma_{inf}^0 + \sigma_{sup}^0}{2}$.

σ étant maintenant calculé μ est obtenu simplement grâce à l'équation 8.16.

Une alternative à la création de règles est l'adaptation de la partie conclusion d'une règle existante.

8.7.2 Adaptation d'une règle

Adapter une règle signifie modifier les paramètres (σ, μ) de sa partie conclusion. De manière à conserver la propriété d'incrémentalité cette modification n'est autorisée que si la règle vérifie la propriété de *localité*. Considérons la règle *If* X_1 *is* A_1 *and* ... *and* X_m *is* A_m *then* Y *is* B. Cette règle sera dite locale si et seulement si :

1. $\forall i \in \{1, \ldots, m\}$ les fonctions d'appartenance des données linguistiques A_i sont des exponentielles (et non des trapèze-exponentielles).

2. $$\forall i \in \{1, \ldots, m\} \begin{cases} x_i \in [\mu_i - \alpha\sigma_i, \mu_i + \alpha\sigma_i] \\ \alpha\sigma_i < \rho\Delta_l \end{cases}$$

x_i représente la valeur non floue de la variable X_i. La première partie de la condition permet d'assurer que la règle est active. La seconde partie indique que la largeur du support de A_i est inférieure à un pourcentage ρ de la taille maximum possible (Δ_l).

Si plusieurs règles vérifient simultanément cette propriété on sélectionne la plus spécifique c'est-à dire celle dont la somme des supports des données linguistique de sa partie condition est la plus faible.

La règle étant sélectionnée la prochaine étape consiste à la modifier. Le principe général de l'adaptation est la modification de la conclusion. On peut néanmoins également réaliser une modification de la partie condition en opérant un recentrage sur le vecteur entrée x_0. Soit $(\mu_1, \ldots, \mu_m)$ les centres des données linguistiques $(A_1, \ldots, A_m)$:

$$\acute{\mu}_i = \beta x_i + (1 - \beta)\mu_i$$

En pratique la valeur de β doit être gardée très faible de manière à éviter un phénomène d'oubli par délocalisation importante de règles existantes.

La partie condition de la règle ayant changée il est nécessaire de recalculer son nouveau degré d'activation $\acute{w}$ par rapport à l'entrée x_0.

Rappelons qu'avant adaptation on a la relation $\frac{a}{b} = y_0$. Adapter la partie conclusion signifie trouver un couple $(\acute{\mu}, \acute{\sigma})$ tel que :

$$\frac{a - w\mu\sigma + \acute{w}\acute{\mu}\acute{\sigma}}{b - w\sigma + \acute{w}\acute{\sigma}} = y_1$$

On essaye tout d'abord de garder un support de taille identique : $\acute{\sigma} = \sigma$. On calcule la valeur $\acute{\mu}$ correspondante grâce à l'équation 8.16. On vérifie ensuite si le couple obtenu est valide c'est-à dire si $[\acute{\mu} - \alpha\acute{\sigma}, \acute{\mu} + \alpha\acute{\sigma}] \subset [\mu_0, \mu_1]$. Si ce n'est pas le cas on détermine alors la valeur des deux termes μ et σ grâce à un processus identique à celui mis en place pour la création de règles.

La troisième et dernière opération que nous avons envisagée dans le cadre de notre algorithme d'apprentissage est la généralisation de la partie condition d'une règle existante.

8.7.3 Généralisation d'une règle

Nous avons vu au cours des deux précédentes sections comment ajouter une nouvelle règle ou modifier la partie conclusion d'une règle déjà existante. Nous allons nous intéresser au cours de cette section à la modification de la partie condition en élargissant le domaine d'activation d'une règle. De manière similaire à l'adaptation les règles susceptibles d'être généralisées doivent répondre à certains critères :

1. $\exists j \ / \ \forall i \in [0, m],\ i \neq j$ A_i est exponentielle A_j est exponentielle ou trapèze-exponentielle.

2. $\forall i \in [0, m],\ i \neq j,\ x_i \in [\mu_i - \alpha\sigma_i, \mu_i + \alpha\sigma_i]$

3. $x_j \in [\mu_j^1 - k\alpha\sigma_j^1, \mu_j^2 + k\alpha\sigma_j^2]$ où k est une constante (voir figure 8.11 pour la définition de $\mu_j^1, \mu_j^2, \sigma_j^1, \sigma_j^2$).

Ces trois conditions spécifient qu'une règle pour être sélectionnée doit être locale dans toutes les directions à l'exception d'une. Dans cette direction particulière la donnée linguistique peut être décrite à l'aide d'une fonction trapèze exponentielle. Le rôle de la généralisation est d'élargir la fonction dans cette direction de manière à prendre en compte le nouvel exemple sans changer la conclusion (voir figure 8.13).

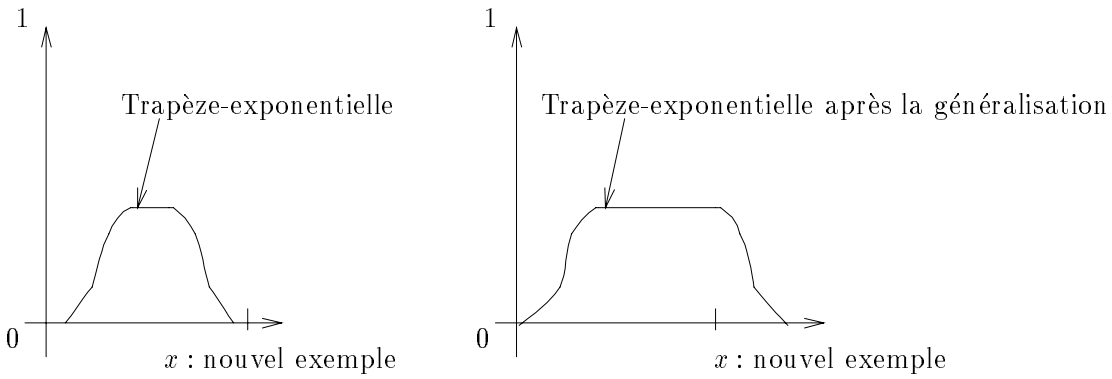


FIG. 8.13 - : *Extension de la fonction d'appartenance de la partie condition pour la prise en compte du nouvel exemple (mécanisme de généralisation)*

Si l'on compare à nouveau la partie condition des règles avec les catégories créées par l'algorithme *Fuzzy Artmap* la différence principale entre les deux approches est que nous n'autorisons l'augmentation de taille que selon l'axe principal de la catégorie (voir figure 8.14). Nous avons imposé cette restriction de manière à conserver un caractère local à la transformation en minimisant la variation de surface de la zone d'activation de la règle.

8.7.4 Algorithme

Les trois actions réalisables sur les règles floues ayant été décrites nous allons maintenant revenir sur l'algorithme proprement dit. A chaque exemple (x_0, y_1) présenté :

1. on calcule tout d'abord la valeur y_0 retournée par l'ensemble courant de règles au point x_0

2. si la valeur y_0 est non significative (voir section 5.3.5) cela veut dire qu'aucune règle n'est suffisamment active. La situation x_0 est inconnue pour le

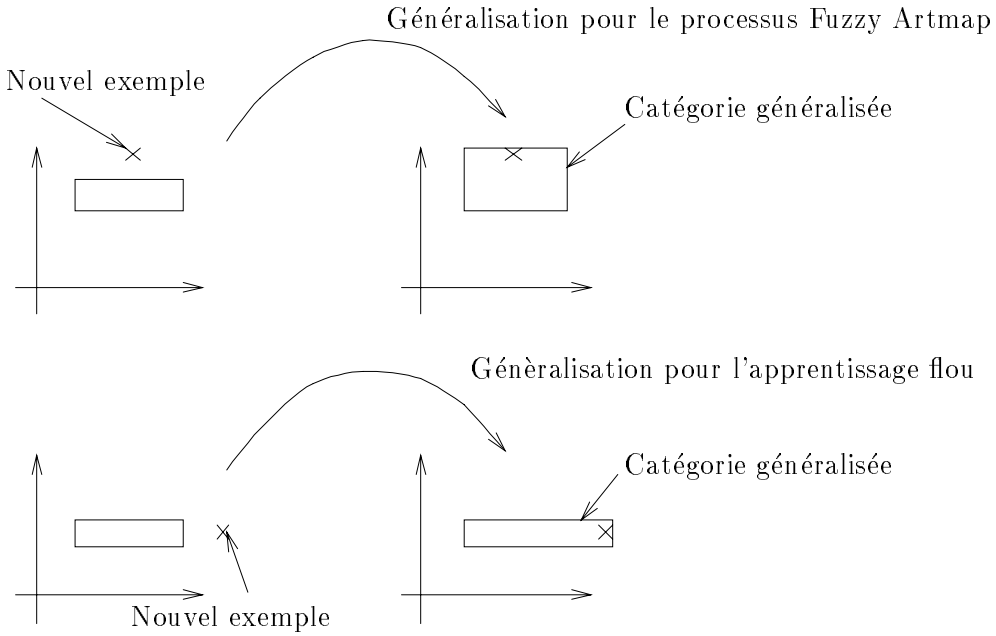


FIG. 8.14 - : *Comparaison entre le processus de généralisation de* Fuzzy Artmap *et notre processus de généralisation : les catégories ne peuvent augmenter que selon leur axe principal*

système. Une nouvelle règle est créée pour ce cas particulier associant y_0 à x_0Γ

3. la valeur y_0 est significative. Si $|y_1 - y_0| < \epsilon$ aucune modification n'est apportée. Dans le cas contraireΓon essaye tout d'abord de généraliser une règle existante. Si cela n'est pas possibleΓon cherche ensuite à réaliser une adaptation. En cas d'échecΓune nouvelle règle est finalement créée. La création est la dernière opération réalisée de manière à tenter de limiter l'explosion du nombre de règles.

L'apprentissage est réalisé par calcul direct de paramètres et ne nécessite donc pas de convergence par itérations successives. La complexité des opérations est de 1 pour la phase de création et de n pour l'adaptation et la généralisation (n étant le nombre courant de règles).

Le principe de l'algorithme ayant été exposéΓnous allons dans un premier temps nous intéresser à son application pour l'apprentissage de fonctions avant de le re-situer dans le cadre de l'apprentissage du critique heuristique adaptatif.

8.8 Application à l'apprentissage de fonctions

Nous allons présenter au cours de cette section quelques exemples d'apprentissage supervisé de fonctions réalisés à l'aide de notre algorithme. De manière à faciliter la visualisation des résultatsΓnous nous sommes intéressés à l'apprentissage de fonctions de $\Re^2 \mapsto \Re$.

8.8.1 Exemple de déformation de surface

Le premier exemple permet de visualiser le principe de l'apprentissage par déformation de surface. La figure 8.15 représente la fonction réalisée par un système flou ne possédant qu'une seule règle. Par présentation d'un exempleΓon impose au système $f(0.5, 0.5) = 0.6$. La valeur actuelle de f à cet emplacement est de 0.2. Il va y avoir apprentissage et donc déformation de la surface en ce point de manière à satisfaire la nouvelle contrainte. Le résultat est présenté figure 8.16. La figure de gauche correspond à une valeur γ faible (voir équation 8.13) et donc à une déformation locale. La figure de droite est obtenue en considérant une déformation plus globale par un paramètre γ important.

8.8.2 Incrémentalité de l'apprentissage

L'exemple suivant a pour but de montrer les propriétés d'incrémentalité de l'algorithme. On présente au système un ensemble de 100 points choisis de manière aléatoire sur le plan $z = \frac{x+y}{2}$. Chaque point n'est présenté qu'une seule fois. Le nombre total de règles créées est 14. Une seule règle a été adaptée. Aucune

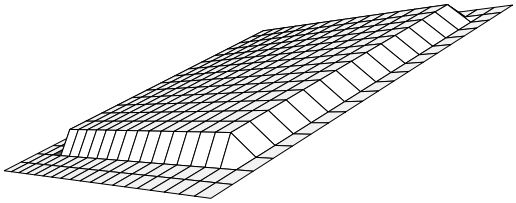


FIG. 8.15 - : *Fonction réalisée par un système flou composé d'une seule règle.*

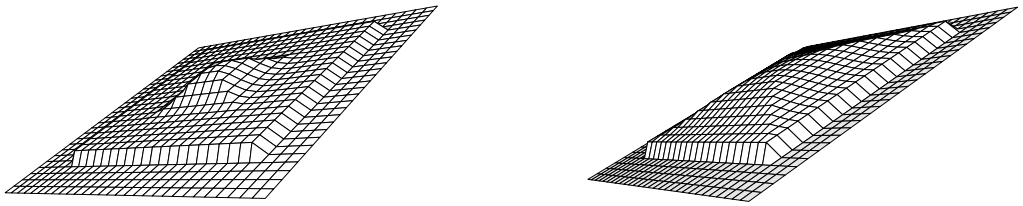


FIG. 8.16 - : *La figure de gauche montre le résultat de l'apprentissage après déformation locale de la surface. La figure de droite correspond à une déformation plus globale.*

généralisation n'a été effectuée. La fonction obtenue est représentée figure 8.17. La zone où f est nulle sur la gauche de la figure correspond à une zone où le système flou ne peut pas fournir de réponseΓaucune règle n'étant suffisamment active. Cela correspond à une région de l'espace où aucun exemple n'est présent et où les capacités de généralisation ne sont pas suffisantes pour pouvoir conclure. Cette première phase réaliséeΓon présente au système un nouvel exemple situé

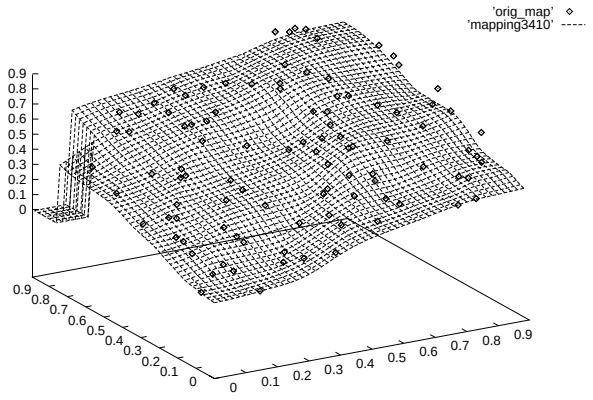


FIG. 8.17 - : *Apprentissage d'un plan à partir de* 100 *points répartis aléatoirement sur la surface.*

en $(0.5, 0.5)$ et dont la valeur associée est 0.3. Ce point est situé en dessous du plan appris. Une règle existante est adaptée de manière à le prendre en compte. La nouvelle fonction obtenue est représentée à gauche figure 8.18. On présente alors l'exemple original $(0.5, 0.5)$ avec comme valeur associée 0.5 (point du plan). Une nouvelle adaptation est réalisée (à droite figure 8.18). La fonction redevient identique à la fonction initialeΓindiquant bien une propriété d'incrémentalité.

Nous avons réalisé une expérience similaire en remplaçant le système flou ainsi que notre algorithme d'apprentissage par un réseau de neurones à propagation unilatérale entraîné par rétro-propagation du gradient. Le réseau est formé par trois couches (une couche d'entréeΓune couche cachée et une couche de sortie) comprenant respectivement 2Γ3 et 1 neurones.

Le premier apprentissage du plan est réalisé en présentant 45 fois la totalité des 100 exemples. La fonction obtenue est présentée figure 8.19. On présente au réseau le contre-exemple sans lui représenter les autres données. Ce point est appris en 43 itérations. La fonction descend de manière globale (l'intersection de la courbe avec l'axe vertical à gauche de la figure passe de 0.5 à 0.4). On représente à nouveau l'exemple correct. La courbe ne revient pas à sa position d'origine (l'intersection avec l'axe vertical reste en 0.4). Les deux surfaces correspondantes sont représentées figure 8.20. Contrairement à notre approche flouеΓla réalisation des deux opérations opposées a provoqué une déformation non réversible de la

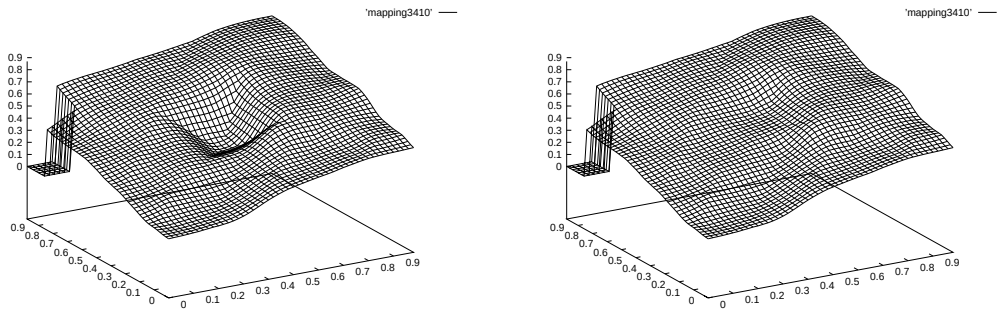


FIG. 8.18 - : *La figure de gauche représente la déformation de la fonction apprise f lors de la présentation d'un contre-exemple (point en dehors du plan). La présentation à nouveau d'un exemple correct ramène la fonction à sa forme initiale (figure de droite).*

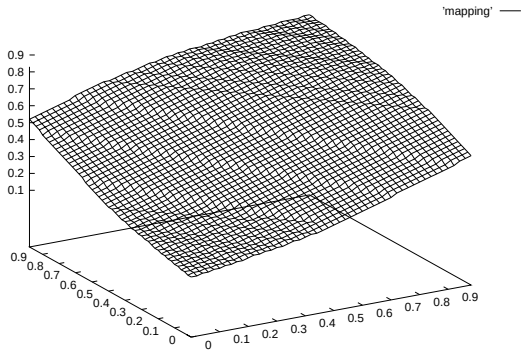


FIG. 8.19 - : *Fonction apprise par le réseau de neurone à propagation unilatérale*

surface.

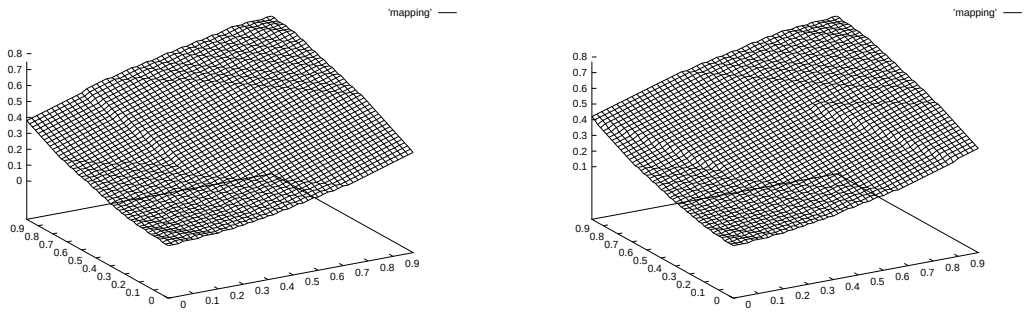


FIG. 8.20 - : *Présentation d'un contre-exemple (surface de gauche) puis de l'exemple correct (figure de droite) dans le cadre du réseau de neurone.*

8.8.3 Capacités de généralisation et de mémorisation

Nous allons maintenant nous intéresser aux capacités de généralisation et de mémorisation de notre système en les comparant en particulier à celles d'un réseau à couche.

Nous avons pour cela demandé au deux systèmes d'apprendre un échantillon de n^2 points prélevés sur une surface g. De manière à reproduire très grossièrement l'effet de l'acquisition de renforcements d'un robot réalisant plusieurs mouvements vers un même butΓces points sont prélevés régulièrement sur n rayons différents (avec n points par rayon). La figure 8.21 représente par exemple l'échantillonnage d'une gaussienne pour $n = 10$.

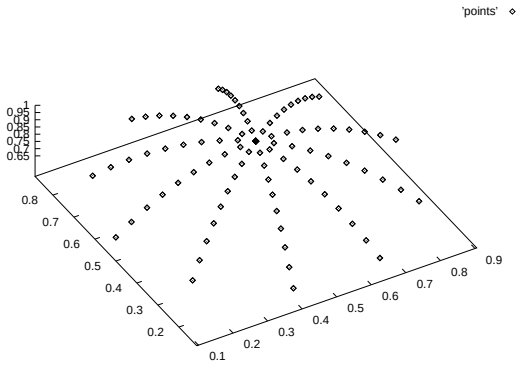


FIG. 8.21 - : *Echantillonnage d'une gaussienne en* 10 *rayons de* 10 *points chacun*

Les deux fonctions f_{flou} et $f_{neurone}$ obtenues sont évaluées en mesurant la moyenne et l'écart type de l'erreur absolue commise par rapport à g sur un

échantillon de 50 points répartis homogènement sur la totalité de l'espace d'entrée. Plus n est faible et plus les capacités de généralisation sont sollicitées. Plus n est grand et plus ce sont les capacités de mémorisation.

Nous avons considéré 4 fonctions g différentes. Les résultats sont regroupés figure 8.22. Chaque ligne comprend 3 figures et correspond à une fonction g particulière dont la liste est donnée ci-dessous. La figure de gauche représente la courbe du pourcentage de l'erreur moyenne commise par le réseau et les règles floues en fonction de n. La figure de droite représente la variation de l'écart type de ces deux erreurs.

Les fonctions considérées sont les suivantes :

- $g(x,y) = \frac{x+y}{2}$
- $g(x,y) = 1 - 2((x-0.5)^2 + (y-0.5)^2)$
- $g(x,y) = \frac{|sin(\pi x)|+|cos(\pi y)|}{2}$
- $g(x,y) = 1 - |x-y|$

L'analyse des courbes obtenues dans ces 4 cas particuliers nous permet de formuler un certain nombre de remarques :

- notre système présente de faibles capacités de généralisation et des capacités correctes de mémorisationΓ
- à un nombre d'exemples égalΓ l'approche neuronale fournie de meilleurs résultats que notre approcheΓ
- il existe une grande différence de temps de calcul entre les deux approches.

Nous allons maintenant revenir séparément sur chacun de ces trois points dans le cadre de notre approche.

La courbe de la moyenne de l'erreur absolue part de très haut (50 à 60% d'erreur) pour décroître et se stabiliser aux alentours de 10% lorsque le nombre d'exemples présentés augmente. L'erreur importante pour un nombre faible d'exemples (et donc la faible aptitude à la généralisation) peut s'expliquer de la manière suivante : les modifications engendrées par l'apprentissage ne sont que locales. Un nombre faible d'exemples va entraîner la création d'un nombre faible de règles localesΓcentrées sur chacun des points. Le système flou obtenu ne sera donc capable de proposer une valeur que sur une faible partie de l'espace d'entrée correspondant à sa zone de compétenceΓpénalisant ainsi la généralisation. Le choix de la mise en place d'un algorithme n'effectuant que des modifications locales a été justifié par la volonté de ne pas interférer avec des "connaissances" déjà stockées dans le système lors de la présentation d'un nouvel exemple. Nous avons choisi de privilégier l'incrémentalité face à la généralisation.

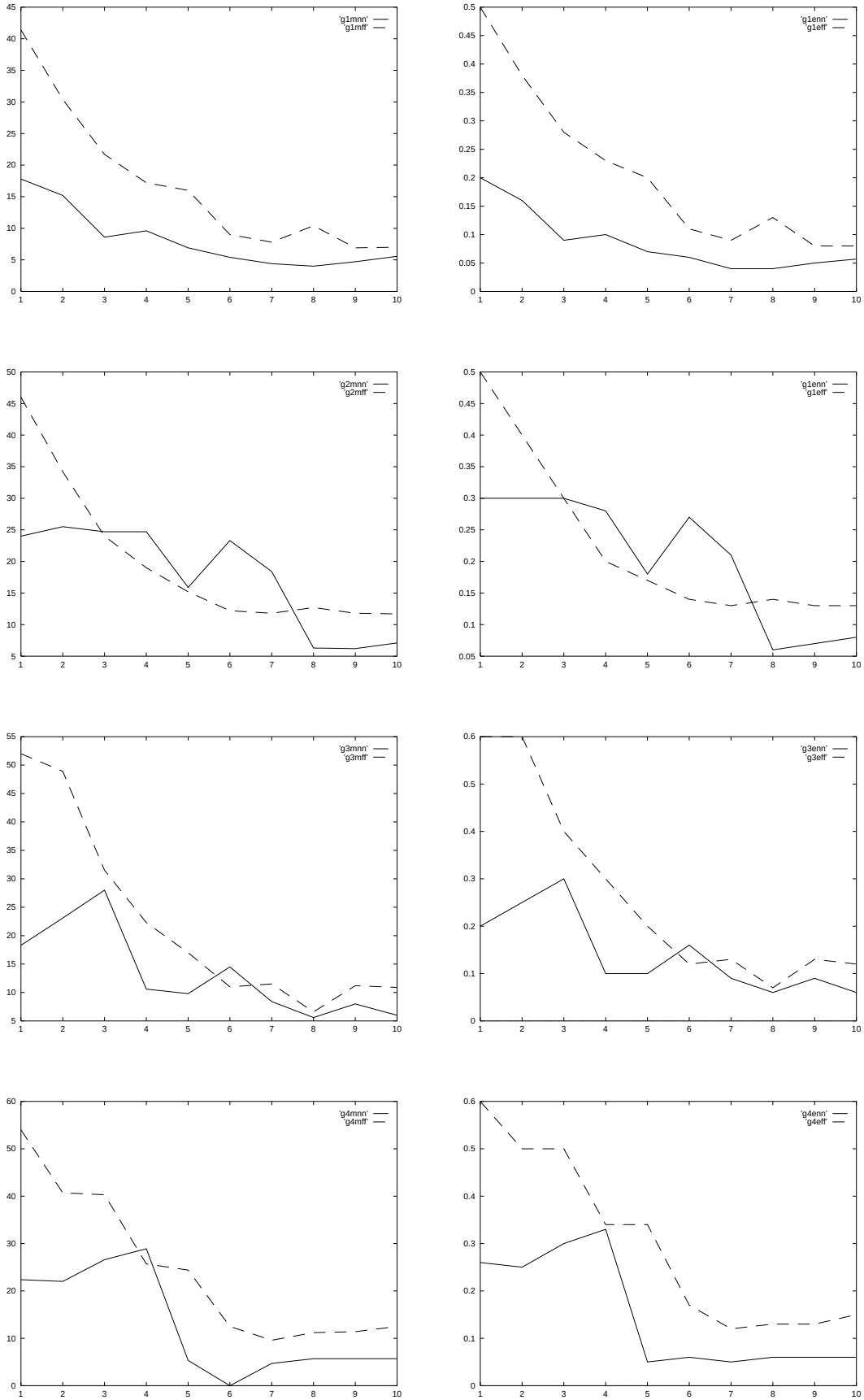


FIG. 8.22 - : *Comparaison pour quatre fonctions apprises de la moyenne et de l'écart type de l'erreur absolue commise par le réseau de neurones à couche (en trait continu) et par le système flou (en trait pointillé).*

Pour un nombre donné d'exemplesΓle système neuronal donne une erreur moyenne et un écart type inférieur à notre approche. Cet écartΓtrès important pour les valeurs faibles de nΓtend à se réduire au fur et à mesure que le nombre d'exemples augmente.

Comme nous l'avons déjà signalé à plusieurs repriseΓl'apprentissage d'un réseau à couche nécessite la présentation multiple de la totalité de la base d'exemples (entre 1000 et 5000 fois pour $n = 10$ dans le cas des fonctions ci-dessus). Le temps de convergence de l'algorithme atteint plusieurs secondes sur une station de type SPARC 10. Notre algorithme en revanche ne nécessite qu'un passage unique des exemples pour un temps de calcul de 0.004 seconde (pour 100 exemples).

Les principaux avantages de notre algorithme sont l'incrémentalitéΓle codage de connaissances initiales et la grande rapidité de calcul. Si la propriété principale recherchée est la généralisation à partir d'un nombre faible d'exemples ou la précision de la fonction appriseΓon préférera l'utilisation d'autres approches plus adaptées.

8.9 Observation du comportement du véhicule

Nous allons maintenant replacer notre algorithme d'apprentissage flou dans le cadre de la détermination d'un critique prédictif adapté à un comportement existant. Ce critique est déterminé grâce à l'algorithme $TD(\lambda)$ fourni section 8.5Γ basé sur l'observation conjointe de l'état du robot et du renforcement émis par l'environnement. Le comportement existant est fourni par le contrôleur flou décrit lors de la section 5.3.

Nous avons considéré l'expérience élémentaire suivante : l'environnement d'évolution du véhicule est complètement vide. On demande au robot de rejoindre un ensemble de buts disposés aléatoirement. Le système $TD(\lambda)$ observe en permanence la distance séparant le robot du butΓl'angle entre les deux ainsi que le renforcement reçu de l'environnement (0 dans le cas généralΓ1 si le but est atteintΓ voir section 8.5). Cet exemple est volontairement simple de manière à pouvoir prédire les résultats attendus et valider l'approche.

Nous savons que notre contrôleur flou a les capacités de rejoindre ce but. Le renforcement total amorti appris doit donc croître lorsque le robot se rapproche de son objectif. On peut s'attendre à apprendre une fonction dont la forme doit plus ou moins être une "cloche" centrée sur le point de distance et d'angle nul.

Les copies d'écran présentées figure 8.23 illustrent les différentes fonctions de renforcement prédictif apprises au cours de la première expérience. La base de règles initiale est vide.

Rappelons que les règles floues de gestion du but ont été initialement conçues de manière à orienter le robot vers son but en corrigeant de manière opposée tout écart à droite ou à gauche. Il est donc naturel de penser que la fonction de renforcement attendu aura son point maximum centré sur la direction nulle. Lorsque

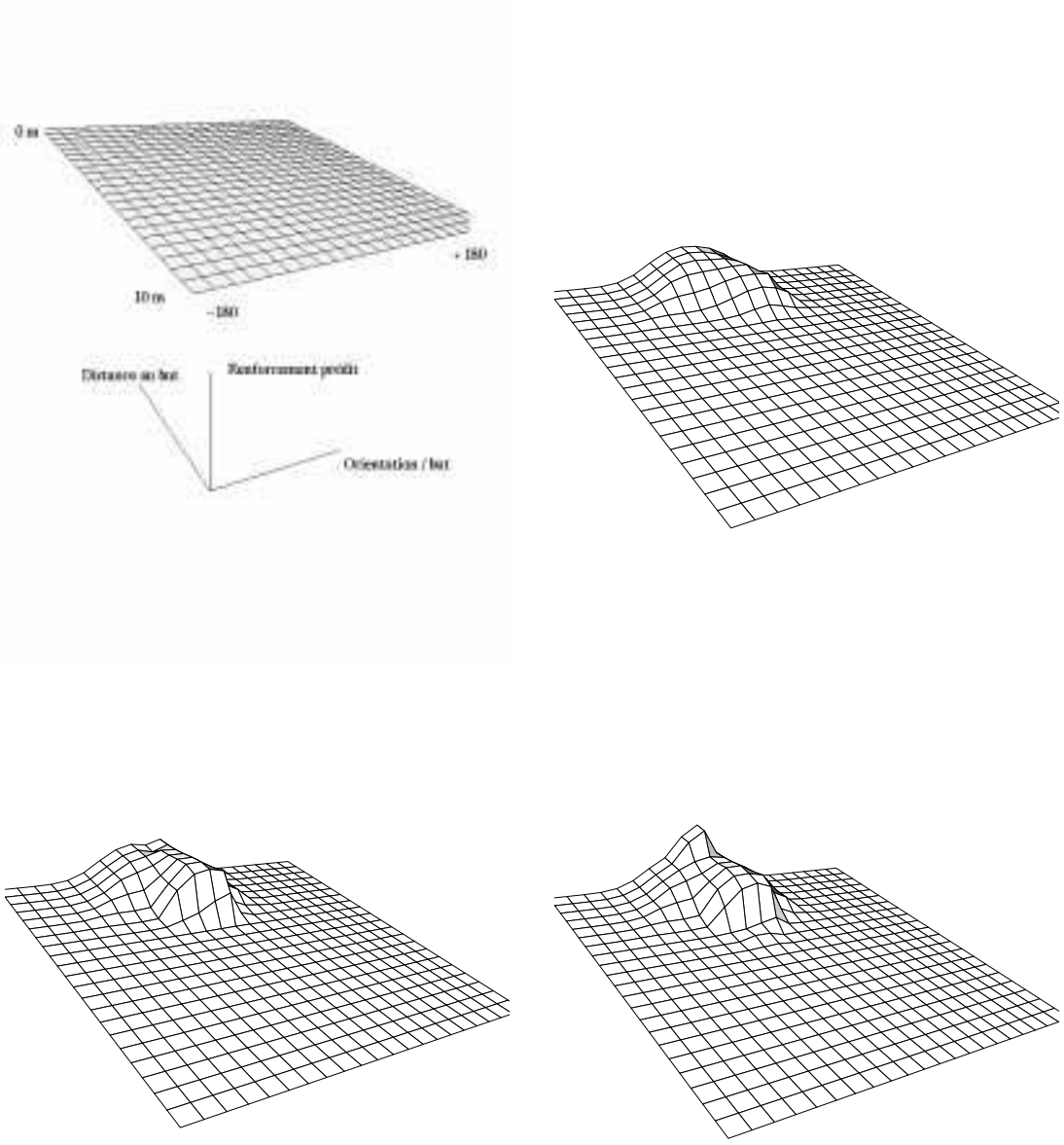


FIG. 8.23 - : *Evolution au cours du temps de la fonction de renforcement total amorti prédit. On n'utilise pas de connaissance initiale (la première fonction en haut à gauche est la fonction nulle)*

l'on reprend la courbe obtenue lors de l'expérience précédente, on constate que ce maximum est en fait centré sur environ -30^o (voir figure 8.24). La fonction

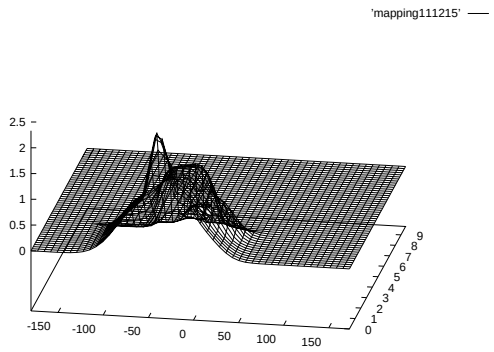


FIG. 8.24 - : *La fonction de renforcement prévu a son point maximum pour une direction vers le but de* -30^o *environ.*

apprise semble être en désaccord avec le comportement qu'elle est sensée représentée. Nous avons reconsidéré notre contrôleur de navigation et avons noté la variation de l'angle vers le but au cours de plusieurs déplacements. Quelques uns des résultats sont reportés figure 8.25. Ces courbes confirment ce qu'indique la

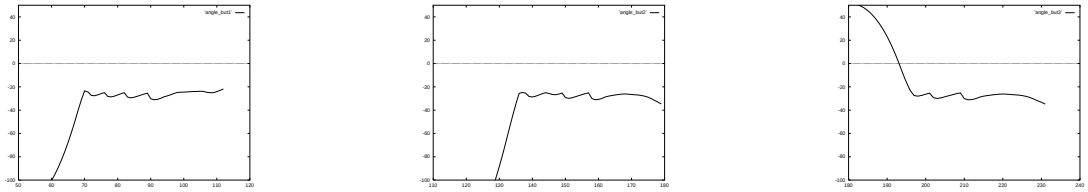


FIG. 8.25 - : *Trois exemples de variation de l'angle vers le but alors que le robot rejoint son objectif. La ligne en pointillés représente l'angle* 0*, direction du but*

fonction de renforcement attendu que nous avons appris. Nous avons reconsidéré les règles et trouvé une erreur dans la définition d'une donnée linguistique entraînant une dissymétrie de la commande pour la correction du cap. Cet incident permet d'illustrer que la fonction apprise est bien un reflet du comportement qu'elle observe.

Remarque : les expériences décrites précédemment on été réalisées en partant d'une base de règle vide. Dans le cas de notre exemple simple, le robot ne reçoit un renforcement non nul de la part de l'environnement que lorsqu'il atteint son but. Il ne peut donc apprendre que lorsque son renforcement prédictif augmente ou diminue d'un cycle à l'autre, provoquant ainsi une erreur de prédiction (voir équation 8.6). Fournir une connaissance initiale signifie créer un ensemble

de règles reproduisant une surface décroissante à partir de son point maximum correspondant au but ($d = \theta = 0$). Exemple (voir figure 8.26) :

```
si angle est gauche & distance est pres alors renforcement est moyen
si angle est droite & distance est pres alors renforcement est moyen
si angle est derriere_droite alors renforcement est faible
si angle est derriere_gauche alors renforcement est faible
si distance est loin alors renforcement est faible
si angle est devant & distance est pres alors renforcement est eleve
```

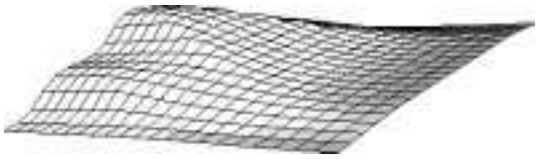


FIG. 8.26 - : *Exemple de renforcement prédit initial fourni par l'utilisateur.*

Notre algorithme permet l'intégration de connaissances initiales et réalise un apprentissage incrémentale. Il nécessite de plus une faible puissance de calcul et se montre très rapide. Il propose en revanche de moins bonnes propriétés de généralisation qu'une approche classique de type réseau à couche et génère un système flou dont la structure n'est pas optimale pour la fonction codée (grand nombre de règles). Ce second point n'est pas très important. Cet algorithme est destiné à être embarqué sur un robot où il doit apprendre et fournir des valeurs en permanence dans un délai très court. Le temps de réponse est primordiale et notre algorithme reste rapide même si le nombre de règles est élevé. Le cycle d'un robot est composé de phases d'activité et de phases de veille. Durant ces phases de veilleΓil est envisageable de réaliser une réorganisation de la base de règles courante de manière à réduire sa tailleΓcette réorganisation pouvant faire appel à des algorithmes hors-ligne tels que les algorithmes génétiques par exemple. Outre la réduction du temps de calculΓla diminution du nombre de règles peut rendre éventuellement envisageable leur interprétation à posterioriΓce qui est impossible actuellement.

8.10 Conclusion

L'apprentissage renforcé est un paradigme adapté aux problèmes de la robotique mobile. En ne séparant pas la phase apprentissage de la phase exploitationΓ il permet en particulier au système de s'adapter à de nouvelles situationsΓces situations pouvant provenir de l'environnement ou d'une modification de son fonctionnement interne (décalibrage de capteurs par exemple).

Comme nous l'avons rapporté au cours de ce chapitreΓun des problèmes principaux posé par cette approche est la lenteur de convergence. Cet aspect est traité

par de nombreux axes de recherche. Nous avons choisi celui proposant l'utilisation de connaissances initiales expertes du domaine et avons proposé un algorithme d'apprentissage rapide et incrémental permettant l'utilisation d'un système à base de règles floues dans un module de type Critique Heuristique Adaptatif. Ce module proposé permet de réaliser la première partie d'un système d'apprentissage renforcé. Nous ne nous sommes pas intéressés à la seconde partieΓindispensable afin d'obtenir un système complet : l'apprentissage de la loi de commande maximisant le critique adaptatif.

Chapitre 9

Conclusion

L'objectif de notre travail est d'accroître l'autonomie de déplacement d'un robot mobile. Cette autonomie a pour but de lui permettre de rejoindre un point quelconque de son environnement de travail tout en évitant les obstacles imprévus. Ce problème peut être décomposé en deux niveaux distincts :

1. le niveau cartographique chargé de prendre en compte la topologie de l'environnement et de sélectionner un ensemble de routes appropriéesΓ

2. le niveau géométrique dont le rôle est de gérer les contraintes imposées par le robot et la présence des différents obstacles afin de rejoindre les sous-buts spécifiés par la cartographie.

Nous nous sommes plus particulièrement intéressés à ce second aspect du problème.

Rejoindre un point P de l'espace d'évolution signifie que le robot est en mesure :

- de connaître avec suffisamment de précision sa position courante.

- de détecter la présence d'obstacles éventuels le séparant du but (le robot doit pouvoir se déplacer dans un environnement dynamique non spécialement préparé pour lui).

- de trouver un passage entre ces obstacles.

Le problème auquel on s'est intéressé est celui de déterminer en fonction des données capteurs quelles commandes doivent être envoyées à chaque instant au robot pour qu'il atteigne son but.

9.1 Travaux réalisés

Nous avons tout d'abord mis en place dans le cadre du projet de recherche Européen MITHRA Eureka : EU 110 une architecture à la fois de type fréquentielle et fonctionnelle. Cette architecture comprend entre autre les modules suivant :

- le contrôleur de véhicule. Son rôle est d'asservir les roues droite et gauche afin d'obtenir les mouvements désirés de translation et de rotation. Ce module a également pour tâche de maintenir la position estimée dans un repère absolu grâce aux valeurs retournées par les odomètres.

- le processus de modélisation. Son objectif est de maintenir à partir des données fournies par les capteurs ultrasons une description de l'environnement immédiat du robot en terme de segments de droite. Cette description a pour double but :

  1. de permettre au robot de corriger l'erreur commise par les odomètres sur sa position absolue.
  2. de permettre de détecter les obstacles.

- le module de perception. Son but est d'interpréter le modèle local construit par la modélisation afin d'extraire un chemin libre permettant au robot d'atteindre son objectif à partir de sa position courante.

- le module de navigation. Le module de navigation a pour tâche d'exécuter le chemin calculé.

Les différentes expériences que nous avons effectuées à l'aide de ce système nous ont permis de mettre en évidence les points suivants :

- le modèle de l'environnement en terme de segments de droite permet de relocaliser le robot de manière satisfaisante si le domaine d'évolution contient suffisamment de surfaces planes visibles (murs dégagés par exemple).

- les données contenues dans le modèle sont en revanche insuffisantes pour résoudre le problème de la navigation : beaucoup d'obstacles ne sont pas suffisamment réguliers pour être perçus sous forme d'ensembles de segments de droite.

- le délai nécessaire entre l'apparition d'un nouvel obstacle dans le champs de vue du robotΓson intégration dans le modèle et la génération d'un nouveau chemin permettant de l'éviter est trop important et représente un danger pour le véhicule.

– le plan généré repose sur le contenu du modèle de l'environnement. Ce modèle est en constante évolution et se modifie au fur et à mesure que le robot se déplace et peut être en mesure de mieux apercevoir les différents obstacles. Le plan est susceptible d'être fréquemment remis en cause.

De manière à gérer les imperfections du plan et à ne pas exiger son recalcul trop fréquentΓnous avons cherché à mettre en place un couplage plus direct entre la perception et l'action.

Depuis les premiers travaux de BrooksΓles systèmes comportementaux ont connu un grand succès en robotique de manipulation et en robotique mobile. La plupart des recherches dans ce domaine ont pour objectif la génération de comportements complexes à partir de comportements élémentaires. Nous avons préféré laisser l'élaboration de comportements complexes à des planificateursΓ adaptés à ce type de tâcheΓpour nous intéresser explicitement à la réalisation du comportement élémentaire nous intéressant : la navigation vers un but en évitant les obstacles.

Un comportement de navigation peut être défini comme une transformation f entre un espace de perception et un espace de commande. Un espace de perception peut être défini comme un espace dont chacun des axes correspond à un capteur. Chacun de ces capteurs peut être réel ou virtuelΓun capteur virtuel étant un capteur physique auquel on associe un traitement. De mêmeΓl'espace de commande est un espace ou chacun des axes correspond à une grandeur commandée.

Nous avons étudié les grandes approches de synthèse de transformations perception $\mapsto$ action existantes dans le cadre de la robotique mobile. On peut distinguer deux grandes catégories :

– celles basées sur un codage manuel par l'utilisateur.

– celles basées sur un apprentissage automatique.

Dans le cadre d'un codage manuel de la transformationΓla première approche que nous avons considérée est basée sur le principe des champs de forces virtuels. Le robot est attiré par son objectif et repoussé par les différents obstacles qu'il perçoit. Le principe de construction d'une transformation est basé sur la sélection d'une fonction particulière au sein d'une famille pré-définie de fonctions. Le problème classique de ce type d'approches est la présence dans le comportement obtenu de minima locaux et d'oscillations. Les minima locaux peuvent être provoqués par une configuration particulière d'obstacles. Leurs zones d'influence peuvent être très limitées. De manière à réduire les risques de blocageΓ nous avons proposé une approche favorisant le déplacement en essayant de générer en permanence des mouvements de translation (sous contrôle d'un module de réflexe)Γles mouvements de rotation étant déduits des champs de forces. Les minima locaux peuvent également être provoqués par la présence d'un passage étroit à franchir. Nous avons proposé la mise en place d'une fonction de répulsion

adaptiveΓréduisant temporairement son intensité lorsque de telles situations sont détectées.

Le comportement obtenu du robot est hésitant. Les problèmes des minima locaux ne sont pas complètement résolus. Ces problèmes ne sont en fait pas propre à une fonction particulière mais sont commun à l'ensemble des fonctions de la famille. Modifier les paramètres (comme l'ajustement dynamique de la force de répulsion) ne fait que déplacer le problème sans le résoudre. Nous avons alors cherché à réaliser la transformation perception - action non plus à l'aide d'une famille fixe de fonctions mais à l'aide d'un approximateur universelΓcapable d'approcher n'importe quelle fonction continue. L'approximateur que nous avons choisi est la logique floue pour ses capacités à traduire des connaissances symboliques en une fonction numérique.

Nous avons proposé dans ce cadre une approche basée sur une réduction préalable de la dimension de l'espace d'entrée par un pré-traitement simple des données capteurs. Nous avons ensuite identifié 8 situations perceptives particulières auxquelles nous avons associé grâce aux règles floues les actions appropriées. Les commandes associées à une situation quelconque sont générées grâce au mécanisme d'inférence. Elles résultent des capacités d'interpolation d'un contrôleur flou. Ce type d'approche est classique. Les résultats que nous avons obtenus sont comparables aux approches à base de potentiels (présence de minima locaux et d'oscillations). Le formalisme flouΓde part ses propriétés d'approximateur universelΓa la possibilité de décrire la transformation recherchée mais l'expression sous formes de règles de notre compréhension du mécanisme de navigation ne permet pas de générer une solution acceptable.

Les deux systèmesΓdont nous venons très brièvement de rappeler les principesΓs'inscrivent dans le cadre d'un codage manuel de la transformation perception action. Nous allons maintenant nous intéresser aux approches basées sur l'apprentissage en contrôle.

Trois grandes catégories d'approches peuvent être distinguées dans ce domaine selon le type d'informations disponibles :

1. l'apprentissage supervisé.
2. l'apprentissage distant.
3. l'apprentissage renforcé.

L'apprentissage distant est plus particulièrement conçu pour les systèmes devant apprendre à suivre une trajectoire de référence. Il n'est donc pas adapté à notre problème. Nous nous sommes plus particulièrement intéressés à l'apprentissage supervisé et à l'apprentissage renforcé.

Le principe général de l'apprentissage supervisé est de permettre à un contrôleur d'apprendre un comportement à partir d'une base d'exemples représentatifs de la forme *(situation perçue, action correspondante)*. L'apprentissage est

réalisé généralement en trois étapes : un opérateur humain pilote le robot par télé-opération afin de lui faire exécuter la tâche pour laquelle on souhaite le programmer. Les couples d'exemples sont stockés puis présentés à l'algorithme d'apprentissage. Le système résultant remplace finalement l'opérateur humain pour le contrôle du véhicule. La dimension de l'espace d'entrée et la complexité de la tâche à résoudre sont généralement telles que les exemples recueillis ne sont pas forcement représentatifs de la tâche à réaliser. De manière à prendre en compte cet aspect du problème, l'algorithme d'apprentissage sélectionné devra donc :

- posséder de bonnes propriétés de généralisation afin de pouvoir faire face correctement à une situation inconnue.

- être incrémental. Les trois étapes précédentes doivent être répétées chaque fois que le robot est confronté à une situation devant laquelle il ne réagit pas correctement. Les espaces d'entrée étant généralement de dimensions élevées, il n'est pas possible de stocker et de traiter l'ensemble des données rencontrées depuis le début de "l'existance" du système.

- la complexité du problème que l'on cherche à résoudre est mal maîtrisée. Le formalisme choisi doit donc pouvoir être en mesure de représenter le plus grand nombre de transformations possible.

Nous nous sommes tournés vers les réseaux de neurones. Contrairement aux approches robotique classiques basées sur l'apprentissage supervisé, nous avons choisi de prendre directement en compte les trois points mentionnés ci-dessus dans le choix de l'architecture retenue. Après une étude des principaux réseaux existants, nous avons sélectionné le réseau *Grow and Learn* proposé par Alpaydin en raison de ses propriétés en accord avec nos besoins. Ces propriétés ont été vérifiées à l'aide d'expérimentations menées sur un robot simulé (et très récemment sur un robot réel, les résultats ne figurant pas dans ce manuscrit).

Nous avons dans un deuxième temps cherché à accélérer et à fiabiliser l'algorithme d'apprentissage en réalisant une (ou plusieurs) transformations des données capteurs brutes permettant de faire ressortir leurs caractéristiques et de réduire la dimension de l'espace d'entrée. Chacune de ces transformations réalisent un codage. Elles sont basées sur une analyse en composantes principales d'un ensemble de données représentatives d'une situation. Le principe de l'algorithme est le suivant :

- les données utilisées pour l'apprentissage sont lues séquentiellement.

- on détermine pour chaque donnée lue s'il existe une transformation parmi celles déjà créées permettant de la coder (test réalisé en évaluant la perte d'information lors de l'opération de codage).

- si aucune transformation existante n'est adaptée, on se trouve dans une nouvelle situation perceptive et une nouvelle transformation est créée.

Cet algorithme permet de créer automatiquement des processus de perception adaptés chacun à une situation particulière. On associe ensuite à chacun de ces processus un réseau de type *Grow and Learn* chargé d'implémenter le comportement correspondant. Ce travail n'en est qu'à son commencement. Les premiers résultats obtenus sont prometteurs mais de nombreux travaux restent à accomplir.

Nous nous sommes tournés en dernière partie de ce travail vers l'apprentissage renforcé. Le principe de cet apprentissage et de laisser le contrôleur trouver lui-même quelle action doit être associée à chaque situation de manière à augmenter ses performances. Les performances sont mesurées par une fonction de renforcement codant les spécifications de la tâche que l'on souhaite accomplir. Les phases d'apprentissage et d'exploitation ne sont plus séparées permettant en théorie au système d'apprendre en permanence et donc de s'adapter automatiquement à toute situation nouvelle.

Nous nous sommes plus particulièrement intéressés dans ce cadre à l'apprentissage renforcé associatif proposé par Sutton. Cette approche permet de déterminer la loi de contrôle tout en ajustant la fonction critique par un mécanisme d'apprentissage supervisé (les exemples n'étant plus fournis par un opérateur humain mais par le robot au cours de ses déplacements). Un des inconvénients principal de ce type d'approche est le temps d'apprentissage important. Ce problème peut être partiellement résolu en ne laissant pas le robot apprendre de zéro mais en lui fournissant une connaissance initiale grâce à la logique floue par exemple. Cette connaissance peut être aussi bien fournie au contrôleur qu'au critique.

Nous nous sommes plus particulièrement intéressés à l'ajustement du critique réalisé par apprentissage supervisé d'un ensemble de règles floue. De part la nature du problème traité l'algorithme d'apprentissage retenu doit :

- présenter des propriétés de robustesse face aux données incohérentes (l'apprentissage renforcé associatif fournit par son principe des données incohérentes avant convergence).
- être incrémental.
- présenter une architecture extensible : la complexité de l'ensemble des règles doit s'adapter à la complexité du problème.

Les principaux algorithmes d'apprentissage supervisé pour systèmes flous ne répondant pas à une ou plusieurs des contraintes énoncées nous avons proposé un nouvel algorithme basé sur une manipulation directe des règles :

- par généralisation de la partie condition.
- par adaptation de la conclusion.
- par création de nouvelles règles.

Les opérations de création et de généralisation sont inspirées des opérations réalisées dans les réseaux de type *ART*. L'algorithme obtenu doit maintenant être intégré au sein d'un système complet d'apprentissage renforcé.

9.2 Discussion et perspectives

Les deux grandes familles de codage de la transformation perception $\mapsto$ action auxquelles nous nous sommes intéressés (champs de forces et logique floue) ont une approche similaire du problème. Le principe général est de déterminer la prochaine direction de navigation en résolvant un ensemble de conflits.

9.2.1 Reformulation de l'approche

.

Des éléments que l'on pourrait appelé "experts" sont chargés d'analyser chacun une particularité de l'environnement et d'apporter une réponse pour la traiter. Dans chacune des méthodes il existe :

- un expert chargé de gérer le but. C'est la force attractive pour le champs de force. C'est un ensemble de règles dans le cas de la logique floue.
- un ensemble d'experts chargés de gérer les différents obstacles. Ce sont par exemple les forces répulsives associées à chaque mesure capteur.

Dans le cas du but à atteindre l'expert fournit la direction à suivre pour le rejoindre directement si aucun obstacle n'est présent. Dans le cas des obstacles chaque expert fournit la direction à suivre pour ne pas entrer en collision avec l'obstacle dont il a la charge. Par exemple :

- dans le cas des champs de force la composante répulsive associée à la mesure capteur n indique la direction à suivre afin de s'éloigner le plus efficacement possible de l'obstacle détecté.
- dans le cas des systèmes flous on retrouve généralement un ensemble de règles invitant le robot à tourner à gauche si un obstacle se présente sur la droite.

Il est important de remarquer que chaque direction fournie n'est que la formulation d'une **hypothèse**. Sur un exemple et de manière très imagée un expert chargé de gérer un capteur sur la gauche du robot et détectant un obstacle proche va proposer de tourner à droite. L'expert sait avec certitude (si les données capteurs sont correctes) que tourner à gauche est une mauvaise direction ne devant pas être retenue. Il propose donc comme alternative de tourner à droite sans savoir si cela est possible ou non : c'est une hypothèse. Cette direction peut être

valide ou invalide mais elle est de toute façon meilleure que la direction vers la gauche.

Chaque expert ayant indiqué sa réponse, il s'agit ensuite de synthétiser l'information afin de fournir une seule direction permettant de guider le véhicule. Comme nous l'avons indiqué auparavant, il n'est pas possible de choisir une des directions parmi celles fournies, aucune n'ayant la garantie d'être valide. La réponse retenue est alors une combinaison des différentes directions et n'a pas de garantie non plus d'être correcte.

- dans le cas des champs de forces, on réalise une somme des différents vecteurs.

- dans le cas de la logique floue, on réalise en quelque sorte une moyenne pondérée des différentes propositions, la pondération étant fournie par le degré d'activation des règles (voir le chapitre 5 pour plus de précisions).

La direction provenant d'une combinaison est un "compromis" entre plusieurs directions proposées. Elle est peut être le résultat de l'opposition de plusieurs directions contradictoires, pouvant amener alors des situations de minima locaux.

Le problème de détermination de direction peut être reformulé en ne cherchant plus à générer des **hypothèses** mais des **certitudes**. Les experts ne s'intéressent plus aux obstacles mais aux directions libres. Chaque proposition formulée correspond à un passage et représente ainsi une direction valide. La direction finale commandée est obtenue simplement en sélectionnant une parmi celles proposées. Cette sélection peut être basée sur la position du but par exemple. Cette formulation a été utilisée par Borenstein (histogramme de densité [26]) et récemment avec beaucoup de succès par Bauer (steer angles [18]).

9.2.2 Perspectives

Repérer dans les données capteurs des directions libres permet de fournir des directions valides. Mais la validité de ces directions est néanmoins fonction de la validité des mesures capteurs. Beaucoup de travaux restent à accomplir dans l'extraction à partir de données capteurs d'indices robustes répondant aux besoins de la tâche.

Il nous semble intéressant également de reprendre le système flou de pilotage selon cette optique du problème. Les contrôleurs flous que nous avons utilisés (Mamdani, Larsen ...) sont par nature basés sur la résolution par fusion de conflits entre plusieurs règles. Ils ne sont donc pas très adaptés. Il semble plus approprié de considérer des algorithmes de sélection d'une décision parmi n utilisant la logique floue pour la manipulation des données.

L'apprentissage automatique et plus particulièrement l'apprentissage renforcé est une voie très intéressante permettant de compléter les systèmes précédents.

Les objectifs proposés correspondent parfaitement aux besoins de la robotique mobile :

- programmation par spécification de la tâche (renforcement) sans avoir à fournir la solution
- apprentissage en continue permettant au robot de faire face à de nouvelles situations ou à des modifications de son fonctionnement.

Les approches actuelles se heurtent à de nombreux problèmes tels que le temps d'apprentissage la convergence des algorithmes ou la généralisation de ce qui a été appris. De nombreux travaux restent à accomplir.

Enfin il y a un point que nous pensons être important et que nous n'avons pas abordé car sortant des limites de ce travail. Un système de navigation réactif n'est qu'un élément d'un robot complet. Comme nous l'avons spécifié au cours du chapitre 3 son but dans notre approche est de seconder le système de planification. Le problème est de déterminer quel type de lien ou de coopération doit exister entre les deux modules.

Annexe A

Le système Molusc

A.1 Mots clés de Molusc Clips

(all-sonars) : renvoie une liste contenant la totalite des mesures capteurs.

(analyse-ligne format data) : renvoie une liste formée à partir des éléments de la liste de nombres flottants *data* de rang spécifié par la liste d'entiers *format*.

(close-fuzzy-view) : fermeture des deux fenêtres d'affichages associées respectivement à la totalité des données capteurs et aux données capteurs simplifiées pour le contrôle flou (voir les fonctions *(open-fuzzy-view)* et *(fuzzy-view)*.

(create-eigen-base filename nb_vec format) : création d'une base de *nb_vec* vecteurs propres à partir des données contenues dans le fichier *filename*. La liste *format* permet d'indiquer la position des données à garder dans chaque ligne du fichier. La fonction génère un fait *(eigen num nb_vec dim)* où *num* correspond au numéro en mémoire de la baseΓ*nb_vec* au nombre de vecteurs contenus dans la base et *dim* à la dimension de ces vecteurs. La valeur retournée est *FALSE* si le fichier de données n'existe pas. Si le fichier existeΓla fonction retourne la liste de l'ensemble des valeurs propres.

(create-gal size_in size_out) : création d'un réseau de type *GAL*. La dimension du vecteur d'entrée est *size_in*. La dimension du vecteur de sortie est *size_out*. La fonction insère en retour un fait clips de la forme *(gal num size_in size_out)* où *num* correspond au numero en memoire du réseau (permet son identification pour les diverses opérations).

(control-cycle) : execute un cycle du controleur de vehicule. cette commande est sans effet lorsque l'on est connecte au vrai robot.

(disable-goto-periodic) : suppression du processus périodique de gestion d'un *goto* flou (voir également la fonction *enable-goto-periodic*). Remarque : cette fonction est executée par défaut lors de l'appel de la fonction CLIPS *reset*.

(eigen-compress num dim vecteur) : réalise la compression de *vecteur* grâce à la base de *dim* vecteur propres numéro *num*. La fonction retourne la liste des coordonées du vecteurs compressé.

(eigen-error num vector) : indique l'erreur commise après compression et décompression de *vector*.

(eigen-uncompress num dim vecteur) : réalise la décompression de *vecteur* grâce à la base de vecteur propres numéro *num* de dimension *dim*. La fonction retourne la liste des coordonées du vecteurs compressé.

(eigen-split filename nb_vec format nb_ex erreur) : création d'une famille de base de *nb_vec* vecteurs propres à partir des données contenues dans le fichier *filename*. La liste *format* permet d'indiquer la position des données à garder dans chaque ligne du fichier. La fonction génère les fait *(eigen num nb_vec dim)* où *num* correspond au numéro en mémoire de chaque base, *nb_vec* au nombre de vecteurs contenus dans chaque base et *dim* à la dimension de ces vecteurs. Les bases sont crées à partir de *nb_ex* exemples. Une nouvelle base est ajoutée lorsqu'aucune base existante ne peut reconstruire le vecteur courant avec une erreur inférieure à *erreur*. La valeur retournée est *FALSE* si le fichier de données n'existe pas. Si le fichier existe, la fonction retourne la liste de l'ensemble des valeurs propres.

(enable-goto-periodic) : mise en place d'un processus periodique charge d'exécuter la fonction *goto* floue. Remarque : la mise en place de ce processus remplace un appel periodique de la fonction *fuzzy-goto* (voir également *disable-goto-periodic*). Ce processus cyclique est automatiquement supprime lors de l'appel de la fonction CLIPS *reset*. Lorsque le but est atteint, le fait *(goal-reached)* est généré.

(eval-fuzzy-rules) : évaluation de l'ensemble des règles floues du système courant (voir également *set-fuzzy-system*).

(freeze-exploration) : empêche le module d'exploration de proposer un prochain but lorsque le but courant sera atteint. Cette fonction a pour but de permettre la réalisation d'un traitement éventuel (voir également *restart-exploration*. Remarque : cette fonction est automatiquement executee lorsqu'un but est atteint.

(fuzzy-display) : autorise l'affichage graphique du degre d'activation des regles du systeme flou courant.

(fuzzy-explore) : execute un cycle du système flou d'exploration de l'environnement. L'execution de cette fonction a pour effet de bord la création de fait clips associés à l'état du système. La liste des faits est :

- (etat-explore crash) : déclenchement de l'arrêt réflexe.
- (etat-explore minimum-potentiel) : le robot ne génère plus de déplacement sans avoir atteint son but. item (etat-explore oscillation) : la direction du robot oscille. Sa vitesse linéaire est nulle.
- (etat-explore but-atteint) : le robot a atteint le but fixé par le module d'exploration.
- (etat-explore navigation-normale) : le robot poursuit son exploration.

Ces états servent de base au calcul du renforcement.

(fuzzy-function l v p) avec l et v deux multifields de tailles identiques. Cette fonction permet de calculer la valeur de la variable de position p du mapping flou au point spécifié par les valeurs des différentes variables d'entrée. l est une mutifield contenant la position des diverses variables dans le système. v contient les valeurs de ces variables. *fuzzy-function* retourne la valeur si l'opération s'est déroulée correctementΓ*FALSE* si la valeur n'est pas significative (aucune règle active en ce point).

(fuzzy-goto) : execute un cycle de la version floue de la fonction *goto*. Lorsque le but est atteintΓle fait *(goal-reached)* est automatiquement inséré. Remarque : Molusc autorise la présence simultanée en mémoire de plusieurs systèmes flous. Le numéro du système flou courant doit donc correspondre aux règles de navigation (voir la fonction *set-fuzzy-system* pour selectionner le système flou courant et la fonction *get-fuzzy-system* pour connaître le système courant).

(fuzzy-view) : affichage des données capteur simplifiée associées à la logique floue (voir la fonction *(open-fuzzy-view)* et *(close-fuzzy-view)*.

(gal num size vec_in) : applique le vecteur d'entrée *vec_in* au reseau *GAL* identifié par *num*. Le vecteur de sortie est retourné sous forme d'une multifield. La dimension du vecteur de sortie est *size*.

(get-fuzzy-input d) : retourne la valeur avant "fuzzyfication" prise par la variable d'entrée numéro d. Si la position de la variable d est incorrecteΓ *get-fuzzy-input* retourne *FALSE*. Remarque : si le système est en mode *(set-fuzzy-input-memory)*Γla valeur de cette variable est celle fournie par la fonction *(set-fuzzy-input)*. Si le système est en mode *(set-fuzzy-input-sensors)*Γ la valeur provient alors de la mesure effectuée par les différents capteurs.

(get-fuzzy-output d) : retourne la valeur après "defuzzyfication" prise par une variable de sortie aprés évaluation des règles courantes (voir également *eval-fuzzy-rules* ainsi que *set-fuzzy-system*). Si la position *d* de la variable est incorrecteΓ*get-fuzzy-output* retourne alors le symbole *FALSE*.

(get-fuzzy-system) : retourne le numéro du système flou courant.

(get-mouse-point) : permet la saisie d'un point grâce à la souris. A l'appel de cette fonctionΓun click souris dans une des fenêtres entrainera la création du fait clips *(mouse-point x y)* où *x* et *y* représente les coordonnées absolue du point cliqué.

(get-robot-position) : retourne une multifield contenant respectivement l'abscisseΓl'ordonnéeΓl'orientation du robot dans le repère absoluΓainsi que sa vitesse linéaire et sa vitesse angulaire.

(get-robot-state l) : retourne une multifield contenant l'état robot. *l* spécifie la position des variables dont on souhaite connaître la valeur.

(init-learn-statistics) : initialise à zero les compteurs associés au nombre de règles créesΓadaptées ou généralisées durant l'apprentissage.

(learn-fuzzy-rules in-pos in-val out-val last-out-val out-pos type) avec :

- input-pos : multifield contenant la position des variables d'entrée présentent en partie gauche des règles concernées par l'apprentissage.
- input-val : multifield contenant la valeur des variables indiquées dans *input-pos*.
- output-val : valeur devant être apprise par la variable de sortie concernée par l'apprentissage au point donné par *input-val*.
- last-out-val : valeur prise par la variable concerné au point courant avant l'apprentissage.
- output-pos : position de la variable de sortie.
- type : type d'apprentissage. L'apprentissage peut être réalisé par *créationΓadaptation* ou (et) *généralisation* de règles (voir section 8.6 pour plus de détails). On associe à chaque type d'opération un nombre :
  - Création : 1
  - Adaptation : 2
  - Généralisation : 4

  Lors de l'apprentissageΓl'opération pourra être réalisée si nécessaire si le *et* binaire entre type et le numéro associé est diff'erent de 0. Par exempleΓsi *type* prend pour valeur 5Γl'apprentissage pourra faire appel à la création et à la généralisation de règles.

La fonction retourne -1 an cas de problème ou le type d'opération effectuée si l'apprentissage s'est bien déroulé :

- Généralisation : 1
- Adaptation : 2
- Création : 3

(learn-gal num vec_in vec_out) : realise un apprentissage du réseau *GAL* identifié par *num* (voir fonction *create-gal*). *vec_in* et *vec_out* sont les vecteurs d'entrée et de sortie à présenter au réseau.

(learn-gal-file num "filename" vec_in vec_out) : réalise l'apprentissage du réseau *GAL* identifié par *num* (voir fonction *(create-gal)*) à l'aide du fichier *filename*. *vec_in* et *vec_out* sont respectivement deux multifields contenant la liste de position dans le fichier d'entrée utilisés pour définir le vecteur d'entrée et de sortie du réseau.

(learn-statistics) : retourne une multifield contenant respectivement le nombre de règles généraliséesΓadaptées et créées depuis le dernier appel à la fonction *init-statistics*.

(load-eigen-base "filename") : charge une base de vecteurs propres à partir du fichier *filename*. La fonction crée un fait *(eigen num nb_vec dim)* où *num* représente le numéro en mémoire de la baseΓ*nb_vec* le nombre de vecteurs dans la base et *dim* la dimension de ces vecteurs. La valeur retournée est *TRUE* si le chargement s'est bien deroule et *FALSE* si le fichier n'existe pas.

(load-fuzzy "filename") : chargement d'un fichier de règles floues. la fonction retourne *TRUE* si le fichier existe et a pu être chargé correctementΓ*FALSE* sinon.

(load-gal "filename" num) : charge en memoire un reseau GAL a partir du fichier *filename*. Le reseau doit être crée auparavant par la commande *create-gal*. *num* correspond à son identificateur.

(modelisation) : execute un cycle du processus de modélisation (lecture des capteurs ultrasons et maintient du modèle local). Un cycle correspond à la détection et prise en compte dans le modèle d'un segment. Si aucun segment n'est détectéΓle cycle s'arrête après lecture de la totalité des 24 capteurs.

(move dist) : ordre de déplacement linéaire du véhicule. Cet ordre n'est pas valide en simulation.

(no-fuzzy-display) : supprime l'affichage graphique du degré d'activation des règles du système flou courant.

(open-fuzzy-view) : ouverture des deux fenêtres d'affichages associées respectivement à la totalité des données capteurs et aux données capteurs simplifiées pour le contrôle flou (voir les fonctions *(close-fuzzy-view)* et *(fuzzy-view)*.

(reset-first-learn) : permet au systéme d'apprentissage flou de considérer qu'aucune règle n'a été apprise. Ceci permet en particulier de réinitialiser le mécanisme de prise en compte de la succession des points.

(reset-reflex) : desactivation des reflexes.

(restart-exploration) : permet la reprise de l'exploration. L'appel à cette fonction autorise la création d'un prochain but par le module d'exploration (voir également *freeze-exploration*).

(say string) : activation de la synthèse vocale. Le système lit la chaine *string*.

(save-eigen-base "filename" num) : sauve la base de vecteur propres *num* dans le fichier *filename*. La fonction retourne *TRUE* si tout se déroule correctement et *FALSE* si *num* ne correspond pas à une base chargée en mémoire.

(save-fuzzy "filename") : sauvegarde sur disque de l'environnement flou courant. Le nom des fichiers créés sont *filename.rule* pour le fichier de règles et *filename.ling* pour le fichier de données linguistiques. Attention : cette fonction n'est disponible que pour un système flou de niveau 2 en mode *calcul exact*. La fonction retourne *TRUE* si la sauvegarde s'est bien déroulée *FALSE* sinon.

(save-gal "filename" num) : sauve dans le fichier *filename* le réseau *GAL* identifié par *num*.

(set-fuzzy-goal x y) : indique au systeme flou les coordonnées absolue (x, y) du but à atteindre.

(set-fuzzy-input l v) avec l et v deux multifields de tailles identiques. Cette fonction permet de spécifier les valeurs des différentes variables d'entrée lors de la prochaine évaluation des règles (voir fonction *set-fuzzy-input-memory* et *eval-fuzzy-rules*). l est une mutifield contenant la position des diverses variables dans le système. v contient les valeurs de ces variables. *set-fuzzy-input* retourne *TRUE* si l'opération s'est déroulée correctement *FALSE* sinon (par exemple si les deux listes ne sont pas de tailles egales).

(set-fuzzy-input-memory) : le données d'entrée du système floue sont prises en mémoire de l'ordinateur et non à partir des capteurs (voir fonction *set-fuzzy-input*). Remarque : le positionnement du mode d'entrée n'est pas attaché à un système flou particulier et reste vrai en cas de changement du système courant.

(set-fuzzy-input-sensors) : le données d'entrée du système floue sont prises à partir des capteurs associés. Remarque: le positionnement du mode d'entrée n'est pas attaché à un système flou particulier et reste vrai en cas de changement du système courant.

(set-fuzzy-system d) : selection du système d comme système flou courant. Le système flou courant est celui exécuté lors de l'appel de la fonction *eval-fuzzy-rules*. *set-fuzzy-system* retourne *TRUE* si d est valideΓ*FALSE* sinon.

(set-reflex d) : activation des reflexes du robot. La distance minimale entre un obstacle et le véhicule est la distance d.

(set-renforcement-period d) : spécifie la période d'envoi du signal de renforcement attaché à la navigation normale. Cette période est multiple de la période d'appel du processus d'exploration. d spécifie ce rapport entre les deux périodes.

(set-robot-pos x y a) : change la position du robot simulé. $x\Gamma y$ et a correspondent respectivement à l'abscisseΓl'ordonnée et l'orientation du véhicule.

(speed-command v_lin v_ang) : commande en vitesse du robot. La vitesse linéaire et angulaire est fixée respectivement par *v_lin* et *v_ang*.

(stop) : arret du véhicule. Cet ordre n'est pas valide en simulation.

(teleopere nom) : positionne le robot en mode téléopération. Le véhicule est piloté en vitesse à l'aide des quatres flèches du clavier (chaque pression sur une touche augmente ou diminue la vitesse d'un incrément constant). Les valeurs retournées par les 24 capteurs ultrasons ainsi que la vitesse courante linéaire et angulaire du robot sont périodiquement stockés dans le fichier "nom". Le mode téléopération est désactivé par pression sur la touche q.

(turn angle) : ordre de rotation du véhicule. Cet ordre n'est pas valide en simulation.

Annexe B

Le système réactif flou

B.1 Définition des données linguistiques

```
o_pres est exponentielle avec 0 4 0.5 0.5
ob_pres est fadeout_line avec 0 4 1.5 1
rapide_droite est exponentielle avec -40 40 -15 5
normal_droite est exponentielle avec -40 40 -10 5
lent_droite est exponentielle avec -40 40 -5 5
lent_gauche est exponentielle avec -40 40 5 5
normal_gauche est exponentielle avec -40 40 10 5
rapide_gauche est exponentielle avec -40 40 15 5
arret_ang est exponentielle avec -40 40 0 5
derriere_droite est fadeout_line avec -180 180 -135 45
droite est exponentielle avec -180 180 -90 45
devant_droite est exponentielle avec -180 180 -45 45
devant est exponentielle avec -180 180 0 45
devant_gauche est exponentielle avec -180 180 45 45
gauche est exponentielle avec -180 180 90 45
derriere_gauche est fadein_line avec -180 180 135 45
lent_avant est exponentielle avec -0.3 0.3 0.03 0.03
normal_avant est exponentielle avec -0.3 0.3 0.07 0.03
rapide_avant est fadein_line avec -0.3 0.3 0.1 0.3
arret_lin est exponentielle avec -0.3 0.3 0 0.3
lent_arriere est exponentielle avec -0.3 0.3 -0.1 0.1
b_pres est fadeout_line avec 0 10 1 3.5
b_normal est exponentielle avec 0 10 4.5 3.5
b_loin est fadein_line avec 0 10 8 3.5
vrai est constante avec 0 4 1
faux est constante avec 0 4 0
```

B.2 Règles du système de navigation réactive

B.2.1 Entête

```
type : 2
def_ling_int these

variable : evite_loin
variable : evite_pres_gauche
variable : evite_pres_droite
variable : evite_pres_devant
variable : couloir
variable : coin_gauche
variable : coin_droite
variable : bloque

affichage 0 e_loin
affichage 1 p_gauche
affichage 2 p_droite
affichage 3 p_devant
affichage 4 couloir
affichage 5 coin_g
affichage 6 coin_d
affichage 7 bloque
affichage 8 r_gauche
affichage 9 n_gauche
affichage 10 l_gauche
affichage 11 s_ang
affichage 12 l_droite
affichage 13 n_droite
affichage 14 r_droite
affichage 15 l_avant
affichage 16 n_avant
affichage 17 r_avant

rule_combine : add
rule_effect : mult
fuzzyfy_op : echelon
```

B.2.2 Détection de la situation courante

```
/* detection de la situation dans laquelle on se trouve */
si non obstacle_gauche est o_pres & non obstacle_droite est o_pres &
        non obstacle_devant est o_pres alors evite_loin est vrai
si obstacle_gauche est o_pres & non obstacle_droite est o_pres &
        non obstacle_devant est o_pres alors evite_pres_gauche est vrai
si obstacle_droite est o_pres & non obstacle_gauche est o_pres &
        non obstacle_devant est o_pres alors evite_pres_droite est vrai
si obstacle_devant est o_pres & non obstacle_gauche est o_pres &
        non obstacle_droite est o_pres alors evite_pres_devant est vrai
si obstacle_gauche est o_pres & obstacle_droite est o_pres &
        non obstacle_devant est o_pres alors couloir est vrai
si obstacle_gauche est o_pres & obstacle_devant est o_pres &
        non obstacle_droite est o_pres alors coin_gauche est vrai
si obstacle_droite est o_pres & obstacle_devant est o_pres &
        non obstacle_gauche est o_pres alors coin_droite est vrai
si couloir est vrai & obstacle_devant est o_pres alors bloque est vrai

\subsection{Gestion du but sans interaction avec les obstacles}

\begin{verbatim}

/* on se dirige vers le but */
comportement 1.0
si but_angle est derriere_gauche alors vitesse_ang est rapide_gauche
si but_angle est gauche alors vitesse_ang est normal_gauche
si but_angle est devant_gauche alors vitesse_ang est lent_gauche
si but_angle est devant alors vitesse_ang est arret_ang
si but_angle est devant_droite alors vitesse_ang est lent_droite
si but_angle est droite alors vitesse_ang est normal_droite
si but_angle est derriere_droite alors vitesse_ang est rapide_droite
si but_distance est b_pres alors vitesse est lent_avant
si but_distance est b_normal alors vitesse est normal_avant
si but_distance est b_loin & but_angle est devant
        alors vitesse est rapide_avant
fin
```

B.2.3 Gestion du but en tenant comptes des obstacles

```
/* on se dirige vers le but */
comportement 1.0
si but_angle est derriere_gauche & non obstacle_gauche est ob_pres &
```

```
        non obstacle_droite est contre alors vitesse_ang est rapide_gauche
si but_angle est gauche & non obstacle_gauche est ob_pres &
        non obstacle_droite est contre alors vitesse_ang est normal_gauche
si but_angle est devant_gauche & non obstacle_gauche est ob_pres &
        non obstacle_droite est contre alors vitesse_ang est lent_gauche
si but_angle est devant & non obstacle_devant est ob_pres
        alors vitesse_ang est arret_ang
si but_angle est devant_droite & non obstacle_droite est ob_pres &
        non obstacle_gauche est contre alors vitesse_ang est lent_droite
si but_angle est droite & non obstacle_droite est ob_pres &
        non obstacle_gauche est contre alors vitesse_ang est normal_droite
si but_angle est derriere_droite & non obstacle_droite est ob_pres &
        non obstacle_gauche est contre alors vitesse_ang est rapide_droite
si but_distance est b_pres & non obstacle_devant est o_pres
        alors vitesse est lent_avant
si but_distance est b_normal & non obstacle_devant est o_pres
        alors vitesse est normal_avant
si but_distance est b_loin & but_angle est devant &
        non obstacle_devant est o_pres alors vitesse est rapide_avant
fin
```

B.2.4 Gestion des obstacles

```
/* comportement en cas de coin gauche     */
comportement 0.5
si coin_gauche est vrai alors vitesse est arret_lin
si coin_gauche est vrai alors vitesse_ang est lent_droite
fin

/* comportement en cas de coin droite     */
comportement 0.5
si coin_droite est vrai alors vitesse est arret_lin
si coin_droite est vrai alors vitesse_ang est lent_gauche
fin

/* comportement par defaut si le robot est */
/* bloque en marche avant                   */
comportement 0.5
si bloque est vrai alors vitesse est lent_arriere
fin
```

```
/* comportement dans le cas d'un couloir etroit */
comportement 0.5
si couloir est vrai alors vitesse_ang est arret_ang
fin

/* comportement en cas de presence d'un obstacle */
/* pres sur la gauche                            */
comportement 0.5
si evite_pres_gauche est vrai alors vitesse_ang est arret_ang
fin


/* comportement en cas de presence d'un obstacle */
/* pres sur la droite                            */
comportement 0.5
si evite_pres_droite est vrai alors vitesse_ang est arret_ang
fin

/* l'obstacle est situe pres devant le robot */
/* cette situation constitue une urgence     */

comportement 0.5
si evite_pres_devant est vrai alors vitesse est lent_arriere
si evite_pres_devant est vrai alors vitesse_ang est lent_droite
fin
```

Annexe C

LMS et incrémentalité

C.1 Apprentissage par descente de gradient

Le but de cet exemple est de montrer l'apprentissage de coefficients dans le cas d'un réseau de type RBF[1] à l'aide d'un loi de mise à jour de type descente de gradient (voir sous-section 7.2.1). A chaque étapeΓle calcul de la modification à apporter aux paramètres nécessite donc la connaissance de la totalité des points (méthode non incrémentale). La base d'apprentissage est constituée de 20 points échantillonés régulièrement sur la droite $y = x$ dans l'intervalle $[-10, 10]$. Le réseau est constitué de 10 champs receptifs disposés régulièrement le long de cet intervalle et de rayon r tous égaux. Dans un premier tempsΓce rayon est déterminer de telle sorte qu'un champ i soit inactif lorsque ses voisins $i-1$ et $i+1$ sont atteigne leur activité maximum. Cela signifie qu'en tout point de l'espace d'entréeΓau plus deux champs sont actifs simultanément (recouvrement faible des différents champs receptifs). Le résultat après apprentissage est fourni figure C.1

Dans le deuxième exempleΓle rayon r a été augmenté de telle sorte que l'ensemble des champs soit actif en tout point de l'espace d'entrée. Le résultat après apprentissage est visualisé figure C.2.

Le système a été capable dans les deux casΓconnaissant l'ensemble des exemplesΓ d'apprendre la fonction $y = x$ initiale.

C.2 Apprentissage par la loi *LMS*

Nous avons repris les deux expériences précédentes dans les mêmes condition en remplaçant la descente de gradient par sa version incrémentale : la loi *LMS*. Les points exemples sont présentés successivement au réseau. A chaque nouvel exempleΓon applique la loi *LMS* jusqùà stabilisation de l'erreur.

La figure C.3 résume les différentes fonctions apprises après présentation

1. Radial Basis Function

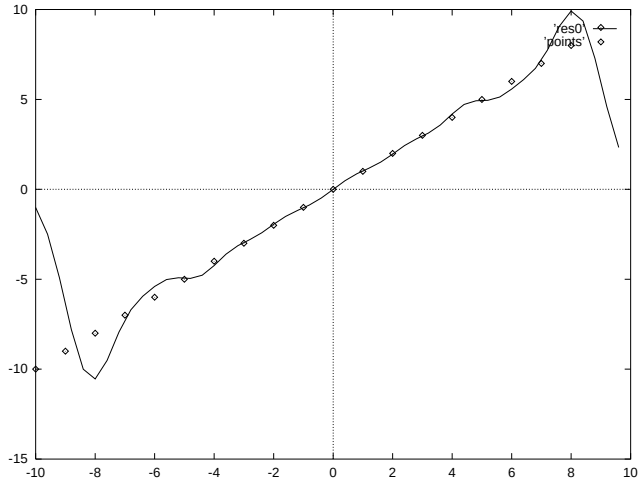


FIG. C.1 - : *Apprentissage par descente de gradient avec un faible recouvrement des champs receptifs. Les croix représentent les points exemples*

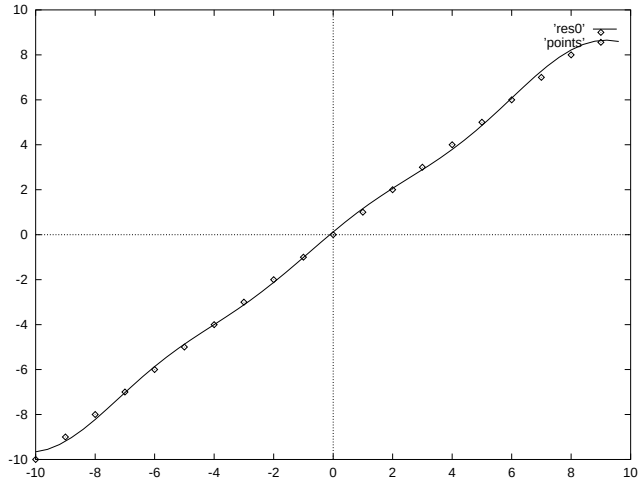


FIG. C.2 - : *Apprentissage par descente de gradient avec un fort recouvrement des champs receptifs*

d'un nouvel exemple. on peut constater que grâce à la faible interconnection des champsΓle système a éte en mesure d'adapter la courbe de manière à satisfaire le nouvel exemple sans modifier considérablement la partie déjà apprise. L'apprentissage est bien incrémental.

En revancheΓlorsque le recouvrement des champs receptifs est fortΓon constate que le système ne modifie plus la fonction de manière locale mais de manière globale afin qu'elle passe par le nouveau point (voir figure C.4). On ne peut plus parler d'apprentissage incrémental.

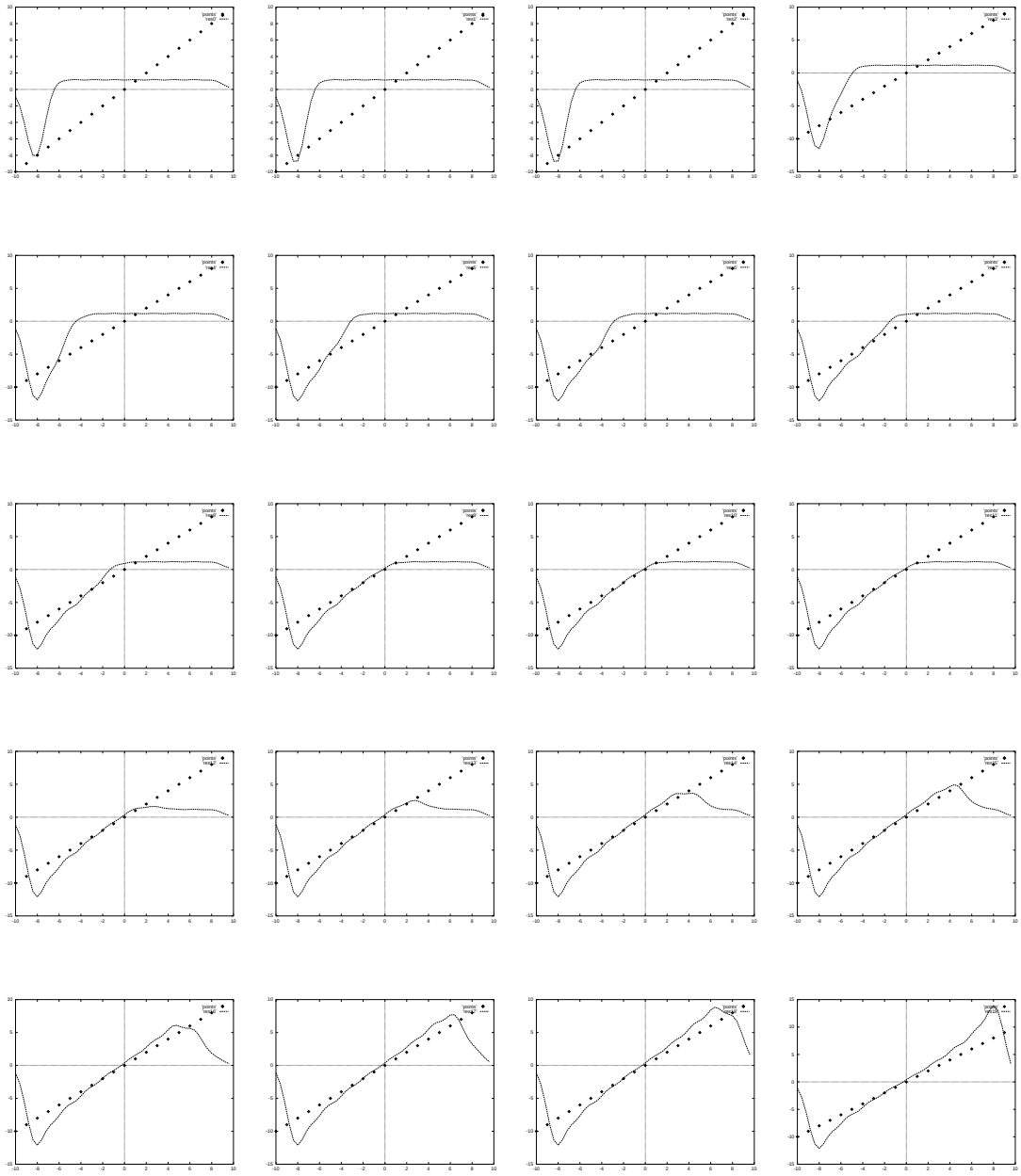


FIG. C.3 - : *Fonctions apprises par application de la loi LMS après chaque présentation d'un nouvel exemple. Le recouvrement des différents champs receptif est faible*

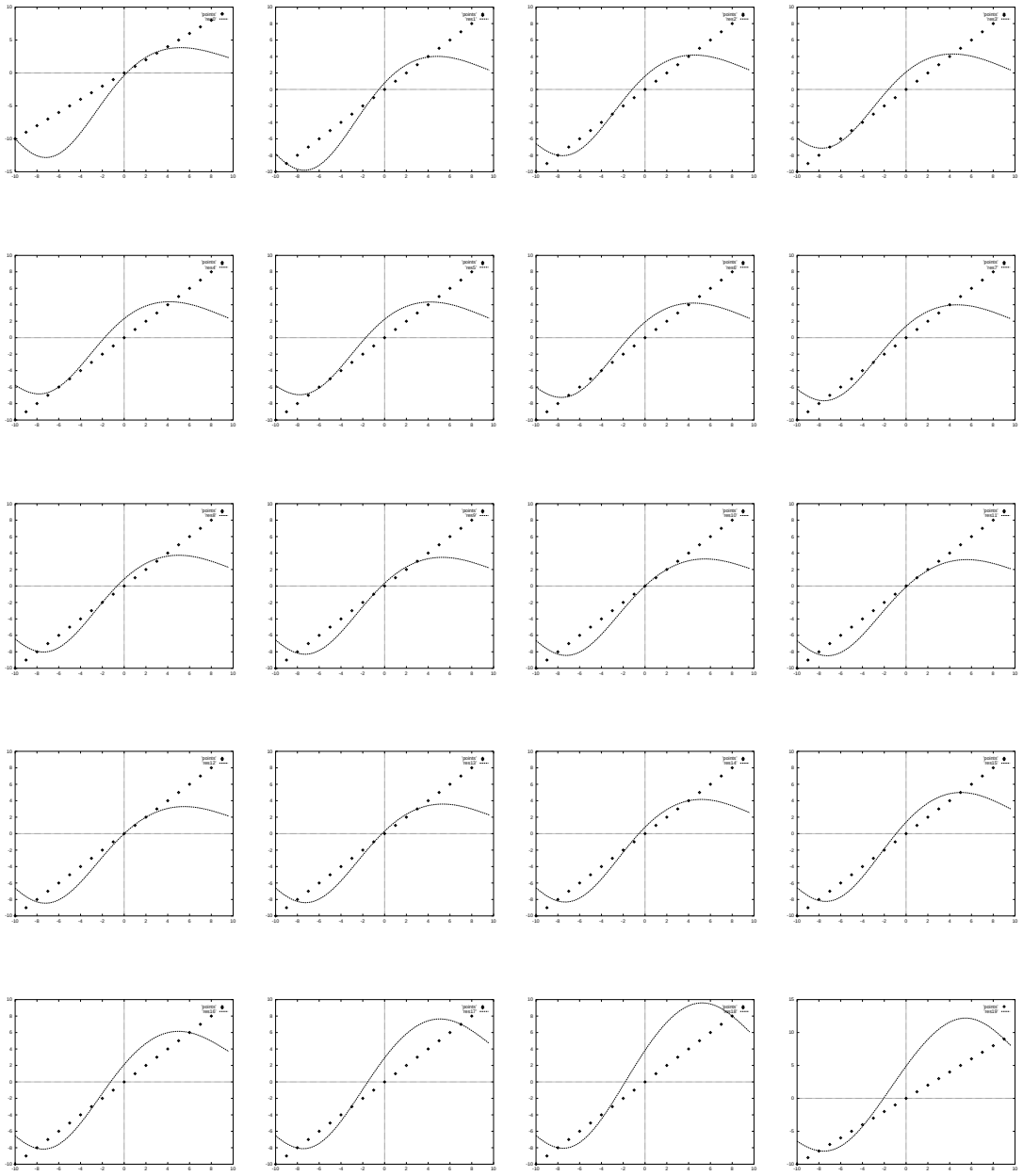


FIG. C.4 - : *Fonctions apprises par application de la loi LMS après chaque présentation d'un nouvel exemple. Le recouvrement des différents champs receptifs est fort*

Table des figures

2.1 Cycle classique issu de l'intelligence artificielle 19
2.2 L'architecture du système Mithra 21
2.3 Le robot mobile du LIFIA 23
2.4 Profil de vitesse pour un ordre de translation ou de rotation. . . . 23
2.5 Réception d'un nouvel ordre. Le profil est calculé de manière à ralentir le robot jusqu'à la nouvelle vitesse commandée puis de l'arrêter à la distance souhaitée. 24
2.6 Le contrôleur de véhicule 26
2.7 Répartition des capteurs ultrasons sur la ceinture du robot 31
2.8 Principe de mise à jour du modèle local de l'environnement libre . 31
2.9 Espace couvert par le robot lors d'une translation 33
2.10 Découpage de l'espace en 9 zones 33
2.11 La figure de gauche représente deux zones dans lesquelles il n'y a pas intersection. La figure centrale représente deux zones dans lesquelles il y a toujours intersection. La figure de droite représente un cas d'ambiguïté. Il suffit alors de calculer la position du point A. 34
2.12 La figure de gauche représente l'approximation de la surface balayée par le robot à deux portions de disques. L'algorithme teste donc l'intersection d'un segment avec un triangle et la partie supérieure du disque (figure de droite) 34
2.13 Principe de l'algorithme de recherche d'un chemin libre. *Findpath* est appelé récursivement sur les 4 segments créés 35
2.14 Construction des points d'évitement 36
2.15 Définition du sens d'un point par rapport à deux autres 36
2.16 Choix du point de passage 37
2.17 Définition du critère de traversée 38
2.18 Premier cas : le critère de connexité n'est pas vérifié pour l'extrémité du segment en cause 38
2.19 Deuxième cas : il faut contourner la barrière formée par les deux obstacles 39
2.20 Troisième cas : il faut s'éloigner du segment en cause 39
2.21 Le chemin créé peut comporter des sommets inutiles 40

2.22 Exemple d'exécution d'une succession d'ordres de translations et de rotations 41
2.23 Contenu du modèle local à trois instants différents lors du parcours d'un couloir par le robot. 42
2.24 Exemple de recherche d'un chemin à partir d'un modèle de l'environnement 42
2.25 Exemple d'exécution sur le robot d'un ordre de navigation. 43

3.1 Décomposition d'un système de contrôle en comportements 50
3.2 Description d'une machine d'état fini augmentée 50
3.3 Description de l'architecture AuRA 53
3.4 Architecture proposée par Payton 54
3.5 Exemples de modules experts 55
3.6 Architecture hybride dérivée de l'architecture proposée par Crowley 58
3.7 Définition d'un comportement f 60

4.1 Point singulier en forme de col. Le gradient de la fonction est nul en ce point 63
4.2 Définition du champ de forces virtuelles 72
4.3 Définition de la force $\vec{F}_{but}$ 74
4.4 Direction des vecteurs forces associés aux obstacles en fonction du capteur responsable de la mesure 75
4.5 Situation provoquant un minimum local. 77
4.6 Le robot est repoussé par les deux murs du couloir et ne peut atteindre son but. 78
4.7 Détection d'une zone étroite en analysant les valeurs respectives de F_{gx} et F_{dx} ainsi que F_{gy} et F_{dy}. 79
4.8 La rotation la plus directe imposée par les potentiels est impossible et provoque une situation de blocage 80
4.9 Définition des sous-buts intermédiaires pour le guidage de la rotation 81
4.10 Contournement d'un obstacle élémentaire 81
4.11 Traversée d'un passage étroit par le robot. La valeur du champ de répulsion est adaptée automatiquement (figure en haut à droite). La figure en bas à gauche représente l'évolution au cours du temps de la distance séparant le robot du but. 82
4.12 Traversée d'un passage étroit par le robot. La valeur du champ de répulsion est gardée constante (figure en haut à droite). La figure en bas à gauche représente l'évolution au cours du temps de la distance séparant le robot du but. 83
4.13 Plan du couloir : le robot doit naviguer entre le point A et le point B. 84
4.14 Trajet effectué par le robot pour aller du point A au point B . . . 84

4.15 Le robot remonte le couloir en marche arrière (figure du haut)Γ effectue un demi-tour dans l'angle du couloir (figure centrale) avant de rejoindre son but en marche avant (figure du bas). 85
4.16 Relevé de la vitesse linéaire (figure du haut) et angulaire (figure du bas) du robot lors du trajet aller-retour entre les deux extrémités du couloir. 86
4.17 Situation classique de blocage pour les approches de type champs de vecteurs. 87
4.18 Exemple d'exécution du module de planification couplé au système réactif. 88
4.19 Oscillations du robot lors de la rencontre du décrochement lorsque l'angle α est inférieur à une limite α_{cr} 89
4.20 Oscillation en présence d'un couloir étroit 90

5.1 Deux représentations différentes d'un même concept 95
5.2 La seule information sur le contenu est donnée par un degré d'appartenance pour la bouteille de gauche et par une probabilité pour la bouteille de droite 96
5.3 Fonctions d'appartenance associées aux différents nombres flous permettant de décrire la variable linguistique température. 98
5.4 Architecture générale d'un contrôleur flou 103
5.5 Exemple de deux codages de nombres réels en ensembles flous . . 103
5.6 Interprétation graphique du calcul du résultat d'un contrôleur flou de type Mamdani 108
5.7 Interprétation graphique du calcul du résultat d'un contrôleur flou de type Larsen 108
5.8 Principe général d'un contrôleur de type Tsukamoto 110
5.9 Données d'entrée du système flou relatives au but à atteindre : θ, d 112
5.10 Pré-traitement des données capteurs avant utilisation par le contrôleur flou. 114
5.11 Incertitude sur la distance à l'obstacle 116
5.12 Représentation de la possibilité de présence d'un obstacle à une distance d (la valeur retournée par les capteurs étant x_0) 116
5.13 L'approche Center of Area reflète plus la forme de la distribution de possibilité 117
5.14 L'utilisation de la somme normalisée pour la combinaison des règles permet de prendre en compte le nombre de règles arrivant à une conclusion similaire 119
5.15 Représentation des 8 situations perceptives différentes associées aux capteurs ultrasons. Sur les quatre données perceptivesΓseules celles correspondant à *près* sont représentées 122
5.16 Situation classique de blocage 123

5.17 Définition des fonctions d'appartenance des données linguistiques présentes dans la partie condition des règles 125
5.18 Le robot se trouve à la fois dans la situation *obstacle gauche* et *coin gauche* 126
5.19 Vitesse linéaire commandée en fonction de la distance au but et à un obstacle frontal. 128
5.20 Prise en compte de la distance des obstacles dans le comportement de gestion du but 129
5.21 Exemple de déplacement effectué par le robot piloté par le contrôleur flou. 131
5.22 Détection et prise en compte d'obstacles étroits tels que les pieds de chaises. 132
5.23 Mauvaise estimation de la situation: le robot pense être bloqué alors qu'il ne l'est pas. 132
5.24 Présence d'un minimum local. La figure de gauche représente les données perceptives utilisée. La figure de droite représente les degrés d'activation des règles. 133

6.1 Schéma général d'un système de contrôle 135
6.2 Systèmes adaptatifs à paramètres pré-programmés 138
6.3 Commande adaptative à modèle de référence 139
6.4 Commande adaptative avec identification du système à régler . . . 140
6.5 Architecture générale d'un système d'apprentissage en contrôle . . 141
6.6 Représentation des trois catégories d'apprentissage: l'apprentissage superviséΓdistant et renforcé 143
6.7 Utilisation d'un modèle de la transformation *(action, sortie)* ou *(action, évaluation)*. 146
6.8 Création du contrôleur par apprentissage direct du modèle inverse 148
6.9 Détermination du gradient de la transformation $action \mapsto sortie$ ou $action \mapsto évaluation$ par perturbation de l'action. 149

7.1 Architecture d'un réseau de type *Radial Basis Function.* 159
7.2 Architecture d'un réseau à propagation unilatérale comprenant 2 couches cachées. 161
7.3 Les fonctions proposées par le réseau de la figure de gauche et de la figure de droite passent toutes les deux par les points imposés. Mais seule la fonction de la figure de gauche généralise correctement.162
7.4 Structure d'un réseau CMAC. Chaque neurone de la couche d'entrée représente un point de l'espace X 165
7.5 Structure du réseau Fuzzy Art 170
7.6 Structure d'un réseau de type *GAL* 172

7.7 Création de neurones inutiles. La présentation des exemples $A\Gamma B\Gamma C$ conduit à la création de trois neurones exemples. La présentation de $B\Gamma C\Gamma A$ n'en crée que 2 173
7.8 Architecture d'un réseau expert dans le cadre de l'approche *IRRE* 179
7.9 Situation miroir d'une situation perçue 183
7.10 Le robot apprend à tourner à gauche 184
7.11 Le robot apprend à tourner à droite. Ses capacités à tourner à gauche ne sont pas altérées : l'apprentissage est incrémental 185
7.12 Comportement du robot après apprentissage du premier exemple . 185
7.13 Comportement du robot après apprentissage du deuxième exemple 186
7.14 Généralisation du comportement de franchissement d'une porte située à gauche. La reproduction d'écran située en haut à gauche représente l'environnement utilisé pour l'apprentissage 188
7.15 Situation perceptive et exemple stocké par le neurone gagnant lors du passage d'un embranchement 189
7.16 La figure de gauche et la figure de droite représentent deux situations différentes perçues de manière semblable 190
7.17 Similarité de situations perçues due à une reflexion multiple . . . 191
7.18 Distribution de points dans un espace à 2 dimensions et transformation dans un espace à 1 dimension. 192
7.19 Distribution des variables aléatoires *poids* et *taille* pour les classes *homme* et *femme* 193
7.20 Configuration du robot et du mur pour l'acquisition des données . 196
7.21 Moyenne (figure de gauche) et écart type (figure de droite) de l'erreur commise à la reconstruction en fonction de la dimension de la base de vecteurs propres. 198
7.22 Trajectoires réalisées par le robot en utilisant respectivement les données brutes puis la compression dans un espace de dimension 1Γ2 et 3. 199
7.23 Erreur commise sur l'ensemble des points ayant servi à la construction de la base. Les figures en haut de gauche à droite sont associées respectivement à des bases de dimension 1Γ2 et 3. La figure en dessous est une superposition des trois précédentes. 200
7.24 Moyenne (figure de gauche) et écart type (figure de droite) de l'erreur commise à la reconstruction en fonction de la dimension des bases de vecteurs propres. 201
7.25 Erreur commise sur l'ensemble des points ayant servi à la construction des 8 bases. 201
7.26 Nombre de bases créées en fonction de l'erreur maximale de reconstruction imposée par l'utilisateur. 202
7.27 Trajectoires obtenues par le robot en utilisant des bases de vecteurs de dimension 1Γ2 et 3 202

8.1 Mise à jour par apprentissage supervisé de l'estimation du temps pour le dimanche 208
8.2 Mise à jour par algorithme $TD(0)$ de l'estimation du temps pour le dimanche. 209
8.3 Lorsque la configuration du jeu atteint l'état étiqueté *mauvais*Γla partie se termine dans 90% des cas par votre perte et dans 10% des cas par votre victoire. Question : comment faut-il étiqueter le nouvel état en cas de victoire après son exploration? 210
8.4 Algorithme général pour l'apprentissage renforcé 213
8.5 Le robot ne peut contourner le mur que s'il accepte d'être momentanément puni avant d'être récompensé 214
8.6 Algorithme d'apprentissage renforcé avec utilisation d'un Critique Heuristique Adaptatif. 216
8.7 Utilisation du modèle d'action afin d'expliquer un épisode 220
8.8 Utilisation de la dérivée afin d'augmenter la vitesse de l'apprentissage. La figure de gauche représente la fonction que l'on cherche à apprendre. La figure centrale correspond à la fonction apprise en utilisant uniquement la valeur en trois points. La figure de droite correspond à la fonction apprise en utilisant la valeur ainsi que la pente en ces points. 221
8.9 Architecture neuronale de TESEO 222
8.10 Environnement d'expérimentation. Le robot doit rejoindre son but à partir du point B. Au bout de quelques essaisΓil apprend à ne plus pénétrer dans la zone A. 223
8.11 Définition des fonctions d'appartenance valides pour les données linguistiques. 233
8.12 Détermination de la zone solution des deux inéquations 237
8.13 Extension de la fonction d'appartenance de la partie condition pour la prise en compte du nouvel exemple (mécanisme de généralisation) 241
8.14 Comparaison entre le processus de généralisation de *Fuzzy Artmap* et notre processus de généralisation : les catégories ne peuvent augmenter que selon leur axe principal 242
8.15 Fonction réalisée par un système flou composé d'une seule règle. . 244
8.16 La figure de gauche montre le résultat de l'apprentissage après déformation locale de la surface. La figure de droite correspond à une déformation plus globale. 244
8.17 Apprentissage d'un plan à partir de 100 points répartis aléatoirement sur la surface. 245
8.18 La figure de gauche représente la déformation de la fonction apprise f lors de la présentation d'un contre-exemple (point en dehors du plan). La présentation à nouveau d'un exemple correct ramène la fonction à sa forme initiale (figure de droite). 246

8.19 Fonction apprise par le réseau de neurone à propagation unilatérale 246
8.20 Présentation d'un contre-exemple (surface de gauche) puis de l'exemple correct (figure de droite) dans le cadre du réseau de neurone. . . . 247
8.21 Echantillonnage d'une gaussienne en 10 rayons de 10 points chacun 247
8.22 Comparaison pour quatre fonctions apprises de la moyenne et de l'écart type de l'erreur absolue commise par le réseau de neurones à couche (en trait continu) et par le système flou (en trait pointillé). 249
8.23 Evolution au cours du temps de la fonction de renforcement total amorti prédit. On n'utilise pas de connaissance initiale (la première fonction en haut à gauche est la fonction nulle) 251
8.24 La fonction de renforcement prévu a son point maximum pour une direction vers le but de -30^o environ. 252
8.25 Trois exemples de variation de l'angle vers le but alors que le robot rejoint son objectif. La ligne en pointillés représente l'angle 0Γ direction du but 252
8.26 Exemple de renforcement prédit initial fourni par l'utilisateur. . . 253

C.1 Apprentissage par descente de gradient avec un faible recouvrement des champs receptifs. Les croix représentent les points exemples 280
C.2 Apprentissage par descente de gradient avec un fort recouvrement des champs receptifs 280
C.3 Fonctions apprises par application de la loi LMS après chaque présentation d'un nouvel exemple. Le recouvrement des différents champs receptif est faible 282
C.4 Fonctions apprises par application de la loi LMS après chaque présentation d'un nouvel exemple. Le recouvrement des différents champs receptifs est fort 283

Liste des tableaux

5.1 Critères intuitifs devant vérifier une implication floue dans le cadre du raisonnement approximatif 102
5.2 Fonctions d'appartenance de la conséquence de la règle en fonction de l'inférence choisie et de l'entrée. 106
5.3 Satisfaction des critères par les différents opérateurs d'inférence . 106

Bibliographie

[1] Adams (M.D.)ΓHu (H.) et Probert (P.J.). – Towards A Real-Time Architecture for Obstacle Avoidance and Path Planning in Mobile Robots. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γ pp. 584–589. – CincinnatiΓUSAΓ1990.

[2] Agre (P.E.) et Chapman (D.). – What are Plans for. *International Journal on Robotics and Autonomous Systems*Γvol. 6Γn° 1-2Γjuin 1990Γpp. 17–34.

[3] Albus (J.S.). – A new approach to manipulator control (cmac). *Journal of Dynamic Systems, Measurement and Control*Γvol. 97Γ1975Γpp. 270–277.

[4] Albus (J.S.). – A theory of cerebellar functions. *Mathematical Biology*Γ vol. 10Γ1975Γpp. 25–61.

[5] Alpaydin (E.). – *GAL : Networks that grow when they learn and shrink when they forget.* – Rapport technique n° TR-91-032Γ BerkeleyΓUSAΓInternational Computer Science InstitutΓ1991.

[6] Anderson (T.L.) et Donath (M.). – A Computational Structure for Enforcing Reactive Behavior in a Mobile Robot. *In : Mobile Robot III : Proceedings of SPIE.* – CambridgeΓMAΓUSAΓnovembre 1988.

[7] Anderson (T.L.) et Donath (M.). – Synthesis of Reflexive Behavior for a Mobile Robot Based Upon a Stimulus-Response Paradigm. *In : Mobile Robot III : Proceedings of the SPIE.* – CambridgeΓMAΓUSAΓnovembre 1988.

[8] Anderson (T.L.) et Donath (M.). – Animal Behavior as a Paradigm for Developing Robot Autonomy. *International Journal on Robotics and Autonomous Systems*Γvol. 6Γn° 1-2Γjuin 1990Γpp. 145–168. – ISSN 0921-8830.

[9] Anderson (T.L.) et Donath (M.). – Autonomous Robots and Emergent Behavior : A Set of Primitive Behaviors for Mobile Robot Control. *In : Proc. of the IEEE/RSJ International Workshop on Intelligent Robots and Systems*Γ pp. 723–730. – TsuchuriaΓIbarakiΓJapanΓjuillet 1990.

[10] Arciniegas (J.I.)ΓCios (K.J.) et Eltimsahi (A.H.). – Fuzzy InferenceΓRadial Basis FunctionsΓand Control of Flexible Robotic Manipulators. *In : Proc. of the International Conference on Artificial Neural Networks*Γéd. par Gielen (S.) et Kappen (B.). pp. 301–304. – AmsterdamΓThe NetherlandsΓ septembre 1993.

[11] Arkin (R.C.). – Motor Schema Based Navigation for a Mobile Robot : An Approach to Programming by Behavior. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γpp. 264–271. – RaleighΓUSAΓ avril 1987.

[12] Arkin (R.C.). – Dynamic Replanning for a Mobile Robot Based on Internal Sensing. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γpp. 1416–1421. – ScottsdaleΓArizonaΓUSAΓmai 1989.

[13] Arkin (R.C.). – Integrating BehavioralΓPerceptualΓand World Knowledge in Reactive Navigation. *International Journal on Robotics and Autonomous Systems*Γvol. 6Γn° 1-2Γjuin 1990Γpp. 105–122. – ISSN 0921-8830.

[14] Atkeson (C.A.). – Using locally weighted regression for robot learning. *In : Proc. of the IEEE International Conference on Robotics and Automation.* IEEEΓpp. 958–962. – SacrametoΓUSAΓavril 1991.

[15] Avnaim (F.)ΓBoissonnat (J-D.) et Faverjon (B.). – A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles. *In : International Conference on Robotics and Automation*Γpp. 1656–1661. – PhiladelphiaΓPA (USA)Γavril 1988.

[16] Baldi (P.) et Hornik (K.). – Neural networks and principal component analysis : Learning from examples without local minima. *Neural Networks*Γ vol. 2Γ1989Γpp. 53–58.

[17] Barraquand (J.) et Latombe (J.C.). – *Robot Motion Planning : a Distributed Representation Approach.* – Rapport technique n° STAN-CS-89-1257Γ Stanford UniversityΓCAΓUSAΓRobotics LabΓComputer Science DepartmentΓmai 1989.

[18] Bauer (R.)ΓFeiten (W.) et Lawitzky (G.). – Steer Angle Fields : An Approach to Robust Manœuvring in Cluttered Unknwon Environments. *In : Proc. of the International Workshop on Intelligent Robotic Systems*Γ éd. par Crowley (James L.) et Dubrawski (A.)Γpp. 67–71. – ZakopaneΓPolandΓ juillet 1993.

[19] Beer (R.D.)ΓChiel (H.J.) et Sterling (L.S.). – A Biological Perspective on Autonomous Agent Design. *International Journal on Robotics and Autonomous Systems*Γvol. 6Γn° 1-2Γjuin 1990Γpp. 169–186. – ISSN 0921-8830.

[20] Beom (H.R.) et Cho (H.S.). – A Sensor-based Obstacle Avoidance Controller for a Mobile Robot Using Fuzzy Logic an Neural Networks. *In : Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*Γ pp. 1470–1476. – RaleighΓUSAΓseptembre 1992.

[21] Berenji (H.R.). – Learning and Tuning Fuzzy Logic Controllers Through Reinforcements. *IEEE Transactions on Neural Networks*Γvol. 3Γn° 5Γseptembre 1992Γpp. 724–740.

[22] Berenji (H.R.)ΓLea (R.N.)ΓJani (Y.)ΓKhedkar (P.)ΓMalkani (A.) et Hoblit (J.). – Space Shuttle Attitude Control by Reinforcement Learning and Fuzzy Logic. *In : Proc. of the IEEE International Conference on Fuzzy Systems*Γpp. 1396–1402. – San FranciscoΓmars 1993.

[23] Berns (K.)ΓDillmann (R.) et Zachmann (U.). – Reinforcement Learning for the Control of an Autonomous Mobile Robot. *In : Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*Γpp. 1808–1815. – RaleighΓUSAΓjuillet 1992.

[24] Bezdek (J.C.). – Fuzzy Models - What Are They and Why? *IEEE Transaction on Fuzzy Systems*Γvol. 1Γn° 1Γfévrier 1993Γpp. 1–6. – ISSN 1063-6706.

[25] Borenstein (J.) et Koren (Y.). – High-speed obstacle avoidance for mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*Γvol. 19Γn° 5Γaoût 1988Γpp. 1179–1187.

[26] Borenstein (J.) et Koren (Y.). – Real-Time Obstacle Avoidance for Fast Mobile Robots in Cluttered Environments. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γpp. 572–577. – CincinnatiΓ 1990.

[27] Brooks (R.A.). – A Robust Layered Control System For A Mobile Robot. *IEEE International Transactions on Robotics and Automation*Γvol. RA-2Γ n° 1Γmars 1986Γpp. 14–23. – ISSN 0882-4967.

[28] Brooks (R.A.). – A Hardware Retargetable Distributed Layered Architecture for Mobile Robot Control. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γ pp. 106–110. – RaleighΓUSAΓ avril 1987.

[29] Brooks (R.A.)ΓConnell (J.) et Flynn (A.). – A Mobile Robot with Onboard Parallel Processor and Large Workspace Arm. *In : Proc. of the American Association for Artificial Intelligence Conference*Γpp. 1096–1100.

[30] Brooks (R.A.) et Flynn (A.M.). – Robots Beings. *In : Proc. of the IEEE/RSJ International Workshop on Intelligent Robots and Systems*Γpp. 2–8. – TsukubaΓJapanΓseptembre 1989.

[31] Canny (J.F.). – *The Complexity of Robot Motion Planning.* – MIT PressΓ 1988.

[32] Carpenter (G.A.) et Grossberg (S.). – The art adaptive pattern recognition by a self-organizing neural network. *IEEE Computer*Γmars 1988Γpp. 77–88.

[33] Carpenter (G.A.)ΓGrossberg (S.)ΓMarkuzon (N.)ΓReynolds (J.H.) et Rosen (D.B.). – Fuzzy ARTMAP : A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps. *IEEE Transactions on Neural Networks*Γvol. 3Γn° 5Γseptembre 1992Γpp. 698–712.

[34] Castro (J.L.)Γ Delgado (M.) et Herrera (F.). – A Learning Method of Fuzzy Reasoning by Genetic Algorithms. *In : Proc. of the First European Congress on Fuzzy and Intelligent Technologies (EUFIF'93)*Γpp. 804–809. – AachenΓGermanyΓseptembre 1993.

[35] Causse (A.) et Crowley (J.L.). – A Man Machine Interface for a Mobile Robot. *In : Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*Γpp. 327–335. – YokohamaΓJapanΓjul 1993.

[36] Causse (O.). – *Navigation sous contraintes : planification et contrôle d'exécution pour un robot mobile autonome.* – 46 Avenue Félix ViallетΓ38031 Grenoble Cedex 1ΓFRANCEΓThèse de PhDΓLIFIA-INPGΓ1994.

[37] Chatila (R.). – Mobile Robot Navigation : Space Modeling and Decisional Processes. *In : Proc. of the 3rd International Symposium of Robotic Research.* – GouvieuxΓFranceΓoctobre 1985.

[38] Chéhikian (A.). – Algorihtmes Optimaux pour la Génération de Pyramides d'Images Passe-bas et Laplaciennes. *Traitement du signal*Γvol. 9Γn° 4Γ1992Γ pp. 297–307.

[39] Chen (S.)ΓCowan (C.F.N.) et Grant (P.M.). – Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks. *IEEE Transactions on Neural Networks*Γvol. 2Γn° 2Γmars 1991Γpp. 302–309. – ISSN : 1045-9227.

[40] Chochon (H.). – Object-Oriented Design of Mobile Robot Control Systems. *In : Proc. of the Second International Symposium on Experimental Robotics.* – ToulouseΓFranceΓjuin 1991.

[41] Clark (R.J.)ΓArkin (R.C.) et Ram (A.). – Learning Momentum : On-line Performance Enhancement for Reactive Systems. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γpp. 111–116. – NiceΓ FranceΓmai 1992.

[42] Collin (I.)ΓMeizel (D.)ΓLefort (N.) et Govaert (G.). – Local Map Design and Task Function Planning for Mobile Robots. *In : Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*Γpp. 273–280. – MunchenΓGermanyΓseptembre 1994.

[43] Connolly (C.I.) et Grupen (R.A.). – On the Applications of Harmonic Functions to Robotics. *Journal of Robotic Systems*Γvol. 10Γn° 7Γoctobre 1993Γpp. 931–946.

[44] Connolly (C.I.) et Grupen (R.A.). – *Nonholonomic Path Planning Using Harmonic Functions.* – Rapport technique n° A-1085ΓUniversity of MassachusettsΓAmherstΓLaboratory for Perceptual RoboticsΓComputer Science DepartmentΓjanvier 1994.

[45] Cooper (M.G.) et Vidal (Jacques J.). – Genetic Design of Fuzzy Controllers : The Cart and Jointed-Pole Problem. *In : Proc. of the IEEE International Conference on Fuzzy Systems.* – OrlandoΓFloridaΓUSAΓjuin 1994.

[46] Cooper (M.G.) et Vidal (J.J.). – Genetic Design of Fuzzy Controllers. *In : Proc. of the Second International Conference on Fuzzy Theory and Technology.* – DurhamΓNorth CarolinaΓUSAΓoctobre 1993.

[47] Crowley (J.) et Christensen (H.). – *Vision as Process.* – Sringer VerlagΓ Basic Research SeriesΓ1993 (à paraître).

[48] Crowley (J. L.). – The state of art in mobile robotics. *In : The Fourth International Symposium on Robotics in Construction.* – HaifaΓIsraelΓjuin 1987.

[49] Crowley (J.L.). – Navigation for an Intelligent Mobile Robot. *IEEE Journal of Robotics and Automation*Γvol. RA-1Γn° 1Γmars 1985Γpp. 31–41.

[50] Crowley (J.L.). – Coordination of Action and Perception in a Surveillance Robot. *In : Proc. of the International Joint Conference on Artificial Intelligence*Γpp. 793–796. – MilanΓItalyΓaoût 1987.

[51] Crowley (J.L.). – Dynamic Modeling of Free-Space for a Mobile Robot. *In : Proc. of the IEEE/RSJ International Workshop on Intelligent Robots and Systems*Γpp. 626–633. – TsukubaΓJapanΓseptembre 1989.

[52] Crowley (J.L.). – World Modeling and Position Estimation for a Mobile Robot Using Ultrasonic Ranging. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γpp. 674–680. – ScottsdaleΓArizonaΓUSAΓMay 1989.

[53] Crowley (J.L.) et Reignier (P.). – Asynchronous Control of Rotation and Translation for a Robot Vehicle. *International Journal on Robotics and Autonomous Systems*Γvol. 10Γ1992Γpp. 243–251. – ISSN 0921-8890.

[54] Dean (T.L.) et Wellman (M.P.). – Learning. *In : Planning and Control*Γ chap. 9Γpp. 391–459. – San MateoΓCAΓUSAΓMorgan KaufmannΓ1991. ISBN 1-55860-209-7.

[55] Dubois (D.) et Prade (H.). – *Théorie des Possibilités : Application à la représentation des connaissances en informatique.* – MassonΓ1988Γ*Méthodes + Programmes.*

[56] Elfes (A.). – Using occupancy grids for mobile robot perception and navigation. *IEEE Computer*Γjuin 1989Γpp. 46–57. – ISSN 0018-9162.

[57] Fahlman (S.E.) et Lebiere (C.). – *The Cascade-Correlation Learning Architecture.* – Rapport technique n° CMU-CS-90-100Γ PittsburghΓPA 15213Γ USAΓSchool of Computer ScienceΓCarnegie Mellon UniversityΓfévrier 1990.

[58] Faibish (S.). – Neural Reflexive Control of a Mobile Robot. *In : Proc. of the IEEE International Symposium on Intelligent Control*Γpp. 144–146. – GlasgowΓScotlandΓaoût 1992.

[59] Ferguson (I.A.). – Toward an Architecture for AdaptiveΓRationalΓMobile Agents. *In : Proc. of Modeling Autonomous Agents in Multi-Agents World (MAAMAW)*Γpp. 1–12. – KaiserslauternΓGermanyΓaoût 1991.

[60] Fraichard (T.). – *Planification de Mouvements pour Mobile Non-Holonome en Espace de Travail Dynamique.* – Thèse de PhDΓ Laboratoire d'Informatique Fondamentale et d'Intelligence ArtificielleΓInstitut National Polytechnique de GrenobleΓavril 1992.

[61] Freisleben (B.) et Kunkelmann (T.). – Combining Fuzzy Logic and Neural Networks to Control an Autonomous Vehicle. *In : Proc. of the IEEE International Conference on Fuzzy Systems*Γpp. 321–326. – San FranciscoΓ CAΓUSAΓmars 1993.

[62] Fritzke (B.). – Let it grow - self-organizing feature maps with problem dependent cell structure. *In : in Proc. of the International Conference on Artificial Neural Networks.* – HelsinkiΓ1991.

[63] Fritzke (B.). – *Growing Cell Structures - A Self-Organizing Network for Unsupervised and Supervised Learning.* – Rapport technique n° TR-93-026Γ BerkeleyΓUSAΓInternational Computer Science InstituteΓ1993.

[64] Fukami (S.)Γ Mizumoto (M.) et Tanaka (K.). – Some Considerations of Fuzy Conditional Inference. *Fuzzy Sets Syst.*Γvol. 4Γ1980Γpp. 243–273.

[65] Fukunaga (K.). – *Introduction to Statistical Pattern Recognition.* – New-YorkΓUSAΓAcademic PressΓ1972.

[66] Garnier (Ph.). – Apprentissage d'une base de règles floues par la méthode du simplexe. *In : Les Applications des Ensembles Flous - Quatrièmes Journées Nationales.* – LilleΓDecember 1994. In french.

[67] Glorennec (P.Y.). – Un réseau Neuro-Flou évolutif. *In : Proc. of the Fourth Interantional Conference on Neural Networks and Applications*Γpp. 301–314. – NîmesΓnovembre 1991.

[68] Glorennec (P.Y.). – Fuzzy Q-Learning and Dynamical Fuzzy Q-LearningΓ1993.

[69] Glorennec (P.Y.). – Logique Neuro-Floue. *In : Actes des Troisèmes journées Nationales : Les Applications des ensembles flous*Γpp. 219–229. – NîmesΓFranceΓOctobre 1993. En Français.

[70] Gonzalez (A.)ΓPerez (P.) et Verdegay (J.L.). – Learning the Structure of a Fuzzy Rule : A Genetic Approach. *To appear in the International Journal of Fuzzy Systems and Artificial Intelligence, Reports and Letters*Γ1994.

[71] Goto (Y.) et Stentz (A.). – Mobile Robot Navigation : The CMU System. *IEEE Expert*Γvol. 2Γn° 4Γ1987Γpp. 44–54.

[72] Grossberg (S.)ΓCarpenter (G.A.) et Rosen (D.B.). – Fuzzy art : Fast stable learning and categorization o analog patterns by an adaptive resonance system. *Neural Networks*Γvol. 4Γ1991Γpp. 759–771.

[73] Guély (F.) et Siarry (P.). – Gradient Descent Method for Optimizing Various Fuzzy Rule Bases. *In : Proc. of the IEEE International Conference on Fuzzy Systems*Γpp. 1241–1246. – San FranciscoΓCAΓUSAΓmars 1993.

[74] Gullapalli (V.). – *Reinforcement Learning and its Application to Control.* – AmherstΓUSAΓThèse de PhDΓUniversity of MassachusettsΓfévrier 1992.

[75] Gullapalli (V.)ΓGrupen (R.A.) et Barto (A.G.). – Learning Reactive Admittance Control. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γpp. 1475–1480. – NiceΓFranceΓmai 1992.

[76] Hansen (S.J.) et Pratt (L.Y.). – Comparing biases for minimal network construction with back-propagation. *In : Advances in neural information processing systems*Γéd. par Touretzky (D.S.)Γpp. 177–185. – Morgan KaufmanΓ1989.

[77] Hansen (V.). – *Apprentissage des Comportements Réactifs pour la Commande des Robots.* – 46 Avenue Félix VialletΓ38031 Grenoble CedexΓDEAΓ LIFIA-INPGΓjuin 1994.

[78] Heikkonen (J.)ΓKoikkalainen (P.) et Oja (E.). – From Situations to Actions : Motion Behavior Learning by Self-Organization. *In : Proc. of the International Conference on Artificial Neural Networks*Γéd. par Gielen (S.) et Kappen (B.). pp. 262–267. – AmsterdamΓThe NetherlandsΓseptembre 1993.

[79] Herrera (F.)ΓLozano (M.) et Verdegay (J.L.). – *Generating Fuzzy Rules from Examples using Genetic Algorithms.* – Rapport technique n° DECSAI-93102Γ18701 GranadaΓSpainΓDepartment of Computer Science and Artificial IntelligenceΓUniversidad de GranadaΓoctobre 1993.

[80] Herrera (F.)ΓLozano (M.) et Verdegay (J.L.). – *Genetic Algorithms Applications to Fuzzy Logic Based Systems.* – Rapport technique n° DECSAI-93116Γ18701 GranadaΓSpainΓDepartment of Computer Science and Artificial IntelligenceΓUniversidad de GranadaΓoctobre 1993.

[81] Herrera (F.)ΓLozano (M.) et Verdegay (J.L.). – *Tuning Fuzzy Logic Controllers by Genetic Algorithms.* – Rapport technique n° DECSAI-93102Γ18701 GranadaΓ SpainΓ Department of Computer Science and Artificial IntelligenceΓUniversidad de GranadaΓjuin 1993.

[82] Irving (E.). – *Commande Adaptative.* – SUPELECΓGif-Sur-YvetteΓFranceΓ 1986.

[83] Ishibuchi (H.)ΓNozaki (K.) et Tanaka (H.). – Empirical Study on Learning in Fuzzy Systems. *In : Proc. of the IEEE International Conference on Fuzzy Systems*Γpp. 606–611. – San FranciscoΓCAΓUSAΓmars 1993.

[84] Ishikawa (S.). – A Method of Indoor Mobile Robot Navigation by Using Fuzzy Control. *In : Proc. of the IEEE/RSJ International Workshop on Intelligent Robots and Systems*Γpp. 1013–1018. – OsakaΓJapanΓnovembre 1991.

[85] Jacobs (P.)ΓLaumond (J-P.)ΓTaix (M.) et Murray (R.). – *Fast and exact trajectory planning for mobile robots and other systems with non-holonomic constraints.* – Research Report n° 90318ΓToulouse (F)ΓLaboratoire d'Automatique et d'Analyse des SystèmesΓseptembre 1990.

[86] Jang (J.R.). – Self-Learning Fuzzy Controllers Based on Temporal Back Propagation. *IEEE Transactions on Neural Networks*Γ vol. 3Γ n° 5Γseptembre 1992Γpp. 714–723.

[87] Jordan (M.I.)Γ Jacobs (R.A.) et Barto (A.G.). – *Task Decomposition Through Competition in a Modular Connectionist Architecture : The What an Were Vision Tasks.* – Rapport techniqueΓAmherstΓUSAΓDept of Computer and Information ScienceΓUniversity of MassachusettsΓ1990.

[88] Jordan (M.I.) et Rumelhart (D.E.). – Forward Models : Supervised Learning with a Distal Teacher. – Occasional Paper #40Γ1990.

[89] Jou (C.) et Wang (N.). – Training a Fuzzy Controller to Back Up an Autonomous Vehicle. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γpp. 923–928 Volume 1. – AtlantaΓUSAΓmai 1993.

[90] Jou (C.C.). – Supervised Learning in Fuzzy Systems : Algorithms and Computational Capabilities. *In : Proc. of the IEEE International Conference on Fuzzy Systems*Γpp. 1–6. – San FranciscoΓCAΓUSAΓmars 1993.

[91] Kaelbling (L.P.). – *Learning in Embedded Systems.* – MIT PressΓ1993. ISBN 0-262-11174-8.

[92] Kambhatla (N.) et Leen (T.). – Fast non-linear dimension reduction. *In : Advances in Neural Information Processing Systems 6.* – San FranciscoΓ CAΓUSAΓMorgan Kaufmannn PublishersΓ1994.

[93] Kandel (A.). – *Fuzzy Mathematical Techniques with Applications.* – Addison WesleyΓjuillet 1986. ISBN 0-201-11752-5.

[94] Katayama (R.)Γ Kajitani (Y.)Γ Kuwata (K.) et Nishida (Y.). – Self Generating Radial Basis Function as Neuro-Fuzzy Model and its Application to Nonlinear Prediction of Chaotic Time Series. *In : Proc. of the IEEE International Conference on Fuzzy Systems*Γpp. 407–414. – San FranciscoΓ CAΓUSAΓmars 1993.

[95] Kato (A.) et Kamikawa (K.). – Obstacle Avoidance Based on Approximate Reasoning for Mobile Robots. *In : Proc. of the IEEE/RSJ International Workshop on Intelligent Robots and Systems*Γ pp. 115–121. – TsukubaΓ JapanΓseptembre 1989.

[96] Khatib (O.). – *Commande dynamique dans l'espace opérationnel des robots manipulateurs en présence d'obstacles.* – ToulouseΓFranceΓThèse de PhDΓ Ecole Nationale Supérieur de l'Aéronautique et de l'Espace (ENSAE)Γ1980.

[97] Khatib (O.). – Real-Time Obstacles Avoidance for Manipulators and Mobile Robots. *The International Journal of Robotics Research*Γvol. 5Γn° 1Γ Spring 1986Γpp. 90–98.

[98] Koditschek (D.E.). – Exact Robot Navigation by Means of Potential Functions : Some Topological Considerations. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γpp. 1–6. – RaleighΓUSAΓ avril 1987.

[99] Koditschek (D.E.). – Autonomous Mobile Robots Controlled by Navigation Functions. *In : Proc. of the IEEE/RSJ International Workshop on Intelligent Robots and Systems*Γpp. 639–645. – TsukubaΓJapanΓseptembre 1989.

[100] Kohonen (T.). – Self-organized formation of topologically correct feature maps. *Biological Cybernetics*Γvol. 43Γ1982Γpp. 59–69.

[101] Kong (S.G.) et Kosko (B.). – Comparison of Fuzzy and Neural Truck Backer-Upper Control Systems. *In : Proc. of the International Joint Conference on Neural Networks*Γpp. 349–358 Volume 3. – Washington DCΓUSAΓ juin 1989.

[102] Koren (Y.) et Borenstein (J.). – Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γpp. 1398–1404. – SacramentoΓUSAΓavril 1991.

[103] Kosko (B.). – *Neural Networks and Fuzzy Systems : A Dynamical Systems Approach to Machine Intelligence.* – LondonΓPrentice-Hall International EditionsΓ1992. ISBN 0-13-612334-1.

[104] Kosko (B.). – Fuzzy Systems as Universal Approximators. *IEEE Transactions on Computers*Γ1993.

[105] Kreinovich (V.)ΓQuintana (C.) et Reiznik (L.). – Gaussian Membership Functions are Most Adequate in Representing Uncertainty in Measurements. *In : Proc. of the North American Fuzzy Logic Processing Society (NAFIPS)*Γpp. 618–624. – Puerto VallartaΓMEXICOΓdécembre 1992.

[106] Krogh (B.H.) et Thorpe (C.E.). – Integrated Path Planning and Dynamic Steering Control for Autonomous Vehicles. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γpp. 1664–1669. – ScottsdaleΓUSAΓmai 1986.

[107] Kröse (B.A.) et van Dam (J.W.M.). – Adaptive State Space Quantisation for Reinforcement Learning of Colision-free Navigation. *In : Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*Γpp. 1327–1332. – RaleighΓUSAΓjuillet 1992.

[108] La (R.)ΓGuély (F.) et Siarry (P.). – Apprentissage d'une base de règles floues par la méthode du recuit simulé. *In : Actes des Troisèmes Journées Nationales : Les Applications des Ensembles Flous*Γpp. 221–240. – NîmesΓ FranceΓOctobre 1993. En Français.

[109] Latombe (J.C.). – *Robot Motion Planning.* – Kluwer Academic PublishersΓ 1991. ISBN 0-7923-9129-2.

[110] Lee (C.C.). – Fuzzy Logic in Control Systems : Fuzzy Logic ControllerΓ Part I. *IEEE Transactions on Systems, Man, and Cybernetics*Γvol. 20Γn° 2Γmars 1990Γpp. 404–418. – ISSN 0018-9472.

[111] Lee (C.C.). – Fuzzy Logic in Control Systems : Fuzzy Logic ControllerΓ Part II. *IEEE Transactions on Systems, Man, and Cybernetics*Γvol. 20Γn° 2Γmars 1990Γpp. 491–435. – ISSN 0018-9472.

[112] Lee (M.A.) et Takagi (H.). – Integrating Design Stages of Fuzzy Systems using Genetic Algorithms. *In : Proc. of the IEEE International Conference on Fuzzy Systems*Γpp. 612–617. – San FranciscoΓCAΓUSAΓmars 1993.

[113] Linsker (R.). – Self-organization in a perceptual network. *IEEE Computer*Γ mars 1988Γpp. 105–116.

[114] Lozano-Perez (T.) et Westley (M.A.). – An Algorithm for Planning Collision-Free Paths among Polyhedral Obstacles. *Communication ACM*Γ vol. 22Γn° 10Γoctobre 1979Γpp. 560–570.

[115] Maes (P.). – A Spreading Activation Network for Action Selection. *In : Proc. of the International Conference on Intelligent Autonomous Systems*Γ éd. par Kanade (T.)ΓGrœn (F.C.A.) et Hertzberger (L.O.)Γpp. 875–885. – AmsterdamΓThe NetherlandsΓdécembre 1989.

[116] Maes (P.). – The Dynamics of Action Selection. *In : Proc. of the International Joint Conference on Artificial Intelligence*Γpp. 991–997. – DetroitΓ USAΓ1989.

[117] Maes (P.). – Designing Autonomous Agents. *International Journal on Robotics and Autonomous Systems*Γvol. 6Γn° 1-2Γjuin 1990Γpp. 1–2. – ISSN 0921-8830.

[118] Maes (P.). – Situated Agents Can Have Goals. *International Journal on Robotics and Autonomous Systems*Γvol. 6Γn° 1-2Γjuin 1990Γpp. 49–70. – ISSN 0921-8830.

[119] Malcolm (C.)ΓSmithers (T.) et Hallam (J.). – An Emerging Paradigm in Robot Architecture. *In : Proc. of the International Conference on Intelligent*

*Autonomous Systems*Γéd. par Kanade (T.)ΓGrœn (F.C.A.) et Hertzberger (L.O.)Γpp. 545–564. – AmsterdamΓThe NetherlandsΓdécembre 1989.

[120] Millán (J.R.). – *Reinforcement Learning of Goal-Directed Obstacle-Avoiding Reaction Strategies in an Autonomous Mobile Robot.* – Rapport techniqueΓ TP 361. 21020 ISPRA (VA)ΠItalieΓCEC Joint Research CenterΓmars 1993.

[121] Millán (J.R.) et Torras (C.). – A Reinforcement Connectionist Approach to Robot Path Finding in Non-Maze-Like Environments. *Machine Learning*Γ vol. 8Γn° 3-4Γmai 1992Γpp. 363–395. – ISSN 0885-6125.

[122] Millán (J.R.) et Torras (C.). – Efficient Reinforcement Learning of Navigation Strategies in an Autonomous Robot. *In : Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems.* – MunichΓ GermanyΓseptembre 1994.

[123] Mitchell (T.M.) et Thrun (S.B.). – Explanation-Based Neural Network Learning for Robot Control. *In : Advances in Neural Information Processing Systems 5*Γéd. par Giles (C.L.)ΓHanson (S.J.) et Cowan (J.D.). – San MateoΓCAΓUSAΓMorgan KaufmannΓ1992.

[124] Moravec (H.P.) et Elfes (A.). – High Resolution Maps from Wide Angle Sonar. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γpp. 116–121. – St LouisΓMissouriΓUSAΓMarch 1985.

[125] Morris (A.J.)ΓPeel (C.)ΓSauders (A.C.G.) et Kiparissides (C.). – Neural network feature detection and process monitoring. *In : Proc. of the IEEE International Symposium on Intelligent Control*Γpp. 560–565. – GlasgowΓ ScotlandΓaoût 1992.

[126] Nauck (D.)ΓKlawonn (F.) et Kruse (R.). – Combining Neural Networks and Fuzzy Controllers. *In : Proc. of the International Conference on Fuzzy Logic and Artificial Intelligence.* – LinzΓAustriaΓjuin 1993.

[127] Nauck (D.) et Kruse (R.). – A Fuzzy Neural Network Learning Fuzzy Control Rules and Membership Functions by Fuzzy Error Backpropagation. *In : Proc. of the IEEE International Conference on Neural Networks*Γpp. 1022–1027. – San FranciscoΓUSAΓmars 1993.

[128] Nie (J.) et Linkens (D.A.). – Learning Control Using Fuzzified Self-Organizing Radial Basis Function Network. *IEEE Transaction on Fuzzy Systems*Γvol. 1Γn° 4Γnovembre 1993Γpp. 280–287. – ISSN 0162-8828.

[129] Oja (E.). – *Data Compression, Feature Extraction and Autoassociation in Feed-Forward Netorks*Γpp. 737–745. – North HollandΓElsevier Science Publishers B.V.Γ1991.

[130] Payton (D.W.). – An Architecture for Reflexive Autonomous Vehicle Control. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γpp. 1838–1845. – San FranciscoΓUSAΓavril 1986.

[131] Payton (D.W.). – Internalized Plans : A Representation for Action Resources. *International Journal on Robotics and Autonomous Systems*Γvol. 6Γ n° 1-2Γjuin 1990Γpp. 89–103. – ISSN 0921-8830.

[132] Pin (F.G.)ΓWatanabe (H.)ΓSymon (J.) et Pattay (R.S.). – Autonomous Navigation of a Mobile Robot Using Custom-Designed Qualitative Reasoning VLSI Chips and Boards. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γpp. 123–128. – NiceΓFranceΓmai 1992.

[133] Pin (F.G.)ΓWatanabe (H.)ΓSymon (J.) et Pattay (R.S.). – Using Custom-Designed VLSI Fuzzy Inferencing Chips for the Autonomous Navigation of a Mobile Robot. *In : Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*Γpp. 790–795. – RaleighΓUSAΓjuillet 1992.

[134] Pomerleau (D.A.). – *Neural Network Perception for Mobile Robot Guidance.* – PittsburghΓPA 15213ΓThèse de PhDΓSchool of Computer ScienceΓCarnegie Mellon UniversityΓfévrier 1992. CMU-CS-92-115.

[135] Pomerleau (D.A.). – Input Reconstruction Reliability Estimation. *In : Proc. of the International Workshop on Intelligent Robotic Systems*Γéd. par Crowley (James L.) et Dubrawski (Artur)Γpp. 101–106. – ZakopaneΓPolandΓ juillet 1993.

[136] Pomerleau (D.A.). – Knowledge-Based Training of Artificial Neural Networks for Autonomous Robot Driving. *In : Robot Learning*Γéd. par Connell (J.H.) et Mahadevan (S.)Γchap. 2Γpp. 19–42. – Kluwer Academic PublishersΓ1993. ISBN 1-7923-9365-1.

[137] Procyk (T.J.) et Mandani (E.H.). – A Linguistic Self-Organizing Process Controller. *Automatica*Γ vol. 15Γ n° 1Γ janvier 1979Γ pp. 15–30. – ISSN 0005-1098.

[138] Raschke (U.) et Borenstein (J.). – A Comparaison of Grid-type Map-building Techniques by Index of Performance. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γ pp. 1828–1832. – CincinnatiΓOhioΓUSAΓmai 1990.

[139] Reif (J.) et Sharir (M.). – Motion planning in the presence of moving obstacles. *In : Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*Γpp. 144–154. – PortlandΓOR (USA)Γoctobre 1985.

[140] Reignier (P.). – Fuzzy Logic Techniques for Mobile Robot Obstacles Avoidance. *In : Proc. of the International Workshop on Intelligent Robotic Systems*Γéd. par Dubrawski (Artur) et Crowley (James L.)Γpp. 187–198. – ZakopaneΓPolandΓjuillet 1993.

[141] Reignier (P.). – Fuzzy Logic Techniques for Mobile Robot Obstacle Avoidance. *International Journal on Robotics and Autonomous Systems*Γvol. 12Γ 1994Γpp. 143–153. – ISSN 0921-8890.

[142] Reignier (P.). – Molusc : an Incremental Fuzzy Learning Approach. *In : Proc. of the International Workshop on Intelligent Robotic Systems*Γéd. par Crowley (J.L.) et Borkowski (A.)Γpp. 178–187. – GrenobleΓFranceΓ juillet 1994.

[143] Reignier (P.)ΓHansen (V.) et Crowley (J.L.). – Incremental Supervised Learning for Mobile Robot Reactive Control. *In : Proc. of the International Conference on Intelligent Autonomous Systems.* – KarlsruheΓGermanyΓ mars 1995.

[144] Rissamen (J.). – Stochastic complexity. *Journal of the Royal Statistical Society*Γ1987.

[145] Rosenblatt (F.). – *Principles of Neurodynamics : Perceptrons and the Theory of Brain Mechanism.* – Washington D.C.ΓUSAΓSpartan BooksΓ 1961.

[146] Rudnick (M.)ΓLeen (T.) et Hammerstrom (D.). – Hebbian feature discovery improves classifier efficiency. *In : Proc. of the International Joint Conference on Neural Networks*Γpp. 51–56.

[147] Rumelhart (D.E.)ΓHinton (G.E.) et Williams (R.J.). – Learning internal representations by error propagation. *In : Parallel Distributed Processing : Exploration in the Microstructure of Cognition*Γéd. par Rumelhart (D.E.) et McClelland (J.L.). – CambridgeΓUSAΓMIT PressΓ1986.

[148] Sacerdoti (E.D.). – Planning in a Hierarchy of Abstraction Spaces. *In : Proc. of the International Joint Conference on Artificial Intelligence*Γpp. 412–422. – StanfordΓCAΓUSAΓaoût 1973.

[149] Saffiotti (A.). – Some Notes on the Integration of Planning and Reactivity in Autonomous Mobile Robots. *In : Proc. of the American Association for Artificial Intelligence Symposium.*

[150] Saffiotti (A.)ΓRuspini (E.H.) et Konolige (K.). – Blending Reactivity and Goal-Directedness in a Fuzzy Controller. *In : Proc. of the IEEE International Conference on Fuzzy Systems*Γpp. 134–140. – San FranciscoΓmars 1993.

[151] Saffiotti (A.)ΓRuspini (E.H.) et Konolige (K.). – Robust Control of a Mobile Robot using Fuzzy Logic. *In : Proc. of EUFIT.*

[152] Saffiotti (A.)ΓRuspini (E.H.) et Konolige (K.). – Robust Execution of Robot Plans using Fuzzy Logic. *In : Proc. of the International Joint Conference on Artificial Intelligence, Workshop on Fuzzy Logic.* – ChamberyΓFranceΓ août 1993.

[153] Saffiotti (A.)ΓRuspini (E.H.) et Konolige (K.). – Using Fuzzy Logic for Autonomous Vehicle Motion Planning. *In : Proc. of the North American Fuzzy Logic Processing Society (NAFIPS).*

[154] Samuel (A.L.). – Some Studies in Machine Learning using the Game of Checkers. *IBM Journal on Research and Development*Γ1967Γpp. 601–617.

[155] Sanger (T.D.). – Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*Γvol. 2Γ1989Γpp. 459–473.

[156] Schiele (B.). – *Perception et Localisation pour un Robot Mobile.* – 46 Avenue Félix VialetΓ38031 Grenoble CedexΓDEAΓLIFIA-INPGΓjuin 1993.

[157] Schiele (B.) et Crowley (J.L.). – Certainty Grids : Perception and Localisation for a Mobile Robot. *In : Proc. of the International Workshop on Intelligent Robotic Systems*Γéd. par Crowley (J.L.) et Dubrawski (A.)Γpp. 159–166. – ZakopaneΓPolandΓjuillet 1993.

[158] Schiele (B.) et Crowley (J.L.). – A Comparison of Position Estimation Techniques Using Occupancy Grids. *In : Proc. of the IEEE International Conference on Robotics and Automation.* – San DiegoΓMay 1994.

[159] Schoppers (M.). – On Hierarchical Architectures for Robot Control Software. – Architecture for Intelligent Control Systems (AICS) Mailing ListΓ octobre 1992.

[160] Schwartz (J. T.) et Sharir (M.). – On the piano movers' problem: II. general techniques for computing topological properties of real algebraic manifold. *Advances in Applied Mathematics*Γvol. 4Γ1983Γpp. 298–351.

[161] Solla (S.A.)Γ Chun (Y. Le) et Denker (J.S.). – Optimal brain damage. *In : Advances in neural information processing systems*Γéd. par Touretzky (D.S.)Γpp. 598–605. – Morgan KaufmanΓ1990.

[162] Song (K.T.) et Tai (J.C.). – Fuzzy Navigation of a Mobile Robot. *In : Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*Γpp. 621–627. – RaleighΓjuillet 1992.

[163] Sutton (R.S.). – Learning to Predict by the Methods of Temporal Differences. *Machine Learning*Γvol. 3Γ1988Γpp. 9–44.

[164] Sutton (R.S.). – Integrated Architectures for LearningΓPlanningΓand Reacting Based on Approximating Dynamic Programming. *In : Proc. of the Seventh International Conference on Machine Learning*Γpp. 216–224.

[165] Sutton (R.S.). – Reinforcement Learning Architectures for Animats. *In : Proc. of the International Workshop on the Simulation of Adaptive Behavior : From Animals to Animats.* – MIT Press.

[166] Sutton (R.S.) et Barto (A.G.). – Time-Derivative Models of Pavlovian Reinforcement. *In : Learning and Computational Neuroscience : Foundations of Adaptive Networks*Γéd. par Gabriel (M.) et Moore (J.)Γchap. 12Γ pp. 497–537. – CambridgeΓMAΓUSAΓMIT PressΓ1990.

[167] Taix (M.). – *Planification de mouvements pour robot mobile non-holonome.* – Toulouse (F)ΓThèse de PhDΓLaboratoire d'Automatique et d'Analyse des SystèmesΓjanvier 1991.

[168] Takagi (T.) et Sugeno (M.). – Derivation of Fuzzy Control Rules from Human Operator's Control Action. *In : Proc. of the IFAC Symposium on Fuzzy Information, Knowledge Representation and Decision Analysis.* – MarseilleΓFranceΓjuillet 1983.

[169] Takagi (T.) et Sugeno (M.). – Fuzzy Identification of Systems and Its Applications to Modeling and Control. *IEEE Transactions on Systems, Man, and Cybernetics*Γvol. 15Γn° 1Γjanvier 1985Γpp. 116–132. – ISSN 0018-9472.

[170] Thorpe (C.E.). – Path Relaxation : Path Planning for a Mobile Robot. *In : Proc. of the American Association for Artificial Intelligence Conference*Γ pp. 318–321. – AustinΓUSAΓaoût 1984.

[171] Thrun (S.B.). – *Efficient Exploration in Reinforcement Learning.* – Rapport technique n° CMU-CS-92-102ΓPittsburghΓPennsylvania 15213-3890Γ School of Computer ScienceΓCarnegie Mellon UniversityΓjanvier 1992.

[172] Thrun (S.B.). – The Role of Exploration in Learning Control. *In : Handbook of Intelligent Control : Neural, Fuzzy and Adaptive Approaches*Γéd. par White (D.A.) et Sofge (D.A.). – FlorenceΓKentucky 41022ΓVan Nostrand ReinholdΓ1992.

[173] Thrun (S.B.). – Exploration and Model Building in Mobile Robot Domains. *In : Proc. of the IEEE International Conference on Neural Networks.* – San FranciscoΓCAΓUSAΓmars 1993.

[174] Thrun (S.B.) et Mitchell (T.M.). – *Lifelong Robot Learning.* – Rapport technique n° IAI-TR-93-7ΓBonnΓGermanyΓUniversität BonnΓInstitut für InformatikΓjuillet 1993.

[175] Thrun (S.B.) et Möller (K.). – Active Exploration in Dynamic Environments. *In : Advances in Neural Information Processing Systems 4*Γéd. par Moddy (J.E.)ΓHanson (S.J.) et Lippmann (R.P.). – San MateoΓCAΓUSAΓ Morgan KaufmannΓ1992.

[176] Tsukamoto (Y.). – An Approach to Fuzzy Reasoning Method. *In : Advances in Fuzzy Set Theory and Applications*Γéd. par Gupta (M.M.)ΓRagade (R.K.) et Yager (R.R.). – North-HollandΓ1979.

[177] Turk (M.) et Pentland (A.). – Eigenfaces for recognition. *Journal of Cognitive Neuroscience*Γvol. 3Γn° 1Γ1991Γpp. 71–86.

[178] Udupa (S.M.). – Collision Detection and Avoidance in Computer Controlled Manipulators. *In : Proc. of the International Joint Conference on Artificial Intelligence*Γpp. 737–749. – CambridgeΓMAΓUSAΓaoût 1977.

[179] Wallace (R.)ΓMatsuzaki (K.)ΓGoto (Y.)ΓCrisman (J.)ΓWebb (J.) et Kanade (T.). – Progress in Road-Following. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γpp. 1615–1621. – San FranciscoΓUSAΓavril 1986.

[180] Wallace (R.)ΓStentz (A.)ΓThorpe (C.)ΓMoravec (H.)ΓWhittaker (W.) et Kanade (T.). – First Results in Robot Road-Following. *In : Proc. of the International Joint Conference on Artificial Intelligence*Γpp. 1089–1095. – Los AngelesΓCAΓUSAΓaoût 1985.

[181] Wallner (Frank). – *Potential Field Based Locomotion Reflexes.* – 46 Avenue Félix VialetΓ38031 Grenoble CedexΓDEAΓLIFIA-INPGΓjuillet 1993.

[182] Wang (L.X.). – Training of Fuzzy Logic Systems using Nearest Neighborhood Clustering. *In : Proc. of the IEEE International Conference on Fuzzy Systems*Γpp. 13–17. – San FranciscoΓCAΓUSAΓmars 1993.

[183] Warren (C.W.). – Global Path Planning Using Artificial Potential Fields. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γpp. 316–321. – ScottsdaleΓUSAΓmai 1989.

[184] Watanabe (Y.) et Pin (F.G.). – Sensor-Based Navigation of a Mobile Robot Using Automatically Constructed Fuzzy Rules. *In : Proc. of the International Conference on Advanced Robotics*Γpp. 81–87. – TokyoΓJapanΓnovembre 1993.

[185] Watkins (C.J.C.H.) et Dayan (P.). – Q-Learning. *Machine Learning*Γvol. 8Γ n° 3-4Γ1992Γpp. 279–292. – ISSN 0885-6125.

[186] Whitehead (S.)ΓKarlsson (J.) et Tenenberg (J.). – Learning Multiple Goal Behavior via Task Decomposition and Dynamic Policy Merging. *In : Robot Learning*Γéd. par Connell (J.H.) et Mahadevan (S.)Γchap. 3Γpp. 45–77. – Kluwer Academic PublishersΓ1993. ISBN 1-7923-9365-1.

[187] Widrow (B.) et Hoff (M.E.). – Adaptive switching circuits. *In : Neurocomputing : Foundations of Research*Γéd. par Anderson (J.A.) et Rosenfeld (E.)Γ pp. 96–104. – CambridgeΓUSAΓThe MIT PressΓ1960.

[188] Yager (R.R.). – Expert Systems Using Fuzzy Logic. *In : An Introduction to Fuzzy Logic Applications in Intelligent Systems*Γéd. par Yager (Ronald R.) et Zadeh (Lofti A.)Γchap. 11Γpp. 1–26. – Kluwer Academic PublishersΓ 1992. ISBN 0-7923-9216-7.

[189] Zadeh (L.A.). – Fuzzy Sets. *Information and Control*Γ vol. 8Γ1965Γpp. 338–353.

[190] Zadeh (L.A.). – Knowledge Representation in Fuzzy Logic. *In : An Introduction to Fuzzy Logic Applications in Intelligent Systems*Γéd. par Yager (Ronald R.) et Zadeh (Lofti A.)Γchap. 11Γpp. 1–26. – Kluwer Academic PublishersΓ1992. ISBN 0-7923-9216-7.

[191] Zanni (Gaelle). – *Etudes de Méthodes Neuronales pour la Navigation Réactive*. – 46 Avenue Félix VialetΓ38031 Grenoble CedexΓDEAΓLIFIA-INPGΓ juin 1993.

[192] Zapata (R.). – *Quelques Aspects Topologiques de la Planification de Mouvements et des Actions Reflexes en Robotique Mobile*. – Thèse de PhDΓ Université de MontpellierΓFranceΓJuillet 1991.

[193] Zhang (J.) et Bohner (P.). – A Fuzzy Control Approach for Executing Subgoal Guided Motion of a Mobile Robot in a Partially-Known Environment. *In : Proc. of the IEEE International Conference on Robotics and Automation*Γpp. 545–550 Volume 2. – AtlantaΓUSAΓmai 1993.