Mises en œuvre de Commandes Neuronales par Rétropropagation Indirecte : Applications à la Robotique Mobile

Patrick Henaff

THESE

présentée à

L'UNIVERSITE PIERRE ET MARIE CURIE

par

Patrick HENAFF

pour obtenir le titre de

DOCTEUR DE L'UNIVERSITE
PIERRE ET MARIE CURIE

Spécialité ROBOTIQUE

Mises en Œuvre de Commandes Neuronales par Rétropropagation Indirecte : Applications à la Robotique Mobile

---

Soutenue le 28 Juin 1994

JURY

Mr. P. COIFFET	Directeur de recherche C.N.R.S.	*Président*
Mr. W. KHALIL	Professeur à L'Ecole Centrale de Nantes	*Rapporteur*
Mr. T. MAURIN	Professeur à L'E.N.S. de Cachan	*Rapporteur*
Mr. M. MILGRAM	Professeur à l'Université Paris 6	*Examinateur*
Mr. J. RABIT	Professeur à l'Université Paris 11	*Examinateur*
Mme. de SAINT BLANCARD	Ingénieur de Recherche à P.S.A.	*Examinateur*

C'était le dernier jour.....
Le soleil d'Océanie s'était levé aussi radieux qu'à l'ordinaire sur ≪Tahiti la délicieuse≫; ce que souffrent dans leur cœur les hommes qui passent et disparaissent n'a rien de commun avec l'éternelle nature, et n'entrave jamais ses fêtes inconscientes.

Pierre Loti *"Le mariage de Loti".*

≪ E hari te fau
E toro te fauro
E no te taata≫
(Le palmier croîtra
Le corail s'étendra
Mais l'homme périra)

à toi papa,
pour tout ce que tu as été,
pour tout ce que tu m'as appris,
et pour tout ce que tu resteras,
à jamais dans mon cœur.

Le breton est-il ma langue maternelle? Non : je suis né à Nantes où on ne le parle pas.... Suis-je même breton? Vraiment je le crois. Mais de "pure race", qu'en sais-je et qu'importe... Séparatiste? Autonomiste? Régionaliste? Oui et non : différent. Mais alors vous ne comprenez plus. Qu'appelons être breton? Et d'abord, pourquoi l'être?
...Français d'état civil, je suis nommé français, j'assume à chaque instant ma situation de français : mon appartenance à la Bretagne n'est en revanche qu'une qualité facultative que je puis parfaitement renier ou méconnaître. Je l'ai d'ailleurs fait. J'ai longtemps ignoré que j'étais breton.... Français sans problème, il me faut donc vivre la Bretagne en surplus, ou pour mieux dire en conscience : si je perds cette conscience, la Bretagne cesse d'être en moi; si tous les bretons la perdent, elle cesse absolument d'être. La Bretagne n'a pas de papiers. Elle n'existe que si à chaque génération des hommes se reconnaissent bretons. A cette heure, des enfants naissent en Bretagne. Seront-ils bretons? Nul ne le sait. A chacun, l'âge venu, LA DECOUVERTE OU L'IGNORANCE....

Morvan Lebesque (chanté par Tri Yann)
Comment peut-on être breton?
"essai sur la démocratie française"

Ur bannig traou dous
yehed mad

Je remercie :

Mr. Jean Claude Guinot, Professeur de l'Université Pierre et Marie Curie et Directeur du Laboratoire de Robotique de Paris, pour m'avoir accueuilli dans son laboratoire. Si le L.R.P est ce qu'il est, c'est grâce à lui. Sa personnalité scientifique et humaine est un exemple pour beaucoup de "thésards".

Mrs. Maurice Milgram et Jacques Rabit pour m'avoir co-encadré et accepté au sein de leurs équipes respectives. Leur aide et leur soutien m'ont été précieux pour accomplir ce travail.

Mrs. Wisama Khalil, Professeur de l'Ecole Centrale Nantes, et Thierry Maurin Professeur de l'Ecole Normale Supérieure de Cachan, pour avoir accepté d'être rapporteur de mon travail, et de le faire dans un délai aussi court.

Mr. Philippe Coiffet, Directeur de Recherches au C.N.R.S et chef du groupe Commande et Apprentissage des Robots au L.R.P . Sa participation au jury en tant que Président est un honneur pour moi et sa personnalité scientifique un exemple.

Mme. de SAINT BLANCARD, Ingénieur de Recherche à P.S.A pour sa participation au jury en tant que représentant industriel et examinateur.

Tout le L.R.P, mais surtout Eric, et surtout Guillaume, et puis aussi Stéphane et Fethi, et... Oh et puis j'embrasse Nathalie,... et puis bref tout le monde quoi...

Et puis bien sur Isabelle et bébé Gwendal qui ne m'ont pas beaucoup vu depuis ces derniers mois.

Je ne remercie pas :

Ceux qui prennent le pont de Bezons pour aller tous les matins en voiture vers Paris. D'ailleurs, je les soupçonne d'insister car j'ai la désagréable impression de les retrouver sur la N 118 tous les soirs.

Sommaire

Table des Matières i

Liste des Figures vii

1 Introduction Générale 1
1.1 Les Modèles Statiques 1
1.2 Les Modèles Dynamiques 2

I Réseaux de Neurones Formels et Apprentissage 5

2 Généralités sur les Réseaux de Neurones 7
2.1 Structure interne du neurone 7
2.2 Structure interne du réseau 8
2.2.1 Qu'est ce qu'un réseau de neurones ? 8
2.2.2 Construction d'un réseau 9
2.3 Calcul de l'état d'un réseau 10
2.4 Algorithme d'apprentissage 11
2.5 Inconvénients de la rétropropagation 14

3 Quelques Utilisations des RNF pour la commande 17
3.1 Commande cartésienne d'un bras manipulateur (Simulation) 17
3.2 Commande cartésienne d'un bras manipulateur (Expérimentation) 19
3.3 Contrôle dynamique d'un pendule inverse avec un *"maître humain"* 20
3.4 Contrôle dynamique d'un pendule inverse (simulation) 22
3.5 Contrôle d'un Robot mobile (Expérimentation) 23
3.6 Contrôle de la locomotion d'un bipède (simulation) 24
3.7 Apprentissage de Contrôle d'Admittance (expérimentation) 25
3.8 Conclusion du chapitre 26

4 l'Apprentissage Indirect 29
4.1 Inconvénients de la Rétropropagation Directe 29
4.2 La rétropropagation indirecte 31
4.2.1 Principe d'utilisation hors-ligne 31
4.2.2 Principe d'utilisation en-ligne 32

4.2.3 Différence entre apprentissage Hors-Ligne et En-Ligne 34
4.2.4 Formulation mathématique 35
4.3 Conclusion 36
4.4 Problème Géométrique avec Contraintes 37
4.4.1 Description du problème 37
4.4.2 Formulation mathématique 38
4.4.3 Structure d'apprentissage 40
4.4.4 Apprentissage Hors ligne 40
4.4.5 Résultats de l'apprentissage 41
4.4.6 Apprentissage en ligne 46
4.4.7 Conclusion 47

II Application à la commande d'un Robot Mobile 49

5 Introduction à la commande Cartésienne d'un Robot Mobile 51
5.1 Description du Robot Mobile 51
5.2 Problème de la commande en position et en orientation 52
5.3 Description du Site expérimental 53

6 Commande par Apprentissage Direct 55
6.1 Analyse du problème 55
6.1.1 Structure de Contrôle 55
6.1.2 Classes de manoeuvre 56
6.2 Apprentissage 57
6.2.1 Surfaces de commande avant apprentissage 58
6.2.2 Surfaces de commande après apprentissage 59
6.2.3 Valeur des poids du réseau 61
6.3 Simulation du Contrôle du robot 61
6.4 Amélioration des performances du réseau 62
6.5 Validation Expérimentale 66
6.6 Conclusion 67

7 Commande par Apprentissage Indirect Hors-Ligne 69
7.1 Formulation Mathématique 69
7.2 Apprentissage 71
7.2.1 Architecture 71
7.2.2 Base d'Apprentissage 71
7.2.3 Résultats 72
7.2.4 Valeur des poids du réseau 73
7.3 Simulation de la Commande 73
7.4 Amélioration par modification du retour d'état angulaire 74
7.5 Expérimentation de la Commande 74
7.6 Conclusion 79

8 Commande par Apprentissage Indirect En-Ligne **81**
8.1 Architecture 81
8.2 Simulation 82
8.2.1 Valeur des poids du réseau après apprentissage en-ligne 85
8.2.2 Influence des paramètres d'apprentissage 85
8.2.3 Comportement du robot après apprentissage 85
8.3 Expérimentations 86
8.3.1 Commande par un réseau formé sur simulateur 86
8.3.2 Apprentissage en-ligne sur le robot réel 89
8.3.3 Valeur des poids 93
8.3.4 Commande après apprentissage sur le robot réel 94
8.4 Conclusion sur l'apprentissage En-Ligne 94
8.5 Conclusion de la seconde partie 95

III Vers la locomotion Articulée **97**

9 Contrôle de l'équilibre dynamique d'un bipède **99**
9.1 Introduction 100
9.2 Modélisation du bipède 100
9.2.1 Le robot Japonnais WL-12 100
9.2.2 Modèle utilisé 100
9.2.3 Modèle de déplacement 101
9.3 Equilibre Dynamique du Bipède 102
9.3.1 Rappels sur l'équilibre d'un système :ZMP 102
9.3.2 Application à l'équilibre du bipède 104
9.4 Structure de Commande 106
9.5 Apprentissage Hors-ligne 107
9.5.1 Choix du critère et calcul du gradient 107
9.5.2 Construction de la base d'apprentissage 108
9.5.3 Résultats de l'apprentissage 110
9.6 Apprentissage En-Ligne 111
9.7 Conclusion 113

IV Conclusion Générale **115**

10 Conclusion et Perspectives **117**

Références bibliographiques **121**

Annexes **127**

A Valeurs des poids après apprentissage pour la commande du robot mobile 129
A.1 Par apprentissage direct (réseau 3-9-2) 129
A.2 Par apprentissage indirect hors-ligne (réseau 3-9-2) 131
A.3 Par apprentissage indirect en-ligne sur simulateur 132
A.4 Par apprentissage indirect en-ligne sur le robot 134
A.4.1 Après le premier apprentissage 134
A.4.2 Après le deuxième apprentissage 136
A.4.3 Après le troisième apprentissage 137

Figures

2.1 Détail du fonctionnement d'un neurone 8
2.2 Exemple de structure d'un réseau 9
2.3 Dépendance dans un réseau multicouche (d'après [Le Cun 87]) 11
2.4 Structure de la rétropropagation 14
2.5 L'apprentissage par coeur 15
2.6 Le problème du sur-apprentissage 16

3.1 17
3.2 Structure de la commande 18
3.3 structure de la commande neuronale de l'INTELLEDEX 605T 19
3.4 Cart-Pole System 20
3.5 Principe de l'apprentissage supervisé par l'opérateur 21
3.6 Principe de l'apprentissage par linéarisation 22
3.7 Principe de commande de la position et de l'orientation du robot mobile . . 23
3.8 Exemple de trajectoires de la base d'apprentissage 23
3.9 modèle simplifié d'un bipède 24
3.10 Tâche d'insertion 25

4.1 Structure de commande par retour d'état 30
4.2 Structure de l'apprentissage direct 31
4.3 Structure de l'apprentissage indirect hors-ligne 32
4.4 Structure de l'apprentissage indirect en-ligne 33
4.5 Illustration de la différence entre Hors-Ligne et En-Ligne dans l'espace des phases du système 34
4.6 le problème test 38
4.7 Structure de l'apprentissage 40
4.8 Apprentissage 42
4.9 Apprentissage 44
4.10 Résultat après l'apprentisage Hors-Ligne 45
4.11 Résultat pendant l'apprentisage En-Ligne 46
4.12 Résultat après l'apprentissage En-Ligne 47

5.1 Commande en position du robot mobile 52
5.2 Robot mobile ROMEO 53
5.3 Site expérimental utilisé 54

6.1 Commande du chariot mobile 56
6.2 Les différentes classes de manoeuvre 56
6.3 Vitesses des roues avant apprentissage ($\theta = 0$) 58
6.4 Vitesses linéaire et angulaire avant apprentissage ($\dot{\theta} = 0$) 58
6.5 Vitesses des roues après apprentissage ($\theta = 0^o$) 59
6.6 Vitesses linéaire et angulaire après apprentissage ($\theta = 0^o$) 59
6.7 Vitesses des roues après apprentissage ($\theta = 180^o$) 60

6.8 Vitesses linéaire et angulaire après apprentissage ($\theta = 180^o$) 60
6.9 Simulation pour $X_M = 4m$, $Y_M = 3m$, $\theta = 180^o$ 61
6.10 Simulation pour $X_M = 0m$, $Y_M = 0,5m$, $\theta = 0^o$ 63
6.11 Simulation pour $X_M = 3m$, $Y_M = 4m$, $\theta = 180^o$ 64
6.12 Simulation pour $X_M = 0m$, $Y_M = 0.5m$, $\theta = 0^o$ 65
6.13 Expérimentation pour $X_M = 4m$, $Y_M = 3m$, $\theta = -180^o$ 66
6.14 Expérimentation pour $X_M = 0m$, $Y_M = 0,5m$, $\theta = 0^o$ 67

7.1 Structure du contrôle en position et en orientation 69
7.2 Apprentissage Hors-ligne 72
7.3 Evolution de l'apprentissage 73
7.4 Evolution des Erreurs 73
7.5 Simulation pour $X_M = 4m$, $Y_M = 3m$, $\theta = 180^o$ 75
7.6 Simulation pour $X_M = 0.5m$, $Y_M = 1m$, $\theta = 0^o$ 76
7.7 Expérimentation pour $X_M = 4m$, $Y_M = 3m$, $\theta = 180^o$ 77
7.8 Expérimentation pour $X_M = 0,5m$, $Y_M = 1m$, $\theta = 0^o$ 77
7.9 Expérimentation pour un demi tour 78

8.1 Structure d'apprentissage En-ligne de la commande en position de ROMEO . 81
8.2 Premier apprentissage en-ligne sur simulateur 83
8.3 Second apprentissage en-ligne sur simulateur 84
8.4 Influence du pas d'apprentissage sur la trajectoire du robot pendant l'apprentissage en-ligne 85
8.5 Trajectoires cartésiennes du robot après apprentissage sur simulateur 86
8.6 Simulation après apprentissage sur simulateur 87
8.7 Expérimentation après apprentissage sur simulateur 88
8.8 Premier apprentissage réel en-ligne 90
8.9 Second apprentissage réel en-ligne 91
8.10 Troisième apprentissage réel en-ligne 92
8.11 Expérimentation après apprentissage réel en-ligne 94

9.1 Bipède Japonais WL12 101
9.2 Modélisation de la répartion de masses du WL-12 102
9.3 Modèle de bipède que nous utilisons 103
9.4 Structure de contrôle du bipède 107
9.5 Shéma de construction des exemples dans l'espace articulaire 109
9.6 Distribution de l'extrémité de la patte 1 obtenue par le processus aléatoire . 110
9.7 Schéma de construction des exemples dans l'espace cartésien 110
9.8 Trajectoire du ZMP au cours d'une enjambée. 111
9.9 Evolution de la trajectoire du ZMP pendant l'apprentissage En-Ligne 112

Chapitre 1

Introduction Générale

Depuis moins d'une dizaine d'années, les réseaux de neurones suscitent un intérêt grandissant dans tous les domaines où interviennent les notions de décision, classification, reconnaissance, statistique, traitement qualitatif,....
Ils ont vite révélé les performances suivantes :

- rapidité de calcul de la sortie en fonction de l'entrée,
- très bons comportements en univers bruité,
- capacités d'apprentissage de lois d'entrée/sortie non analytiques,
- capacités d'identification et de prédiction
- classification des entrées.

Ces propriétés sont très intéréssantes pour la commande des systèmes, et plus particulièrement celui des robots, qui supporte des contraintes importantes : dynamique non-linéaire couplée et difficilement identifiable, bruits métrologiques, contraintes temps réel.

Plusieurs types de modèles de réseaux de neurones sont utilisés depuis 1990 pour des applications de commande, de filtrage ou de prédiction. On distingue principalement les *modèles statiques* et les *modèles dynamiques*. (D. T. Pham et X. Liu font dans [*Pham 94*] une étude comparative sur un exemple de prédiction.)

1.1 Les Modèles Statiques

Ce sont essentiellement les perceptrons multicouches. Ils sont dits statiques car leur sortie n'évolue pas dans le temps lorsque l'entrée est fixée. C'est à dire, que la relation entrée/sortie qu'ils réalisent est considérée comme instantanée. Ces réseaux ont montré leur efficacité pour beaucoup d'application en commande (voir chapitre 2), mais aussi en classification et en reconnaissance de forme (citons par exemple [*Desj 93*], [*Novi 91*], [*Le Cun 87*]). Ils sont en général entrainés par l'algorithme de rétropropagation du gradient.

1.2 Les Modèles Dynamiques

Ils tiennent compte d'une dimension temporelle entre l'entrée et la sortie du réseau, par des connexions en boucles. B. Gas ([*Gas 94*]) fait une synthèse détaillée des différentes structures et techniques d'apprentissage.
C'est généralement une partie de la sortie qui est rebouclée sur l'entrée à travers des unités induisant un retard. On parle alors de réseaux récurrents ([*Nerr 94*]). Dans ce cas, les cellules de la couche d'entrée sont divisées en deux groupes. L'un reçoit l'état de l'environnement (entrée réelle), l'autre celui de la sortie retardée. Pour une entrée fixée, la sortie va évoluer dans le temps, générant ainsi une séquence temporelle. Ces réseaux sont entraînés par l'algorithme de rétropropagation, modifié pour tenir compte du rebouclage (rétropropagation temporelle). Ils sont utilisés principalement pour des problèmes d'identification, de prédiction et de filtrage ([*Nerr 93*]). Des applications existent néanmoins pour la commande des sytèmes : citons [*Nguy 91*], [*Riva 93*] [*Narr 91*]. Elles sont cependant plus rares que pour les réseaux statiques.

Il n'existe pas, à notre avis, de méthode qui permette de décider s'il convient d'utiliser un réseau récurrent ou un réseau non récurrent pour un problème donné de commande. De plus, la littérature offre peu d'exemples de mise en œuvre expérimentale de commande neuronale de robot avec des réseaux récurrents, comparativement aux réseaux statiques. Les modèles dynamiques étant plus souvent utilisés pour le traitement du signal (identification et prédiction). Enfin, les méthodes d'apprentissage en-ligne à base de réseaux dynamiques utilisent un réseau récurrent supplémentaire qui émule le système à commander ([*Nguy 91*]). La structure du neuro-contrôleur nécessite alors plusieurs réseaux et devient à notre avis trop complexe [*Mart 91*]. Or notre travail s'insère dans un projet d'intégration, dans une bibliothèque d'actions, de commandes bas niveau pour l'exécution de missions complexes par un robot mobile. C'est la raison pour laquelle nous proposons d'approfondir l'utilisation des réseaux statiques et notamment la rétropropagation en-ligne pour la commande de sytèmes robotiques.

Ce mémoire présente les résultats d'une étude sur l'utilisation de réseaux de neurones formels pour la commande de systèmes robotiques mobiles à roues et à pattes. Les réseaux utilisés sont de type perceptrons multicouches. L'algorithme d'apprentissage est la rétropropagation du gradient.

Le travail que nous avons mené s'est voulu exploratoire et expérimental, tant du point de vue connexionniste que robotique. Nous avons cherché à appliquer de la façon la plus simple possible les perceptrons multicouches, et la technique de rétropropagation du gradient, dans un but de mise en œuvre de lois de contrôle neuronales pour des problèmes classiques, mais non triviaux, de robotique mobile. Le but était de dégager des principes d'utilisation pour la commande des systèmes, et d'en évaluer les limites.

Ce mémoire s'adresse donc principalement au roboticien qui cherche des explications et

des exemples rationnels sur cette technique neuronale particulière.

Il se divise en **trois parties**:

- La **première** concerne les réseaux de neurones et l'algorithme d'apprentissage de rétropropagation.
- La **deuxième** correspond à la mise en œuvre expérimentale de lois de commandes neuronales pour la commande en position et en orientation d'un robot mobile à roues non-holonome.
- La **troisième** aborde un problème de locomotion articulée en simulation, que nous résolvons grâce aux techniques acquises dans la seconde partie.

Nous présentons au **chapitre 2** les RNF[1] de type perceptrons multicouches et l'algorithme d'apprentissage usuel dit de rétropropagation du gradient dont le but est de minimiser un coût quadratique calculé en sortie du réseau. Généralement ce coût est une erreur quadratique entre la sortie du réseau et une sortie désirée. On l'appelle alors RétroPropagation Directe (RPD). Nous en rappelons les principes fondamentaux de fonctionnement.

Le **chapitre 3** présente sommairement quelques applications issues de la littérature sur la commande neuronale des robots. Ca ne peut pas être un état de l'art dans la mesure ou les travaux dans ce domaine sont rares et très différents par leur nature. On montre simplement l'intérêt de cette technique à travers diverses méthodes de mise en œuvre. Nous soulignons les inconvénients principaux de la RPD pour la commande, notamment l'utilisation d'une sortie désirée qui nécessite une inversion de modèle et qui ne permet pas un apprentissage En-Ligne. La conclusion de ce chapitre nous permet d'esquisser un schéma particulier d'apprentissage, appelé RétroPropagation Indirecte (RPI), qui doit éviter ces inconvénients.

Le **chapitre 4** présente la RétroPropagation Indirecte. L'idée de base est d'utiliser la rétropropagation sans sorties désirées mais avec un critère quadratique exprimant l'objectif et les contraintes du problème. Le Réseau de Neurones Formels va, pendant son apprentissage, tenter de minimiser ce critère sur un grand nombre d'exemples représentatifs des différents états a priori possibles du système. Après avoir souligné les avantages de cette méthode, nous présentons deux concepts d'utilisation :

- La RPI Hors-Ligne qui utilise un modèle du robot pour calculer un correcteur neuronal quasi-optimal par rapport au modèle,
- La RPI En-Ligne qui permet d'adapter En-Ligne le réseau obtenu Hors-Ligne, afin de le rendre optimal par rapport au robot réel. Nous montrons que cette technique s'apparente à un contrôle adaptatif.

A titre d'illustration nous appliquons ces concepts à un cas d'école.

[1] Réseaux de Neurones Formels

Le **chapitre 5** introduit pour la seconde partie le problème du contrôle de la position et de l'orientation du robot mobile en insistant sur son caractère non-holonome. Il présente aussi le site expérimental et le robot mobile ROMEO du L.R.P. .

Au **chapitre 6**, nous proposons une solution qui utilise l'apprentissage direct. Nous décomposons le problème en différentes classes de manoeuvres caractérisées chacune par des configurations similaires de l'entrée et une commande désirée. Le RNF apprend à séparer ces classes, et à calculer pour chacune la commande correspondante. En se déplaçant, le robot passe ainsi par les configurations apprises. L'ensemble des commandes fournies par le RNF fait alors converger le robot vers la configuration désirée.
Cette technique appartient à la fois à la séparation de classes en connexionisme et à l'apprentissage de règles floues en commande.

Le **chapitre 7** applique la RPI Hors-ligne. Nous proposons un critère de distance quadratique dans l'espace des configurations, et une solution de linéarisation pour le calcul de son gradient, rendu difficile par l'aspect non-holonome du robot. Nous montrons qu'il est possible d'améliorer fortement les résultats en modifiant simplement le critère. Nous soulignons l'importance de la méthode de construction de la base d'apprentissage. Enfin, nous validons expérimentalement la méthode.

Nous utilisons au **chapitre 8** la RPI En-ligne. Nous comparons expérimentalement un réseau obtenu par un apprentissage sur un simulateur du robot avec un autre obtenu par un apprentissage sur le robot réel. Nous montrons la différence de comportement en insistant sur la caractéristique d'adaptivité du contrôle. Nous constatons que le RNF maîtrise un ensemble important de configurations, alors qu'il n'a observé pendant l'apprentissage que trois trajectoires du robot.

L'ensemble des résultats obtenus nous permet enfin de dégager des conclusions sur l'expérience acquise au niveau de la méthodologie de mise en œuvre d'une commande neuronale basée sur la rétropropagation, et d'envisager la commande dynamique de systèmes mécaniques.

Le **chapitre 9** aborde la robotique mobile à patte. Nous prenons l'exemple simplifié d'un bipède dont il faut contrôler l'équilibre dynamique par l'accélération articulaire du tronc. Un critère exprimant directement l'équilibre dynamique est utilisé. Nous insistons comme au chapitre 7 sur l'importance de la méthode de construction de la base d'apprentissage, en proposant ici une construction des exemples à partir de l'espace articulaire ou de l'espace cartésien. Nous comparons ensuite la commande neuronal avec un contrôle analytique simple. Nous faisons apparaître la capacité du RNF à maintenir l'équilibre dynamique du système sans aucune connaissance a priori sur les trajectoires des pattes. Nous appliquons ensuite la RPI En-Ligne sur un réseau obtenu Hors-Ligne. Nous montrons alors que cela permet d'accroître fortement l'apprentissage de l'équilibre, et qu'il y a une adaptation du correcteur pour les mouvements effectués.

Partie I

Réseaux de Neurones Formels et Apprentissage

Chapitre 2

Généralités sur les Réseaux de Neurones

Nous proposons dans ce chapitre une présentation des réseaux de neurones utilisés . Ils sont de type perceptrons multicouches associés à l'algorithme d'apprentissage dit de rétropropagation du gradient.
Il doit être clair pour le lecteur que nous n'aborderons pas dans ce rapport les autres modèles connexionnistes (réseaux récurrents ou bouclés, réseaux à temps, réseaux neuro-flous...). Le but de notre travail est en effet d'approfondir les techniques d'utilisation des perceptrons multicouches et de la rétropropagation pour la commande des systèmes robotiques.

2.1 Structure interne du neurone

Un *neurone formel* se veut un modèle mathématique d'un neurone biologique. Le modèle utilisé est celui de Mac Culloch et Pitts ([*Pitt 43*]). C'est un système élémentaire qui a des entrées pondérées et une seule sortie. Il fonctionne de la manière suivante :

1. Il calcule l'addition pondérée de ses entrées. Le résultat obtenu représente *l'activité* du neurone.
2. Il opère une fonction discriminante (seuil) sur cette activité. En général, cette fonction est une sigmoïde.
3. Le résultat obtenu est la sortie du neurone.

La figure 2.1 représente le modèle de neurone que nous utilisons. Dans ce modèle[1],

- a_i est l' activité du neurone i.
- x_i est la sortie du neurone i.
- x_j est la sortie du neurone j connecté au neurone i.

[1]Ces notations peuvent être différentes dans la littérature

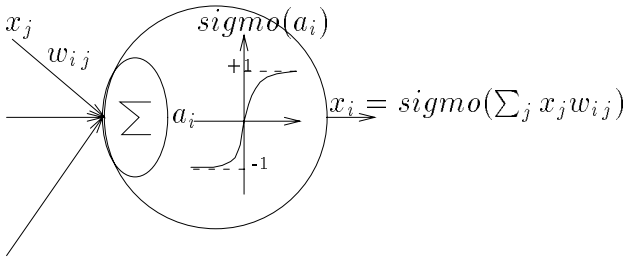

Fig. 2.1: *Détail du fonctionnement d'un neurone*

- w_{ij} est le poids de la connexion du neurone j vers le neurone i.
- la fonction discriminantes sigmoïde est telle que $sigmo(u) = \frac{1-e^{-u}}{1+e^{-u}}$. On l'utilise pour ses propriétés d'inversibilité et de différentiabilité.

Remarque

La sigmoïde est la fonction discriminante la plus courante. Cependant d'autres fonctions existent pour des applications spécifiques : fonction seuil, tangente hyperbolique, fonction linéaire par morceaux, sinusoïdale, gaussienne,.. .

2.2 Structure interne du réseau

2.2.1 Qu'est ce qu'un réseau de neurones ?

Un réseau de neurones est un ensemble de neurones connectés entre eux. C'est un modèle informatique[2]. On parle alors de réseaux neuromimétiques ([*CNRS 91*]). La structure des connexions peut être :

- *Totale* (un neurone est connecté à l'ensemble des neurones du réseau y compris lui même).
- *Partielle* (un neurone est connecté à un sous ensemble défini du réseau).

Nous utiliserons pour tout ce qui va suivre, une connexion partielle structurée en couche où chaque neurone d'une couche à sa sortie connectée à tous les neurones de la couche suivante, et son entrée composée des sorties de tous les neurones de la couche précédente (voir figure 2.2).
C'est une structure de type **perceptrons multicouches**. On parle aussi de réseau "feed-forward" puisque l'information ne se diffuse dans le réseau que dans un seul sens : de l'entrée vers la sortie.

[2]Quelques prototypes de construction *hardware* de structures neuronales existent ([*Noss 92*] [*Forn 94*]). Citons en France : ARIANE du Laboratoire d'Informatique de l'Ecole Polytechnique, et NeuroGen du Laboratoire de Conception de Systèmes Intégrés de l'INPG

La figure 2.2 représente un réseau à une couche cachée de 6 neurones, une couche d'entrée de 9 neurones et une couche de sortie de 6 neurones. On parle de réseau 9-6-6. Il possède $9\times6 + 6\times6 = 90$ connexions. On remarquera à la fois l'homogénéité et le nombre élevé des

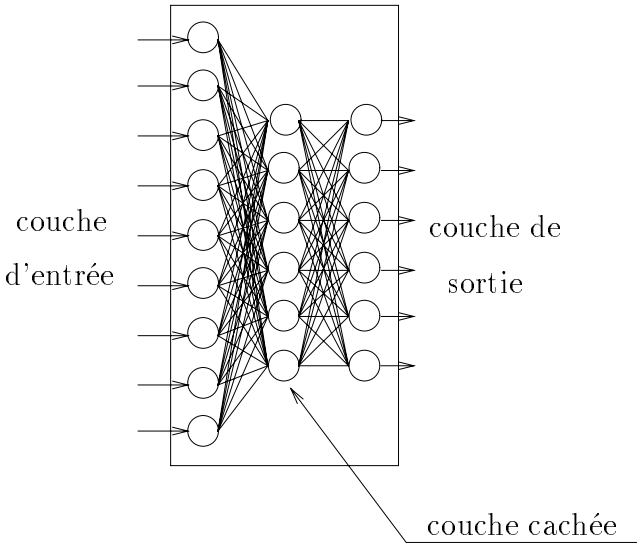

Fig. 2.2: *Exemple de structure d'un réseau*

connexions malgré la petite taille du réseau. Ces connexions sont pondérées par des poids (poids synaptiques en référence à la neurobiologie). C'est cet ensemble de poids qu'il faut déterminer pour obtenir un réseau capable de résoudre le problème considéré.

2.2.2 Construction d'un réseau

La construction (ou l'initialisation) d'un réseau est la phase de choix de sa structure et de la valeur de ses poids. Choisir une structure consiste à fixer définitivement le nombre des couches et leurs tailles. Les tailles des couches d'entrée et de sortie sont en général imposées par le nombre de paramètres du problème. Seuls donc, sont libres les tailles et le nombre des couches cachées. Cette liberté dans le choix de la structure interne du réseau laisse souvent perplexe le "non-initié" puisque il va être "assez arbitraire et laissé au savoir faire de l'utilisateur".
Cependant, quelques remarques sont à faire :

- Dans la pratique, on constate souvent qu'au delà de deux couches cachées, l'apprentissage et l'utilisation du réseau deviennent délicats.

- L'expérience montre qu'un réseau de plusieurs couches cachées peut se ramener à un réseau équivalent de 2 couches cachées de plus grandes tailles.

- Dans le cas de la commande des systèmes, les tailles des couches cachées sont souvent liées à celle du vecteur d'état.

- La pratique montre que pour un problème donné, il existe un ensemble de réseaux optimaux pour le résoudre. C'est à dire que l'apprentissage sera plus ou moins long et/ou plus instable si le réseau est trop petit ou trop grand.

Une fois donc choisie la structure du réseau, il faut fixer de la valeur initiale des poids. Ces valeurs peuvent être aléatoires ou choisies par l'utilisateur. Dans le cas d'une initialisation aléatoire, elles sont bornées à deux valeurs opposées. Dans le cas où elles sont imposées, elles doivent dépendre de la connaissance a priori du problème et de la façon dont le réseau va le traiter.
Cependant, il convient là aussi de faire quelques remarques :

- Les valeurs initiales des poids influencent considérablement la qualité de l'apprentissage. Un apprentissage non convergent pas, peut très bien le devenir avec une initialisation différente.

- En général, une répartition aléatoire des poids donne de moins bons résultats qu'un choix a priori de ceux-ci.

- En commande, il est malheureusement très difficile de faire ce choix a priori puisqu'on ne sait pas comment le réseau peut utiliser l'état de son entrée pour calculer la sortie.

Lorsque tous ces choix ont été faits, il convient de calculer par apprentissage la valeur optimale des poids pour obtenir le "bon réseau" devant résoudre le problème proposé. Il est important de savoir auparavant comment un réseau calcule sa sortie à partir de l'entrée qui lui est présentée. C'est l'objet du paragraphe suivant.

2.3 Calcul de l'état d'un réseau

C'est la phase dite de **propagation** qui consiste à calculer l'activité de tous les neurones à partir du vecteur d'entrée qui est ainsi **propagé** vers la sortie.
L'activité du neurone i est définie par la formule :

$$a_i = \sum_{j=0}^{n} w_{ij} x_j$$

et sa sortie par la suivante :

$$x_i = f(a_i)$$

où $f(x)$ est la fonction discriminante (sigmoïde).

- Pour un neurone de la couche d'entrée, x_j est l'élément j du vecteur d'entrée.

- Pour un neurone de la couche de sortie, x_i est l'élément i du vecteur de sortie.

L'activité du réseau est une notion importante. Le paragraphe suivant montre que dans la phase d'apprentissage, elle permet de modifier l'ensemble des poids à partir d'une fonction de coût calculée en sortie.

2.4 Algorithme d'apprentissage

C'est la phase de calcul de l'ensemble des poids. Il permet de trouver un réseau optimal pour résoudre le problème considéré. Nous décrivons ici l'algorithme dit de **rétropropagation du gradient de l'erreur** qui est le plus utilisé. Le but de la rétropropagation est d'obtenir d'un réseau la **généralisation** d'une fonction décrite par un ensemble de couples entrée-sortie appelés **exemples**. Ceux-ci doivent être suffisamment représentatifs de la fonction pour que l'apprentissage permette au réseau de la généraliser à partir de la base d'exemples considérée.

Pour obtenir cette généralisation, la rétropropagation minimise un coût par rapport à l'ensemble des poids, c'est à dire qu'elle trouve le minimum global d'une fonction dans un espace de dimension égale au nombre de connexions. La représentativité des exemples doit donc être suffisante pour couvrir tout l'espace des poids.
Pour l'apprentissage, on dispose d'une base d'exemples constituée par un ensemble de couples entrée-sortie désirée. On présente au réseau le vecteur d'entrée et en sortie le vecteur de sortie désiré. Par propagation de l'état de l'entrée, on calcul l'état de la sortie du réseau (appelé sortie actuelle). Cet état est comparé au vecteur de sortie désirée. Un calcul de dérivées partielles [*Le Cun 87*] montre qu'il suffit de calculer le gradient de cette erreur par rapport à la sortie du réseau pour connaître, pour tout neurone du réseau, le gradient de sa sortie par rapport aux poids qui le connectent à la couche précédente (voir 2.3).
Nous présentons ci-après les grandes lignes de ce calcul. Soit ε^2 l'erreur quadratique calculée

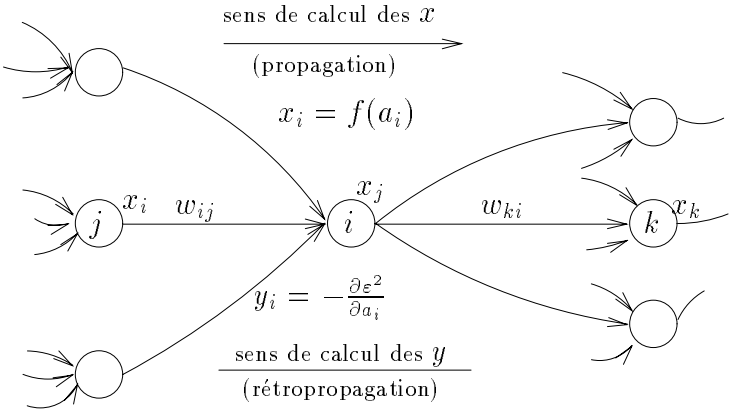

Fig. 2.3: *Dépendance dans un réseau multicouche (d'après [Le Cun 87])*

en sortie de réseau (pour un exemple de la base d'apprentissage) :

$$\varepsilon^2 = \sum_{sorties} (d - x(W))^2$$

où d est la sortie désirée et $x(W)$ la sortie dite actuelle calculée par propagation de l'entrée et dépendante de la matrice W des poids.
Minimiser ce coût par rapport aux poids entraîne le calcul du gradient

$$\frac{\partial \varepsilon^2}{\partial w_{ij}}$$

où w_{ij} est le poids de la connexion du neurone j vers le neurone i.
Or

$$\frac{\partial \varepsilon^2}{\partial w_{ij}} = \frac{\partial \varepsilon^2}{\partial a_i}\frac{\partial a_i}{\partial w_{ij}} = \frac{\partial \varepsilon^2}{\partial a_i}x_j$$

Et

$$\frac{\partial \varepsilon^2}{\partial a_i} = \frac{\partial \varepsilon^2}{\partial x_i}\frac{\partial x_i}{\partial a_i} = \frac{\partial \varepsilon^2}{\partial x_i}f^{'}(a_i)$$

En posant

$$y_i = -\frac{\partial \varepsilon^2}{\partial a_i}$$

on a:

$$y_i = -\frac{\partial \varepsilon^2}{\partial x_i}f^{'}(a_i)$$

où :

- a_i est l'activité du neurone i,
- x_j la sortie du neurone j,
- y_i le gradient attaché au neurone i.

Deux cas se présentent pour le calcul de $\frac{\partial \varepsilon^2}{\partial x_i}$:

- le neurone appartient à la couche de sortie, alors :

$$y_i = 2f^{'}(a_i)(d_i - x_i)$$

- le neurone appartient à une couche cachée, alors :

$$y_i = f^{'}(a_i)\sum_{k=0}^{n} w_{ki}y_k$$

où

- $f^{'}(a_i)$ est la dérivée de la sigmoïde au point de fonctionnement a_i.
- d_i est la sortie désirée présentée au neurone i.
- k est le nombre de connexions en sortie du neurone i.

On remarquera pour les couches cachées l'expression :

$$\sum_{k=0}^{n} w_{ki}y_k$$

qui est une somme pondérée des gradients des cellules aval (vers la sortie). C'est cette **dépendance récursive** qui montre que seul les gradients des couches de sortie doivent être connus. Ceci justifie le nom de **"rétropropagation du gradient"** donné à l'algorithme.

Pour un ensemble de h exemples entrée-sortie, la fonction coût devient :

$$C(w) = M\{C_h(w)\} = M\{\sum_{q \in U_s} (d_{qh} - x_{qh}(w))^2\}$$

où

- U_s est l'ensemble des cellules de sortie,
- x_{qh} l'état de la cellule d'indice q pour l'exemple h,
- d_{qh} l'état désiré de la cellule d'indice q pour l'exemple h,
- $M[.]$ un opérateur de moyennage habituel.

En résumé, l'algorithme est le suivant

- **Sélection du couple entrée-sortie dans la base d'exemples :**
  - Présentation du vecteur d'entrée sur les cellules d'entrée et calcul de l'état du réseau (les x_i) par propagation, à l'aide de la formule :

    $$x_i = f(a_i) \quad a_i = \sum_{j=0}^{n} w_{ij} x_j$$

  - Présentation du vecteur de sortie désirée sur les cellules de sortie et calcul des gradients relatifs aux a_i par rétropropagation, à l'aide des formules :
    - $y_i = 2f'(a_i)(d_i - x_i)$ pour les cellules de sortie,
    - $y_i = f'(a_i) \sum_{k=0}^{n} w_{ki} y_k$ pour les cellules cachées.
- **Quand tous les exemples ont été présentés :**
  - Si $C(w)$ diminue, alors application d'une itération de la procédure du gradient :

    $$w_{ij} \leftarrow w_{ij} + \lambda y_i x_j$$

  - Si $C(w)$ augmente, alors légère modification aléatoire de l'ensemble des poids :

    $$w_{ij} \leftarrow w_{ij} + perturbation\ alatoire$$
- **Retourner au début jusqu'à ce que $C(w)$ soit suffisamment faible.**

La figure 2.4 schématise succeintement l'algorithme. La présence obligatoire de la base d'apprentissage fait que la rétropropagation appartient à la classe des apprentissages supervisés.

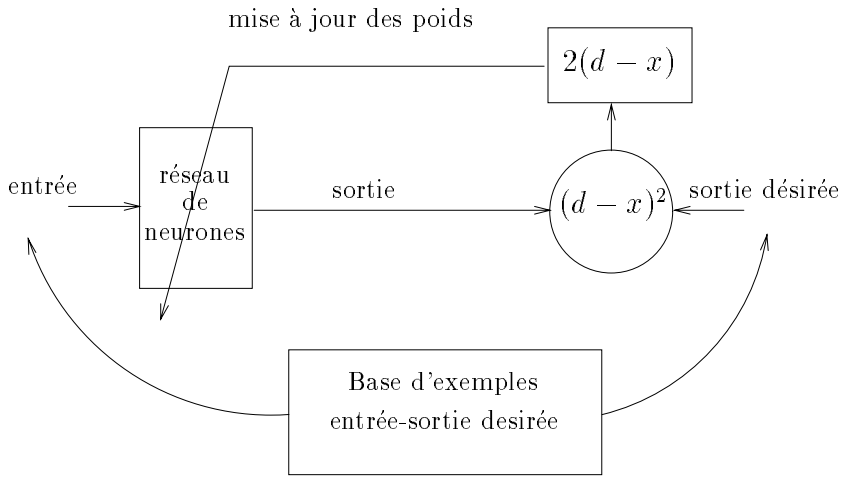

Fig. 2.4: *Structure de la rétropropagation*

Remarques

L'algorithme obtenu est donc simple, mais mérite quelques éclaircissements:

- λ est le pas d'apprentissage. L'expérience montre que l'algorithme converge difficilement si $\lambda > 1$.
- L'introduction d'une perturbation aléatoire des poids, lorsque le coût augmente, permet de sortir des minimums locaux en "secouant légèrement" le réseau.
- Afin d'améliorer la convergence, on peut introduire un terme de moment (ou viscosité) dans la procédure du gradient. A l'itération k, la modification du poids w devient:

$$w_{ij}^k \leftarrow w_{ij}^{k-1} + \Delta w^k$$

$$\text{où} \quad \Delta w^k = 2\lambda(1-\eta)y_i x_j + \eta \Delta w^{k-1} \quad \text{avec} \quad 0,8 < \eta < 0,9$$

  Ce paramètre joue un rôle de lissage (ou de filtrage) des oscillations au début de l'apprentissage.

- T. Yamada et T. Yabuta proposent, pour des problèmes de commande dans [*Yama 93*], d'adapter pendant l'apprentissage la pente de la fonction sigmoïde afin d'en chercher la valeur optimale.

2.5 Inconvénients de la rétropropagation

Liés à la nature de l'algorithme

C'est un algorithme de minimisation d'une fonction par la procédure du gradient dans un espace de grande dimension. Les inconvénients sont donc les suivants :

- présence de minimums locaux
- présence de plateaux et de ravins
- fonction de coût non convexe

De plus, la qualité de la convergence est extrêmement liée à l'ensemble d'apprentissage puisque toutes les modifications de poids dépendent de l'erreur de sortie. Les paramètres λ et η permettent respectivement de modifier la vitesse de convergence et de lisser les oscillations. Malheureusement, une adaptation en cours d'apprentissage de ces paramètres est très délicate. On préfère en général stopper l'apprentissage, modifier soi-même ces paramètres, puis reprendre l'apprentissage. Enfin L. Holmström et P. Koistinen proposent dans [*Homs 92*] de bruiter les exemples pour améliorer la convergence afin de rendre l'algorithme plus robuste aux minimums locaux.

Liés à l'utilisation de l'algorithme

Le but de la rétropropagation est de minimiser l'erreur quadratique afin d'obtenir du réseau une généralisation de la fonction décrite par la base d'exemples. Or, du fait de l'utilisation de la technique du gradient, la convergence est asymptotique. On atteint donc le minimum global au bout d'un nombre infini d'itérations.
C'est là tout le problème de la rétropropagation :

- si on veut minimiser le plus possible l'erreur sur la base d'exemples, il faut a priori poursuivre longtemps l'apprentissage.
- si l'on veut bien généraliser la base d'exemples, il ne faut pas poursuivre longtemps l'apprentissage.

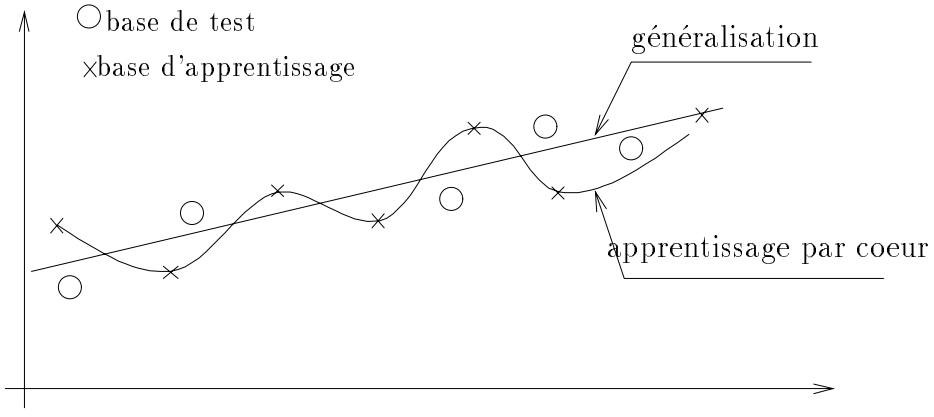

Fig. 2.5: *L'apprentissage par coeur*

La minimisation et la généralisation constituent donc un dilemme difficile à résoudre. On peut expliquer ce phénomène en supposant que l'apprentissage est équivalent à un calcul des

coefficients d'une fonction d'interpolation.
Prenons l'exemple d'un problème d'identification (figure 2.5). Le réseau doit par apprentissage identifier une fonction décrite par un ensemble d'exemples. On sépare la base en deux parties, l'une qui sert d'apprentissage (les croix), l'autre qui sert de test pendant l'apprentissage (les cercles). Les erreurs sur la base d'exemples et sur la base de test évoluent alors, pendant l'apprentissage, comme le montre la figure 2.6.
On constate que l'erreur sur la base de test décroit en même temps que l'erreur d'appren-

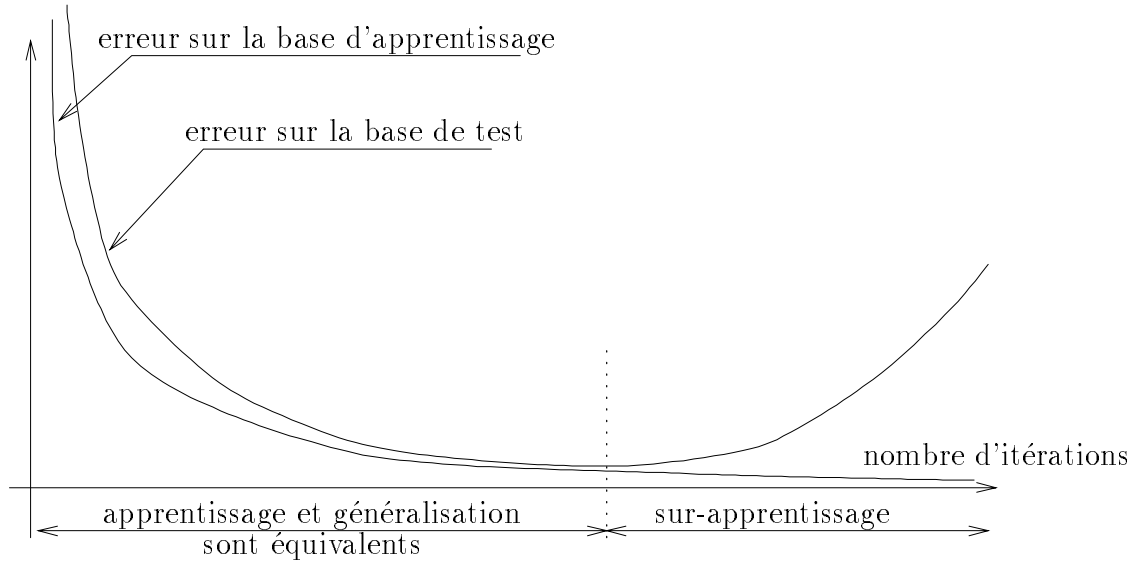

Fig. 2.6: *Le problème du sur-apprentissage*

tissage pendant un nombre limité d'itérations. Il y a donc généralisation puisque le réseau commet une faible erreur sur des situations qui ne lui ont pas servi d'exemples. Au delà de cette limite, l'erreur de test augmente tandis que l'erreur d'apprentissage diminue encore. C'est la phase dite de sur-apprentissage ou d'apprentissage par coeur, qui efface peu à peu la généralisation. Dans le premier cas, le réseau interpole la base d'exemples comme sur la figure 2.5. Il trouve une courbe (proche d'une droite) qui passe en moyenne près de tous les points. Dans le cas d'un apprentissage par coeur, la courbe passe exactement par les points de la base d'apprentissage, mais loin des points de la base de test. Il n'y a pas de généralisation.
Le critère d'arrêt de la boucle d'apprentissage est donc un point important de l'algorithme. Il convient en général de disposer d'une base de test pour évaluer en permanence pendant l'apprentissage le taux de généralisation. Malheureusement, en pratique cela ralentit beaucoup l'algorithme. Une technique plus rudimentaire consiste à sauvegarder périodiquement le réseau pendant l'apprentissage, et de tester ensuite la capacité de généralisation de chaque réseau obtenu. C'est la méthode que nous avons choisie.

Chapitre 3

Quelques Utilisations des RNF pour la commande

L'objet de ce chapitre est de montrer l'intérêt des réseaux de neurones pour la commande de robots à travers quelques applications originales et très différentes sélectionnées dans la littérature. Cette liste n'est bien sûr pas exhaustive.

3.1 Commande cartésienne d'un bras manipulateur (Simulation)

G. Josin, D. Charney, D. White proposent dans [*Josi 88*] d'utiliser un réseau de neurones pour corriger les erreurs entre les positions articulaires et la position réelle du robot. Le système neuronal apprend donc à corriger les imperfections difficilement modélisables du bras (causées par exemple par les effets mécaniques).

Le système à commander

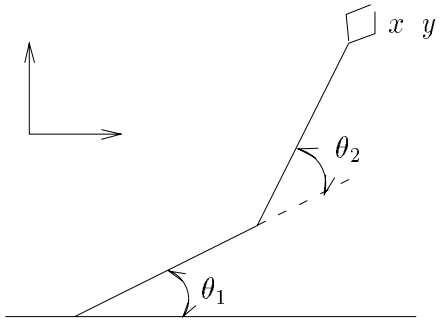

Fig. 3.1:

Le manipulateur considéré est plan à 2 degrés de liberté. Deux types d'imperfections peuvent être simulées sur la position du terminal. L'une est une erreur constante, l'autre une erreur qui dépend de la configuration du système.

Protocole d'apprentissage

Vecteur d'entrée : $\begin{pmatrix} x \\ y \\ \theta_1' \\ \theta_2' \end{pmatrix}$

Vecteur de sortie : $\begin{pmatrix} \Delta\theta_1 & = & \theta_1 - \theta_1' \\ \Delta\theta_2 & = & \theta_2 - \theta_2' \end{pmatrix}$

où

- x y sont les coordonnées cartésiennes du terminal,
- θ_1 θ_2 sont les positions articulaires réelles du bras,
- θ_1' θ_2' sont les positions articulaires désirées (par le correcteur).

Structure du réseau

Couche d'entrée : 4 cellules.
Couche de sortie : 2 cellules.
Une couche cachée de 24 cellules.

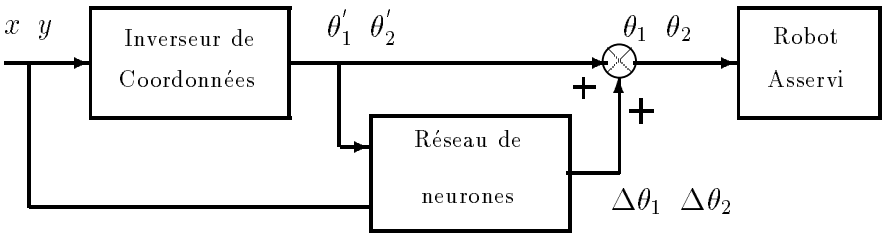

Fig. 3.2: *Structure de la commande*

Conclusion

L'auteur montre qu'un réseau de neurones est capable d'anticiper et de corriger les imperfections du système commandé (s'il a appris à le faire pendant l'apprentissage). La phase d'apprentissage est très rapide. Par contre, la taille de la couche cachée (144 poids) nous paraît trop importante compte tenu du problème.

3.2 Commande cartésienne d'un bras manipulateur (Expérimentation)

C'est l'aspect expérimental qui est très intéressant dans cet application. En effet, les auteurs ([*Soba 88*]) remplacent (après une phase d'apprentissage) le correcteur classique par un réseau de neurones.

Système contrôlé

Il s'agit d'un bras industriel INTELLEDEX 605T à 6 degrés de libertés (dont 4 sont bloqués pour l'expérience) asservi en position. La communication entre le robot et le neurocontrôleur se fait via une liaison série RS 232C.

Protocole d'apprentissage

Vecteur d'entrée : $\begin{pmatrix} \theta_1 \\ \theta_2 \\ \rho_t \\ \theta_t \end{pmatrix}$

vecteur de sortie $\begin{pmatrix} \Delta\theta_1 & = & \theta_1(t+1) - \theta_1(t) \\ \Delta\theta_2 & = & \theta_2(t+1) - \theta_2(t) \end{pmatrix}$

où

- θ_1 θ_2 sont les positions articulaires,
- ρ_t θ_t sont les coordonnées polaires désirées du terminal,
- $\Delta\theta_1$ $\Delta\theta_2$ sont les incréments angulaires des articulations.

Deux types de tâches sont apprises :

- Atteindre toujours le même point quelque soit la position initiale (26 exemples).
- Atteindre n'importe quel point quelque soit la position initiale (64 exemples).

Structure du réseau

2 couches cachées de 4 et 6 cellules.

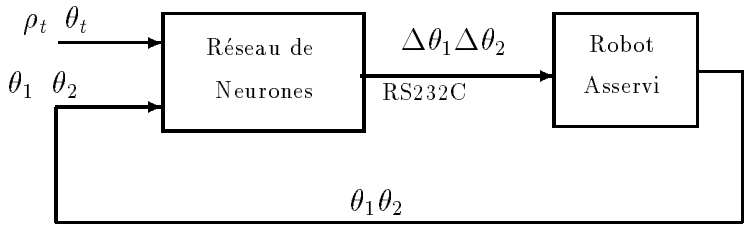

Fig. 3.3: *structure de la commande neuronale de l'INTELLEDEX 605T*

Conclusion

Les erreurs statiques relatives, de positionnement polaire du terminal, sont faibles :
de l'ordre de 1% pour la première tâche et de 2% pour la seconde.
L'auteur souligne la rapidité de réponse du contrôleur neuronal et sa robustesse face à deux types de perturbations :

- un déplacement de la cible avant qu'elle ne soit atteinte (poursuite),
- un échelon aléatoire sur la position du terminal.

On peut remplacer un contrôleur classique par un réseau de neurones dans un système réel. L'ensemble est robuste aux perturbations. Enfin, la méthode ne nécessite pas de connaître un modèle mathématique complexe du manipulateur, mais de procéder à une séance d'apprentissage pour calculer le réseau.

3.3 Contrôle dynamique d'un pendule inverse avec un *"maître humain"*

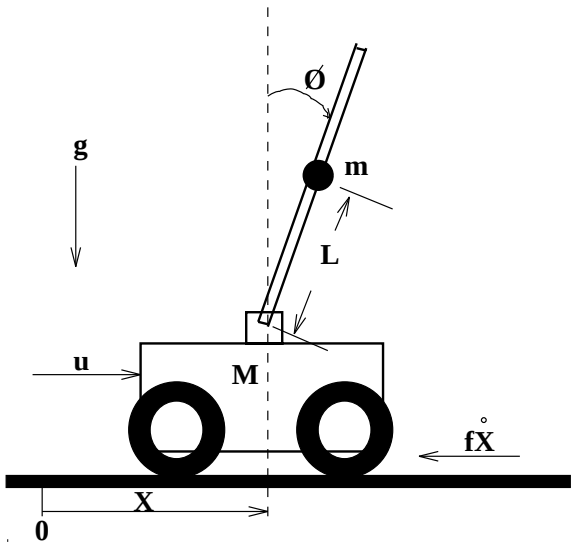

Fig. 3.4: *Cart-Pole System*

Il s'agit ici d'un problème d'apprentissage de comportement réflexe. A. Guez, J. Selinsky dans [*Guez 88*] font superviser l'apprentissage par un opérateur humain. Le réseau neuronal apprend ainsi la stratégie de commande de l'homme.

Système contrôlé

Le but est d'obtenir l'équilibre stable d'un pendule inverse, fixé par une liaison pivot sur un chariot (figure 3.4) dont la position est contrôlée par l'application d'une force sur ce chariot. Le comportement dynamique de l'ensemble est simulé et des perturbations sont introduites en modifiant la position de la masse m.

Protocole d'apprentissage et architecture de commande

L'architecture de commande est décrite par la figure 3.5. On remarque sur la figure la mise en parallèle du réseau neuronal et de l'opérateur humain, ce qui induit un apprentissage En-Ligne. De plus, une compensation du temps de réaction de l'opérateur (0,2 s) peut être prise en compte.

Le vecteur d'entrée est le vecteur d'état : $\begin{pmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{pmatrix}$

et la sortie : la force appliquée u.
La convergence du réseau est obtenue en 4000 itérations (100 s) sans compensation et en moins de 10 secondes avec. L'architecture ci-dessus permet donc d'apprendre la stratégie de l'opérateur, c'est à dire les réflexes qui permettent de garder l'équilibre d'un système en fonction de son état (à condition que l'opérateur ne se trompe pas).

Structure du réseau

2 couches cachées (de taille inconnue).

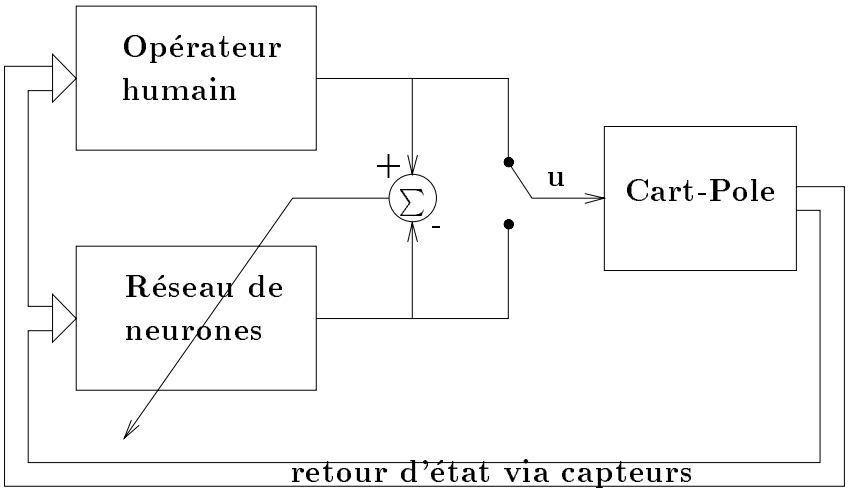

Fig. 3.5: *Principe de l'apprentissage supervisé par l'opérateur*

Conclusion

Ce travail montre qu'un réseau de neurones peut apprendre en-ligne le contrôle d'un modèle de référence. Seul, le vecteur d'état est nécessaire. L'ensemble est très robuste aux changements de masse et aux frottements. Inconvénient principal est la présence du superviseur humain.

3.4 Contrôle dynamique d'un pendule inverse (simulation)

Le problème est le même que précédemment[1] (figure 3.4), avec aussi un contrôle en-ligne. Mais F. C. Chen et Y. H. Pao ([*Chen 89*] proposent une solution sans supervision humaine.

Protocole d'apprentissage

Vecteur d'entrée : $\begin{pmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{pmatrix}$ et le vecteur de sortie : u

Etat désiré du système : $x_d = \begin{pmatrix} x_d \\ \phi_d \end{pmatrix}$

Structure de commande

L'idée (très voisine de la nôtre) est de souhaiter un état désiré du système plutôt qu'un état désiré du réseau. Pour cela, l'auteur déduit une erreur équivalente en sortie de réseau par dérivation numérique de cet état désiré (linéarisation du modèle dynamique)

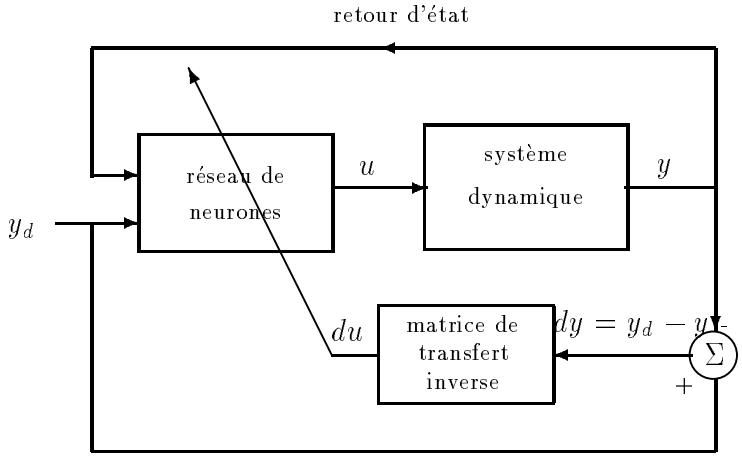

Fig. 3.6: *Principe de l'apprentissage par linéarisation*

Conclusion

La convergence du réseau est assurée en-ligne par cette méthode et sans supervision. L'inconvénient réside dans le calcul numérique du modèle inverse linéarisé.

[1] B. Widrow dans [*Widr 92*] propose aussi une solution. Il utilise un retour visuel du pendule comme entrée du réseau

3.5 Contrôle d'un Robot mobile (Expérimentation)

Le travail présenté ici ([*Daxw 94*]) utilise le concept de "clonage". Un réseau de neurones apprend à garer en marche arrière un robot mobile à roues. La base d'exemples est constituée de trajectoires enregistrées pendant quelques manoeuvres télécommandées par un opérateur.

Système contrôlé

Il s'agit d'un robot mobile MACROBE doté de deux roues motrices fixes et d'une roue directrice d'orientation θ. La position du robot est définie par ses coordonnées (x, y) et son orientation par l'angle ϕ. Le réseau calcule ϕ à partir de l'état courant (x, y, θ) du robot qui se déplace à vitesse constante.

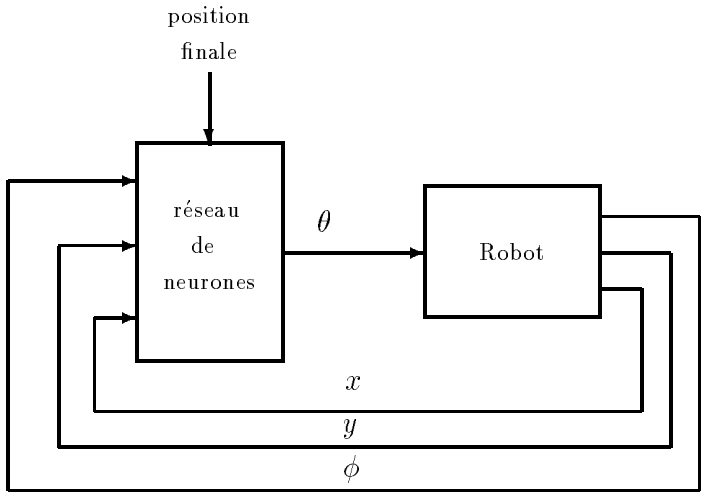

Fig. 3.7: *Principe de commande de la position et de l'orientation du robot mobile*

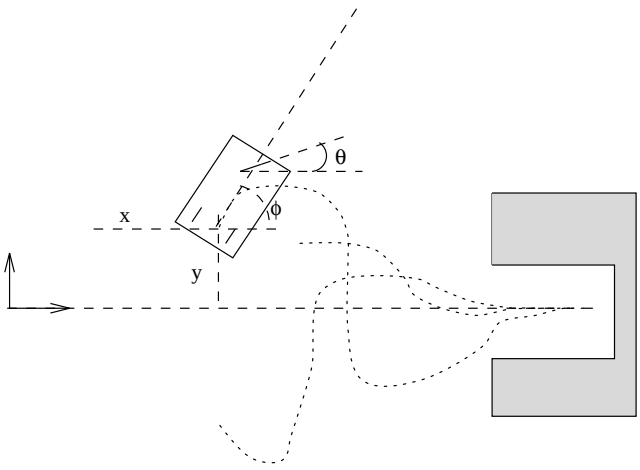

Fig. 3.8: *Exemple de trajectoires de la base d'apprentissage*

Conclusion

L'auteur montre les capacités de généralisation du réseau pour des manoeuvres d'accostage d'un robot mobile expérimental. L'algorithme généralise les exemples fournis par un opérateur. Il apprend donc sa stratégie.

3.6 Contrôle de la locomotion d'un bipède (simulation)

D.M.A. Lee et W.H. ElMaraghy utilise dans cette étude ([*Lee 92*]) deux réseaux de neurones pour contrôler l'allure dynamique d'un bipède. Le but de l'apprentissage est de garder l'équilibre d'un bipède, pendant son déplacement, plus longtemps qu'un contrôleur linéaire traditionnel (2 à 3 secondes).

Système contrôlé

Le bipède simulé est plan, simplifié à 3 corps liés par 2 articulations. Les contacts pieds-sol sont ponctuels et les centres de masses sont au centre de chaque corps (figure 3.9).

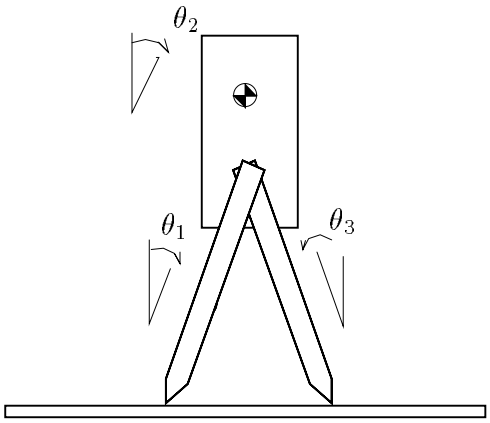

Fig. 3.9: *modèle simplifié d'un bipède*

Protocole d'apprentissage

Le vecteur d'entrée est le vecteur d'état : $(\theta_1,\ \dot{\theta_1},\ \theta_2,\ \dot{\theta_2},\ \theta_3,\ \dot{\theta_3})^t$
Le vecteur de sortie est constitué des couples articulaires : $(\tau_1,\ \tau_2)^t$

Deux réseaux sont utilisés alternativement en fonction de la patte qui est au sol. Ils observent tout l'état du bipède et calculent les couples articulaires des pattes. Chaque réseau apprend la sortie désirée de trois contrôleurs P.I.D. (correspondants à trois allures différentes). Il y a 2528 exemples par contrôleur, soit 7584 exemples au total. La durée de l'apprentissage est très longue (50 heures sur une station de travail SPARC).

Structure des réseaux

Ils sont identiques et de petite taille (une couche cachée de 20 neurones).

Conclusion

L'auteur obtient après apprentissage, deux réseaux commandant chacun les deux pattes. L'un est utilisé lorsque la patte gauche est au sol, l'autre lorsque c'est la patte droite. En commutant ainsi sur l'un puis l'autre, la méthode permet de multiplier par 13 la durée de l'équilibre pendant le déplacement, par rapport aux contrôleurs P.I.D. Le temps d'apprentissage est très long à cause du très grand nombre d'exemples.

3.7 Apprentissage de Contrôle d'Admittance (expérimentation)

Un réseau de neurones est utilisé par V. Gullapalli et al. ([*Gull 92*]) pour une tâche plane d'insertion ("Peg-in-hole" problem). Le réseau à deux couches cachées de 15 neurones (ré-

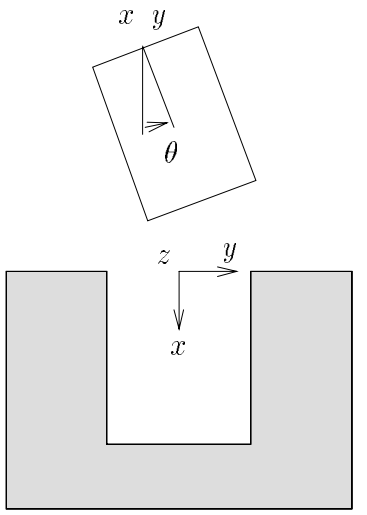

Fig. 3.10: *Tâche d'insertion*

seau : 6-15-15-3). Le jeu pour l'insertion est important (0,8 mm pour un diamètre de 22×50 mm). L'entrée du réseau combine les efforts F_x, F_y et le moment M_z, mesurés par le capteur, ainsi que la position x, y et l'orientation θ du terminal.
Ses sorties sont les vitesses V_x, V_y, ω_z.

L'apprentissage utilise la technique de renforcement. C'est à dire que des heuristiques permettent de qualifier chaque action par rapport à chaque situation évaluée sur le nouvel état du système (forces et positions). L'objectif est d'obtenir du réseau un comportement qui

soit le plus récompensant possible, maximisant un signal de renforcement[2] (en général un critère qualitatif). Les poids sont modifier par rétropropagation d'une erreur entre la sortie du réseau et une sortie désirée liée au signal de renforcement.
L'apprentissage est fait en-ligne à partir de positions et d'orientations aléatoires de la tige. L'auteur montre que le temps d'insertion diminue après chaque apprentissage, c'est à dire qu'une stratégie d'insertion est déterminée par le réseau en fonction des récompenses et des pénalités.

Conclusion

Une tâche d'insertion est réalisée directement par un réseau de neurones. Celui-ci a su identifier une stratégie sur les vitesses de position et d'orientation du terminal. Aucun modèle du problème n'a été utilisé. Seules les mesures d'efforts au contact et de posture du terminal sont combinées pour évaluer s'il y a récompense ou pénalité.

3.8 Conclusion du chapitre

Les avantages

- Un réseau de neurones est capable de généraliser un comportement réflexe dont la loi de commande n'est pas formalisable.
- Le temps de propagation est très rapide et permet une solution en temps réel pour la commande.
- Les méthodes ne demandent pas une connaissance très précise du modèle du processus.
- La taille du réseau n'est jamais très grande, mais sa structure diffère beaucoup (sans cohérence) d'une application à l'autre.
- Les paramètres de la tâche (position désirée, ..) peuvent êtres présents en entrée.
- La commande est robuste. Elle l'est encore plus pour des structures adaptatives en-ligne.
- La mise en œuvre pratique ne pose apparemment pas de problème quant aux bruits des signaux issus des capteurs.

Les inconvénients

- Nécessité pour construire la base d'exemples, d'une connaissance a priori de la commande pour obtenir la sortie désirée. Ceci entraine souvent une inversion de modèle (géométrique, cinématique ou dynamique) et donc l'utilisation de techniques académiques.

[2] C. Touzet applique cette technique dans [*Touz 93*] pour commander un mini robot mobile, et dans [*Touz 92*] pour simuler l'apprentissage d'un déplacement d'un insecte hexapode.

- Présence nécessaire d'une supervision qui dicte le bon comportement si l'on veut éviter le problème précédent. Ce superviseur peut être un contrôleur classique, un opérateur humain ou même un contrôleur flou [*Kong 92*] [*Ouss 94*].
- Manque d'outils théorique d'analyse des résultats (stabilité, ...).

Ces inconvénients sont très pénalisants puisque beaucoup de limitations dans la commande des processus complexes proviennent de l'inversion des modèles. En effet, dans le cas le plus courant (système redondant), on se heurte au choix d'une solution parmi plusieurs. Ce choix, effectué en minimisant un critère (énergétique, manœuvrabilité, ...), est par conséquent couteux en temps de calcul et pénalise une commande en temps réel. Par contre, une tâche robotique est plus souvent exprimable sous forme de critère à minimiser que sous forme d'une loi générale.

De cette analyse se dégage une structure générale d'apprentissage et d'adaptation dans laquelle le réseau est situé à la place du contrôleur classique. Il est calculé hors-ligne par la minimisation d'un critère exprimé en fonction des variables de commande et peut être adapté en-ligne de la même façon. C'est la méthode que nous proposons et que nous appellerons apprentissage indirect. Elle est présentée au chapitre suivant.

Chapitre 4

l'Apprentissage Indirect

Le chapitre précédent a présenté l'intérêt majeur des Réseaux de Neurones pour la commande des robots :

- capacité d'identification et de généralisation grâce à l'algorithme de rétropropagation,
- possibilité de calcul en temps réel.

Le but principal de la commande de ces systèmes est de maîtriser l'exécution d'une ou de plusieurs tâches particulières dans l'espace opérationnel, comme par exemple :

- atteindre un point et une orientation,
- suivre une trajectoire,
- exercer un effort,
- maintenir un équilibre (statique ou dynamique).

Nous montrons au début de ce chapitre que l'obstacle principal de l'apprentissage direct par rétropropagation du gradient de l'erreur, est le calcul de la sortie désirée du réseau à partir d'un état désiré du système dans l'espace opérationnel.

Nous proposons ensuite une solution qui consiste à rétropropager le gradient d'un critère qui exprime la tâche dans cet espace. Nous présentons enfin les intérêts de cette technique, appelée **rétropropagation indirecte** par opposition à la rétropropagation directe. Nous montrons qu'elle permet un apprentissage en-ligne similaire à une commande adaptatif.

4.1 Inconvénients de la Rétropropagation Directe

Un schéma habituel de commande utilise une observation sur l'état courant du système pour calculer la commande (figure 4.1). L'entrée $e$du réseau est une combinaison entre l'état y du système et un état désiré y_d (sous forme d'erreur par exemple). La sortie est un vecteur de

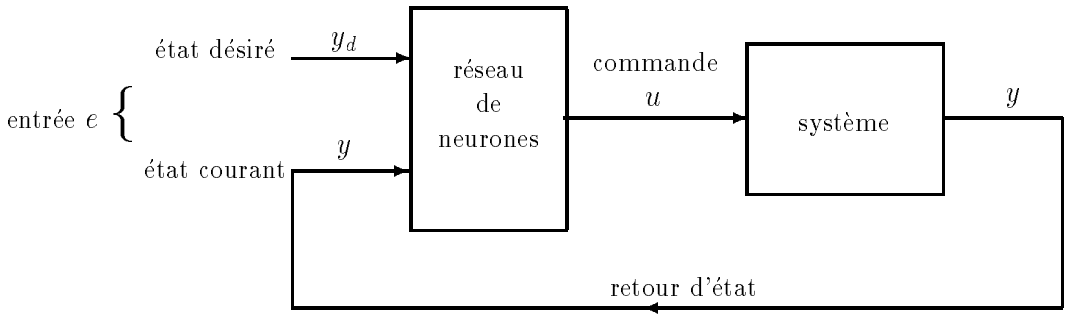

Fig. 4.1: *Structure de commande par retour d'état*

commande u.

Pour obtenir un apprentissage de la commande par rétropropagation directe, il faut constituer une base d'exemples de vecteurs d'entrée et de vecteurs de sortie désirés. C'est à dire qu'il faut calculer la commande désirée u_d. Pour obtenir cet état désiré en sortie de réseau, à partir de l'état désiré en sortie du système, il existe plusieurs solutions :

- Utiliser, comme W. Daxwanger ([*Daxw 94*]) ou A. Guez ([*Guez 88*]), un opérateur humain qui dicte la commande. On parle alors de "clonage neuronal".
- Utiliser une technique d'apprentisasge par renforcement basée sur une heuristique de comportement ([*Touz 93*] et [*Touz 92*]).
- Utiliser un contrôleur flou comme modèle de référence ([*Ouss 94*].
- Utiliser un contrôleur conventionnel comme modèle de référence.

C'est la dernière solution qui est généralement utilisée. En effet, la base d'apprentissage est alors fabriquée à partir de méthodes académiques qui ont une assise théorique éprouvée. La figure 4.2 schématise l'apprentissage direct. Elle fait clairement apparaître l'utilisation d'un modèle inverse (géométrique, cinématique, dynamique) du système pour transposer l'état désiré exprimé dans l'espace de la tâche vers l'espace de commande.
Or, l'inversion du modèle d'un système mécanique complexe (robot manipulateur, robot à pattes), particulièrement pour la commande dynamique, pose des difficultés face aux redondances cinématiques. Le calcul du modèle dynamique inverse est alors basé sur l'optimisation d'un critère (énergétique, manœuvrabilité,...) pour choisir une solution parmi une infinité ([*Ouez 90*]). Sa résolution en temps réel est donc très coûteuse.

Face à ces rappels, nous pouvons faire deux remarques :

1. l'inversion du modèle rend impossible l'apprentissage direct en-ligne, donc la commande neuronale adaptative.
2. La rétropropagation du gradient étant un algorithme d'optimisation. On peut donc directement l'appliquer pour optimiser un critère. Dans ce cas, on n'utilise plus de modèle inverse, mais un modèle direct du robot. Utilisé en-ligne, l'apprentissage devient alors simple et permet une commande neuronale adaptative.

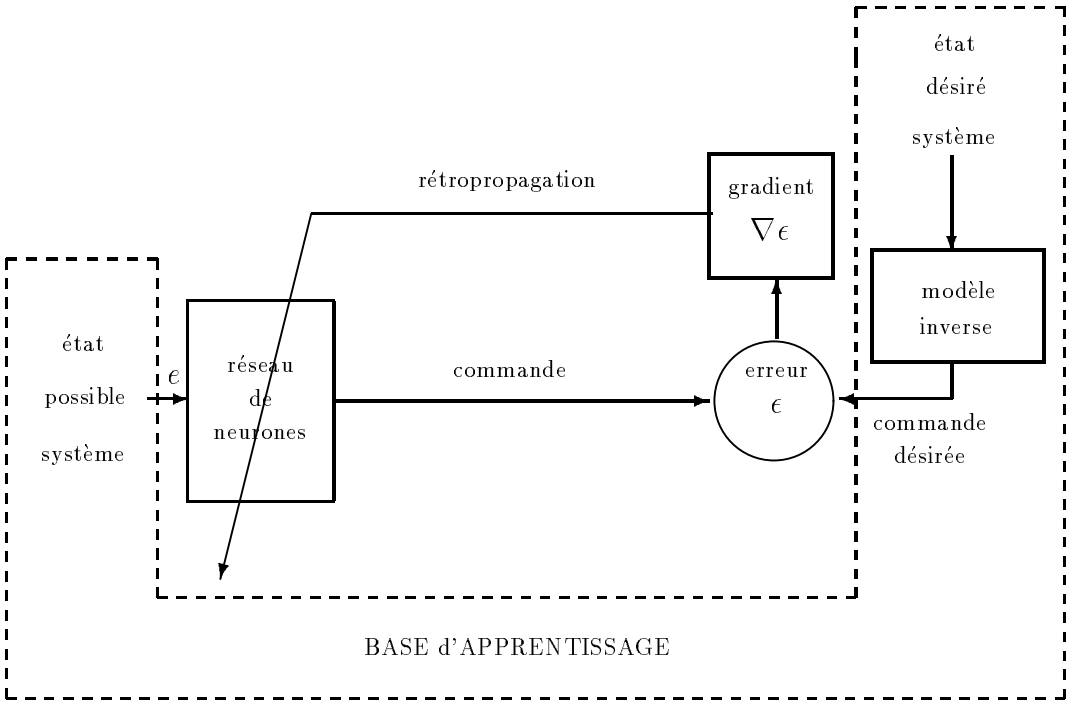

Fig. 4.2: *Structure de l'apprentissage direct*

4.2 La rétropropagation indirecte

La rétropropagation directe minimise un coût, qui est une erreur quadratique calculée entre la sortie du réseau et une sortie désirée, c'est à dire entre son état actuel et un état désiré. La seule condition que doit respecter ce coût est de pouvoir calculer analytiquement son gradient par rapport aux sorties ([*Le Cun 87*]). Nous proposons donc de remplacer cette erreur par un critère plus général qui exprime analytiquement le but à atteindre, c'est à dire un état désiré du processus et non plus un état désiré du réseau. Il suffit ensuite de calculer analytiquement le gradient de ce critère par rapport aux sorties qui sont les variables de commande.
Il y a deux principes d'utilisation :

- l'apprentissage indirect Hors-Ligne, qui utilise un modèle de comportement du système,
- l'apprentissage indirect En-Ligne, qui utilise le systéme réel (ou un modèle de comportement dans le cas d'une simulation). On se trouve alors dans un cadre de commande adaptative.

4.2.1 Principe d'utilisation hors-ligne

La structure d'apprentissage hors-ligne peut être schématisée par la figure 4.3 Seule une modélisation directe du robot est utilisée pour calculer le critère et le gradient[1]. De plus,

[1]D. A. Derradji et N. Mort parlent dans [*Derr 94*], de rétropropagation à travers le système

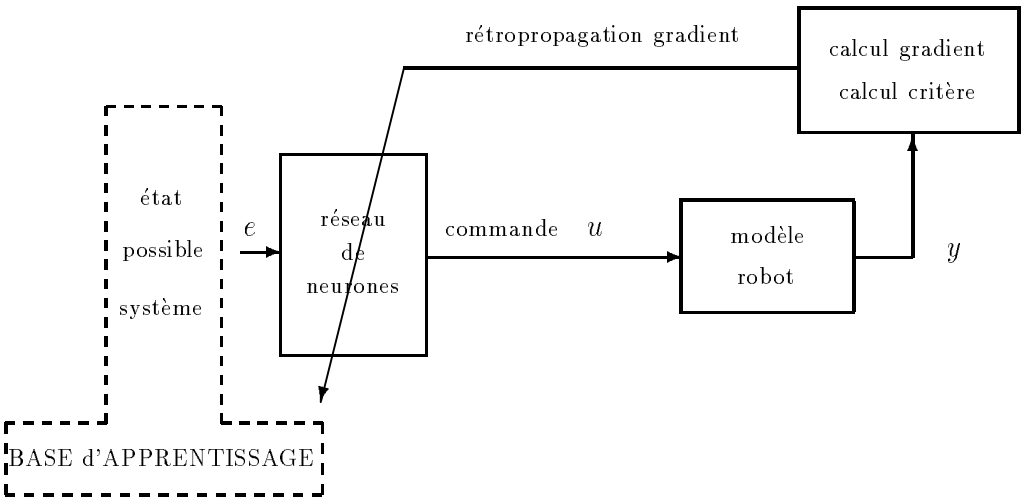

Fig. 4.3: *Structure de l'apprentissage indirect hors-ligne*

la base d'apprentissage est fortement diminuée, puisqu'elle n'est constituée que du vecteur d'entrée. On gagne donc en espace mémoire sur le simulateur.

Remarques

- Le calcul du critère et de son gradient, dans la boucle d'apprentissage, nécessite un modèle de comportement du système à commander. Un modèle dégradé est suffisant puisque nous verron que l'utilisation en-ligne permet d'affiner le calcul du réseau par rapport au comportement réel du système.
- Cette structure est purement algorithmique. On pourrait imaginer que la commande soit directement envoyée au robot. Du nouvel état correspondant, on calculerait le gradient du critère... Ceci serait, d'une part trop long puisqu'il faudrait quelques milliers d'itérations pour quelques centaines ou milliers d'exemples, d'autre part trop dangereux puisque la commande en début d'apprentissage est incohérente.
- Les paramètres d'entrée du réseau peuvent être de toutes sortes d'informations nécessaires. On utilisera souvent les paramètres de la tâche (positions, vitesses, accélérations désirées, ...).
- Le robot est supposé asservi, c'est à dire que nous ne nous occupons pas des asservissements des actionneurs.

4.2.2 Principe d'utilisation en-ligne

Nous avons vu au paragraphe précédent que l'apprentissage du réseau (donc le calcul du contrôleur) est effectué Hors-Ligne. Le calcul du critère nécessite un modèle du système éventuellement dégradé. De plus, en fin d'apprentissage, le critère global (somme du critère sur tous les exemples) tend asymptotiquement vers un minimum. Par conséquent, le réseau obtenu après apprentissage n'est pas exactement le réseau optimal mais en est assez proche.

La structure présentée ici (figure 4.4) est mixte commande/apprentissage. Elle est composée d'une boucle de commande et d'une boucle d'apprentissage. On y distingue plusieurs

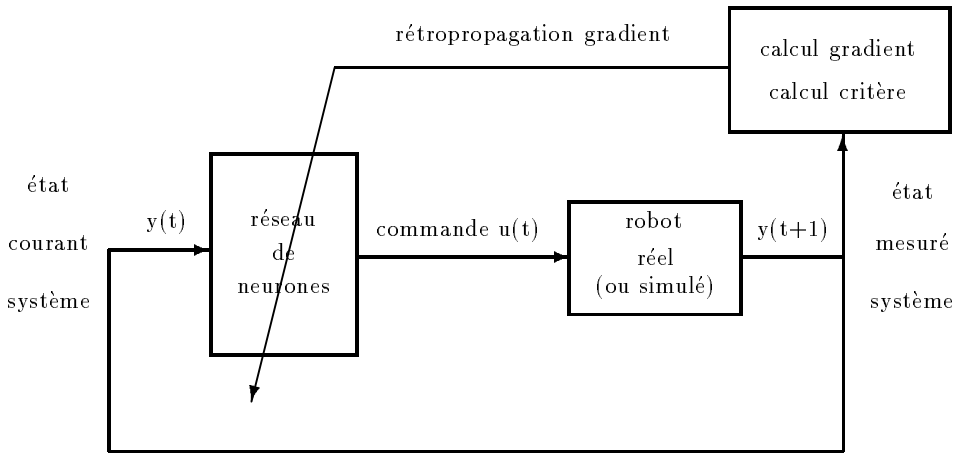

Fig. 4.4: *Structure de l'apprentissage indirect en-ligne*

particularités :

1. A partir d'une observation du système à l'instant t, le réseau calcule une commande qui minimise un critère évalué sur l'état à $t+1$ du robot. Il s'agit donc d'une commande prédictive[2].

2. les boucles d'adaptation et de commande ont la même échelle de temps. C'est à dire qu'il y a une passe d'apprentissage (rétropropagation) pour une passe de commande (propagation). Rien n'empêche cependant de prendre des échelles différentes. On tend alors vers un apprentissage hors-ligne mais avec une structure qui génère elle même ses exemples en fonction de la commande calculée. Cela devient très intéressant dans le cas du contrôle de mouvements périodiques, où la rétropropagation peut avoir lieu à la fin de chaque période. Nous avons appliqué cette technique au chapitre 9 pour la locomotion articulée où la modification du réseau a lieu après chaque enjambée.

Remarques

- C'est une structure d'adaptation qui permet un ajustement optimal des poids face au comportement réel du système. Il serait donc dangereux de mettre un réseau vierge dans cette boucle, puisqu'au début de l'apprentissage, la commande obtenue est aberrante. Elle pourrait être destructrice pour un robot à faible constante de temps. (Nous avons toutefois montré au chapitre 8 que cela est possible dans le cas d'un robot mobile lent).

- Cette structure adaptative, associée aux capacités de généralisation des réseaux de neurones, nous semble tout à fait appropriée pour la commande dite réactive, c'est à dire, lorsqu'on souhaite un comportement réflexe face à une situation particulière : contrôle d'efforts lors de tâches au contact en manipulation, contrôle d'équilibre en locomotion articulée,.... (Le chapitre 9 aborde ce principe de commande réflexe).

[2]Les simulation du chapitre 9 confirment par l'expérience cette hypothèse

4.2.3 Différence entre apprentissage Hors-Ligne et En-Ligne

Du point de vue de la généralisation de la commande d'un système, les deux méthodes sont a priori fondalementalement différentes. Nous pouvons illustrer cette différence en raisonnant dans l'espace des phases du système (figure 4.5) :

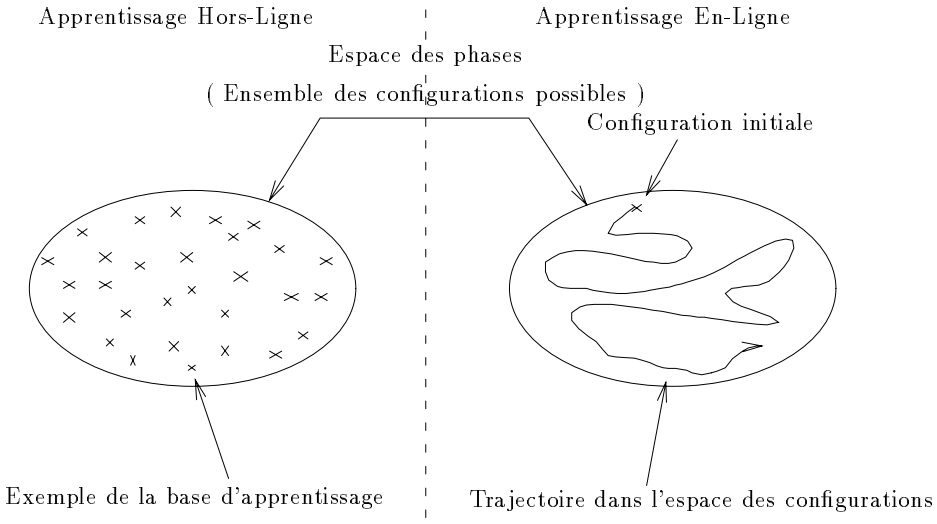

Fig. 4.5: *Illustration de la différence entre Hors-Ligne et En-Ligne dans l'espace des phases du système*

L'apprentissage Hors-ligne utilise une base d'exemples qui discrétise l'espace des phases. Plus cette base le remplit correctement, meilleure est la généralisation. C'est à dire que le RNF aura observé un ensemble représentatif des configurations possibles du système. Il sera donc en mesure de calculer une commande "correcte" pour toutes les configurations.

L'apprentissage En-Ligne fait évoluer le système à partir d'une configuration initiale. Ce dernier évolue dans l'espace des phases en suivant une trajectoire particulière qui dépend des paramètres d'apprentissage et de la configuration initiale. Il ne remplit donc pas entièrement l'espace. Le RNF va donc plus ou moins généraliser suivant cette trajectoire. C'est à dire qu'il contrôlera surtout les configurations situées dans un "tube" autour d'elle. Il y a donc un risque important d'instabilité dans la commande du système s'il se trouve dans une configuration loin de la trajectoire.

C'est pourquoi nous pensons que dans la plus part des cas, il est impératif d'utiliser pour l'apprentissage En-Ligne un réseau de neurones qui a appris Hors-Ligne pour le même objectif. L'expérience du chapitre 8 nous a toutefois montré qu'il est possible d'utiliser la RPI En-Ligne avec un réseau non formé et un robot réel, pourvu que celui ci soit lent, c'est à dire, ayant une faible bande passante (quelques Hertz).

4.2.4 Formulation mathématique

Soient :

- $u(t)$ un vecteur de commande appliqué au système,
- $y(t+1)$ l'état du système correspondant,
- $u_d(t)$ le vecteur de commande désiré,
- $J[y(t+1)]$ le critère à minimiser sur la base d'exemples.

L'algorithme d'apprentissage schématisé par le figure 4.3 minimise $J[y(t+1)]$ par rapport aux poids du réseau. Si on se réfère au paragraphe 2.4, on doit calculer :

$$\frac{\partial \sum_{ex} J[y(t+1)]}{\partial w} \quad \text{au lieu de} \quad \frac{\partial \sum_{ex} [u(t) - u_d(t)]^2}{\partial w}$$

C'est à dire, compte tenu de la rétropropagation

$$\frac{\partial J[y(t+1)]}{\partial u(t)} \quad \text{au lieu de} \quad \frac{\partial [u(t) - u_d(t)]^2}{\partial u(t)}$$

Pour un réseau à n sorties, $u(t)$ est de dimension n, et le gradient $\frac{\partial J[y(t+1)]}{\partial u(t)}$ s'écrit :

$$\frac{\partial J[y(t+1)]}{\partial u(t)} = \nabla J = \begin{cases} \frac{\partial J[y(t+1)]}{\partial u_1(t)} \\ \frac{\partial J[y(t+1)]}{\partial u_2(t)} \\ \vdots \\ \vdots \\ \frac{\partial J[y(t+1)]}{\partial u_n(t)} \end{cases}$$

Nous devons calculer analytiquement ces n termes.

Exemple

Soit un système représenté dans l'espace d'état par les équations discrètes suivantes :

$$\begin{cases} x(t+1) = Ax(t) + Bu(t) \\ y(t) = Cx(t) \end{cases}$$

où les matrices $A \in R^{n \times n}$, $B \in R^{n \times r}$, $C \in R^{m \times n}$,
et les vecteurs $x \in R^{n \times 1}$, $u \in R^{r \times 1}$, $y \in R^{m \times 1}$.

Soit $y_d(t+1)$ l'état désiré du système.
Pour atteindre $y_d(t+1)$, on doit minimiser :

$$J[y(t+1)] = e^2(t+1) = [y(t+1) - y_d(t+1)]^2$$

On doit donc calculer :

$$\frac{\partial e^2(t+1)}{\partial u(t)} = 2e(t+1)\frac{\partial y(t+1)}{\partial u(t)}$$

Or

$$\frac{\partial y(t+1)}{\partial u(t)} = \frac{\partial y(t+1)}{\partial x(t+1)}\frac{\partial x(t+1)}{\partial u(t)}$$

Et

$$\frac{\partial y(t+1)}{\partial x(t+1)} = C \quad \text{et} \quad \frac{\partial y(t+1)}{\partial u(t)} = B$$

On rétropropage finalement

$$\nabla J = 2[y(t+1) - y_d(t+1)]CB,$$

qui est bien un vecteur de dimension n.

Avantages

- La méthode est très simple.
- Il y a très peu de modifications à apporter. En effet, seuls les calculs qui concernent la couche de sortie sont modifiés par l'expression du gradient. Tout ce qui concerne les couches cachées reste inchangé.

Inconvénient

- La détermination d'un critère à la fois pertinent pour la commande et analytiquement dérivable pour l'apprentissage peut être difficile.
- Le gradient n'est plus borné : dans la méthode directe, il l'est par l'erreur maximale en sortie. Cela peut poser des problèmes de convergence en début d'apprentissage (gradient trop fort), ou en fin d'apprentissage (gradient trop faible par rapport à sa valeur maximale).

4.3 Conclusion

Nous proposons une méthode d'apprentissage dont l'objectif est d'éviter l'utilisation d'un modèle inverse du système à commander. La notion de sorties désirées laisse la place à un critère qui exprime analytiquement le but du contrôle.

Nous avons souligné les avantages *a priori* de cette technique pour la commande :

- utilisation d'un modèle direct (même dégradé) du système,
- possibilité dans le critère de prendre en compte des contraintes liées au système,

- possibilité de commande neuronale adaptative,
- possibilité de commande réflexe,

et ses inconvénients :

- détermination d'un critère pertinent,
- calcul du gradient par rapport aux paramètres de commande.

Sur la forme, apprentissage direct et indirect sont très semblables. Mais sur le fond, une différence fondamentale existe à notre avis. En effet, la première méthode utilise un ensemble de sorties désirées pour apprendre au réseau à resoudre un problème. Il est donc contraint de suivre *le chemin du maître* pour arriver à la solution.

L'idée que nous proposons ne contraint pas le réseau. On lui exprime le but à atteindre. L'apprentissage indirect lui permet de trouver *un chemin particulier* pour aller au but.

Nous présentons au paragraphe suivant un "cas d'école" qui permet de comprendre la rétropropagation indirecte pour en montrer l'efficacité et en évaluer certaines limites.

4.4 Problème Géométrique avec Contraintes

L'objectif de cette étude est d'aborder la rétropropagation indirecte Hors-Ligne et En-Ligne. Nous proposons pour cela un problème "test" sans réalité physique, mais qui prend en compte des concepts robotiques : positionnement cartésien, butées, réflexes, ...

Nous ne cherchons pas à atteindre les meilleures performances, mais à comprendre comment on pourra utiliser la rétropropagation d'un critère dans un cadre plus réaliste.

4.4.1 Description du problème

Nous nous proposons de commander la position cartésienne d'un système plan à 3 rotations, supposé asservi en position (voir fig 4.6). Sur cette figure :

- m_i, l_i sont respectivement la masse et la longueur du segment i (rayon du cercle pour l_4),
- (X_g, Y_g) sont les coordonnées du point G, centre de masse du système,
- (l_x, l_y) sont les coordonnées du point C, position désirée (ou cible à atteindre),
- $Q = (q_1\, q_2\, q_3)^T$ vecteur des paramètres articulaire (vecteur de commande).

Nous supposons le système muni de butées articulaires sur les deux premières liaisons. De plus, le sol est considéré comme un obstacle infranchissable.

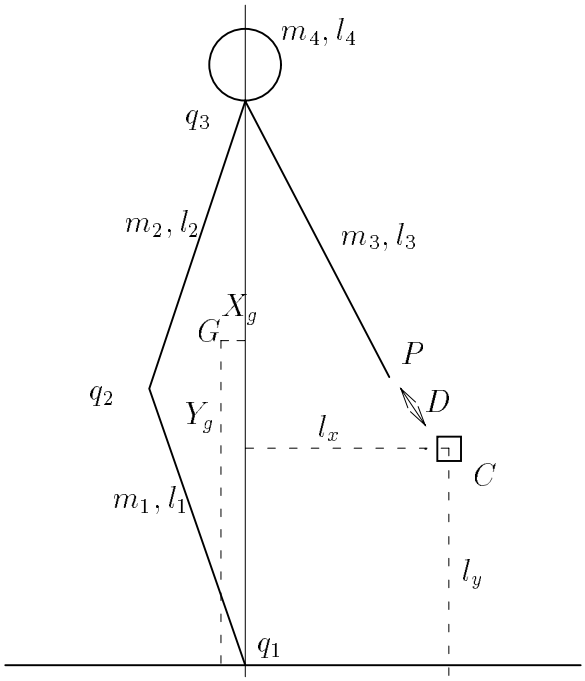

Fig. 4.6: *le problème test*

tâche à résoudre

Nous nous fixons l'objectif suivant :

"S'approcher le plus près possible de la cible C, tout en gardant l'équilibre".

La notion d'équilibre peut être exprimée par $X_g = 0$ et $Y_g > 0$ (projection du centre de masse au point de contact). Le problème posé est donc un problème d'optimisation sous les contraintes non linéaires suivantes :

- contraintes de réflexe (garder l'équilibre),
- contrainte d'environnement (obstacle),
- contraintes d'espace accessible (butées).

4.4.2 Formulation mathématique

Elaboration du critère

La loi de commande doit permettre au point P de se rapprocher de la cible C en respectant les contraintes énoncées au paragraphe précédent. Il faut donc minimiser la distance entre

ces deux points sous les contraintes, c'est à dire :

$$\text{minimiser} \quad D(Q) \quad \text{sous} \quad \begin{cases} X_g(Q) = 0 & \text{(équilibre)} \\ Y_g(Q) > 0 & \text{(debout)} \\ Y_p(Q) > 0 & \text{(obstacle)} \\ q_i \leq q_{i\,max} & \text{(butées)} \end{cases}$$

Ceci revient à minimiser le critère modifié 4.1 dans lequel les contraintes d'inégalité sont incorporées sous forme de pénalités :

$$J(Q) = \alpha_1 D^2(Q) + \alpha_2 X_g^2(Q) + \alpha_3 p_3 Y_p^2(Q) + \alpha_4 p_4 Y_g^2(Q) + \alpha_5 \sum_{i=1}^{3} p_i (q_i - q_{i\,max})^2 \qquad (4.1)$$

$$\text{avec} \begin{cases} X_p(Q) &= -l_1 sin(q_1) - l_2 sin(q_1+q_2) - l_3 sin(q_1+q_2+q_3) \\ Y_p(Q) &= l_1 + cos(q_1) - l_2 cos(q_1+q_2) + l_3 cos(q_1+q_2+q_3) \\ D(Q) &= \sqrt{(X_p - l_x)^2 + (Y_p - l_y)^2} \\ X_g(Q) &= \frac{-A_1 sin(q_1) - A_2 sin(q_1+q_2) - A_3 sin(q_1+q_2+q_3)}{m_1+m_2+m_3+m_4} \\ Y_g(Q) &= \frac{-A_1 cos(q_1) - A_2 cos(q_1+q_2) + A_3 cos(q_1+q_2+q_3)}{m_1+m_2+m_3+m_4} \\ \text{et:} & \\ A_1 &= (\frac{m_1}{2} + m_2 + m_3 + m_4) l_1 \\ A_2 &= (\frac{m_2}{2} + m_3 + m_4) l_2 + m_4 \frac{l_4}{2} \\ A_3 &= m_3 \frac{l_3}{2} \end{cases}$$

où

- $p_i = 0$ si la contrainte correspondante est vérifiée, $p_i = 1$ si elle est forcée.
- α_i coefficients de pondération de la contrainte.

Remarques

- Nous choisissons de minimiser $D^2(Q)$ plutôt que $D(Q)$, puisque le calcul du gradient entraînerait dans ce cas une division par $\sqrt{(X_p - l_x)^2 + (Y_p - l_y)^2}$, donc une instabilité en fin de convergence.
- Les contraintes inégalités sont incorporées dans le critère sous forme de pénalités. Elles sont élevées au carré si elles sont forcées. On peut imaginer des fonctions de pénalisations plus puissantes (mais plus délicates à manipuler) comme la fonction exponentielle. Par exemple, $Y_p^2 > 0$ pour la contrainte, nous pouvons modifier le critère par l'expression : $-p_3 e^{-Y_p}$.
- Il existe une autre technique pour exprimer les contraintes liées aux butées. C'est la méthode par contrainte d'éloignement maximal ([*Coif 86*]):

$$\sum_{i=1}^{3} \frac{(q_i - q_{i\,moy})^2}{(q_{i\,max} - q_{i\,min})^2} \qquad (4.2)$$

4.4.3 Structure d'apprentissage

D'après la figure 4.3, nous proposons de présenter à l'entrée du réseau l'état cartésien désiré et l'état articulaire courant du système. Le RNF doit fournir les paramètres articulaires qui sont solutions du problème. Ceci peut être schématisé par la figure 4.7.

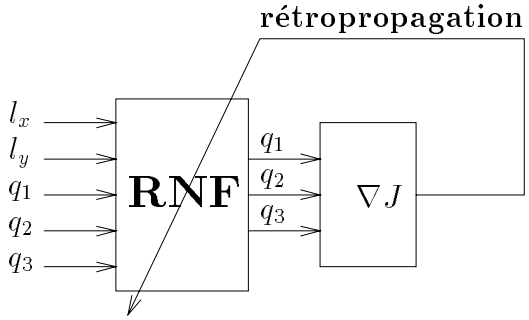

Fig. 4.7: *Structure de l'apprentissage*

Remarques

- L'utilisation de paramètres angulaires est une source de problèmes pour les RNF. En effet, lorsqu'un angle n'est pas borné, ses valeurs extrêmes (par exemple, $-\pi$ et $+\pi$) sont équivalentes. Elles vont correspondre alors aux valeurs maximales non équivalentes de l'entrée du neurone (-1 et +1). Cette discontinuité rend très difficile l'apprentissage. Cependant, on pourrait travailler avec des angles bornés à $\pm\frac{\pi}{2}$, mais ceci limiterait les solutions du problème.

Nous proposons donc de prendre comme paramètres articulaires les sinus et cosinus de chaque angle. Cela permet d'avoir des entrées bornées et normalisées mais présente l'inconvénient de doubler le nombre des sorties et des entrées.

Nouvelle Structure d'apprentissage

Nous changeons simplement les variables angulaires. Le vecteur d'entrée devient :

$$(l_x\, l_y\, cosq_1\, cosq_2\, cosq_3\, sinq_1\, sinq_2\, sinq_3)^t$$

C'est à dire qu'il suffit de calculer les dérivées partielles du critère par rapport aux sinus et cosinus en les considérant comme des variables indépendantes.

4.4.4 Apprentissage Hors ligne

Choix du réseau

La couche d'entrée est constituée de 8 neurones, et la couche de sortie de 6 neurones. Nous prendrons pour la couche cachée 6 ou 10 neurones. Les réseaux à deux couches cachées n'ont en effet pas donné de bons résultats (convergence difficile due à des oscillations).

Construction de la base d'exemples

Nous générons aléatoirement le vecteur d'entrée tel que les angles soient compris entre 0 et 2π et les coordonnées de la cible entre 0 et 0,5. Le choix du nombre d'exemples est lié à la taille du réseau. Si l'on choisit un réseau à une couche cachée de 6 neurones, nous aurons $8 \times 6 + 6 \times 6 = 84$ connexions. Avec de 3 à 10 exemples par connections, cela fait entre 250 et 840 exemples.

4.4.5 Résultats de l'apprentissage

Protocole

Les résultats les plus intéressants ont été obtenu avec les paramètres suivant :

- Réseau : 8 6 6
- Nombres d'exemples : 300
- Lambda = 0,9
- Viscosité = 0

Remarques

- Les sorties du réseau utilisées comme des variables trigonométriques ne sont pas normalisée entre elles. C'est à dire que la relation n'a aucune raison d'être satisfaite, donc le gradient d'être bien calculé. Aussi, nous allons contraindre le RNF à apprendre cette relation. Nous rajoutons donc au critère la contrainte égalité suivante :

$$norme = \sum_{i=1}^{3} (cos^2 q_i + sin^2 q_i - 1)^2 = 0$$

Les courbes suivantes (4.9) résument l'apprentissage pour 10000 itérations, c'est à dire 10000 passes du gradient.

Commentaires sur le critère

Son évolution montre (4.8) très bien la convergence de l'algorithme. La valeur moyenne finale du coût est de 0,022 pour une valeur moyenne initiale de 5,52. L'évolution des contraintes montre qu'elles sont assez bien assimilées. Seul l'objectif (minimiser la distance) est plus difficile à "apprendre" pour le réseau. Ceci s'explique par le fait que l'objectif est une fonction entre l'entrée et la sortie (modèle géométrique), tandis que les contraintes expriment des classes de solutions. On le voit très bien par l'évolution des contraintes obstacle ($Y_p(Q) > 0$) et debout($Y_g(Q) > 0$) qui impose chacune un domaine de solutions. Si nous poursuivons l'apprentissage, nous pouvons encore diminuer la valeur moyenne finale du critère. Toutefois, la décroissance est très lente (on passe en effet de 0,022 à 0,02 en 5000 itérations) puisque le gradient devient très faible (de l'ordre de 10^{-3}). C'est une limite bien connue des algorithmes d'optimisation par le gradient.

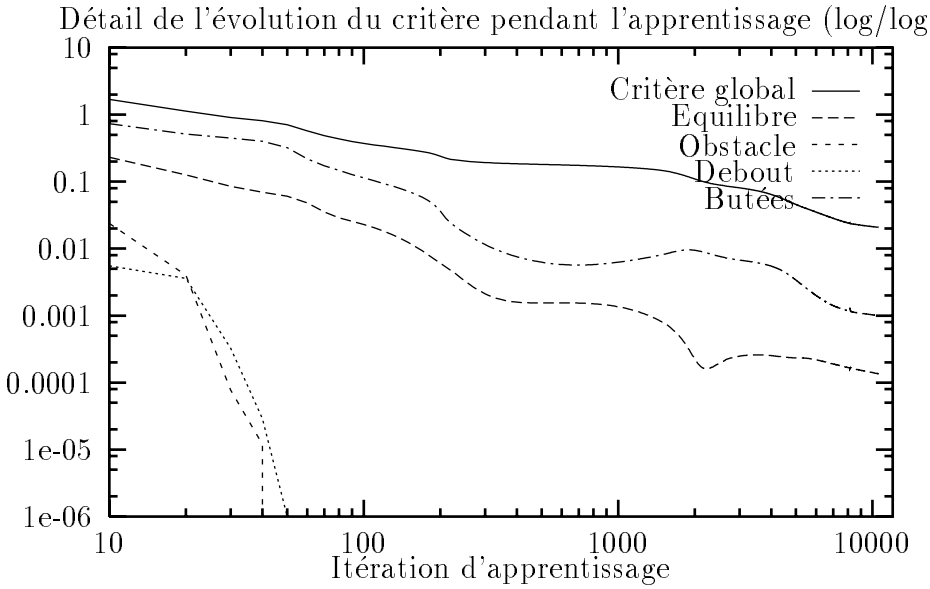

Fig. 4.8: *Apprentissage*

Commentaires sur le gradient

Les six composantes décroissent très vite, puis tendent asymptotiquement vers zéro, mais pas à la même vitesse. En début d'apprentissage, les perturbations sont importantes mais continues.

Remarques

Nous avons obtenu des résultats similaires avec un réseau à une couche cachée de 10 neurones sur une base d'apprentissage de 1000 exemples au bout de 3000 itérations, avec un temps d'apprentissage équivalent.

Réseau obtenu

Les tableaux 4.1 et 4.2 montrent les valeurs des poids obtenus. Le poids $w_{kl}(i,j)$ est le poids reliant le neurone j de la couche k au neurone i de la couche l. Nous apercevons que "globalement" les connexions affectées à l'état du système sont de poids très inférieur à celles de la cible. Ce qui veut dire que le réseau ne tient que peu compte de l'état initial. Ceci est assez normal, puisque l'état initial du système n'intervient pas dans le calcul du critère, puisque le problème est indépendant du temps.

Test du réseau

On génère aléatoirement les positions de la cible l_y et l_x. Globalement, on s'aperçoit que l'erreur est très faible dans le domaine appris (inférieure à l'erreur moyenne finale). Toutefois, plus on s'en éloigne, plus le critère augmente. Ceci confirme que les exemples pour lesquels

neurone 1	$l_x \rightarrow w_{12}(1,1)$= +1.320868
	$l_y \rightarrow w_{12}$ (2,1)= -1.278282
	$c_1 \rightarrow w_{12}$ (3,1)= -0.601213
	$c_2 \rightarrow w_{12}$ (4,1)= -1.641159
	$c_3 \rightarrow w_{12}$ (5,1)= +2.016644
	$s_1 \rightarrow w_{12}$ (6,1)= -2.967143
	$s_2 \rightarrow w_{12}$ (7,1)= +0.283052
	$s_3 \rightarrow w_{12}$ (8,1)= -0.682655
	w_{12} (seuil,1)= +0.159123
neurone 2	$l_x \rightarrow w_{12}(1,2)$= -1.736236
	$l_y \rightarrow w_{12}(2,2)$= -1.325125
	$c_1 \rightarrow w_{12}(3,2)$= +1.024949
	$c_2 \rightarrow w_{12}(4,2)$= -0.129885
	$c_3 \rightarrow w_{12}(5,2)$= +0.542738
	$s_1 \rightarrow w_{12}(6,2)$= -1.392586
	$s_2 \rightarrow w_{12}(7,2)$= +1.308135
	$s_3 \rightarrow w_{12}(8,2)$= -0.895982
	w_{12}(seuil,2)= -0.171615
neurone 3	$l_x \rightarrow w_{12}(1,3)$= -1.759028
	$l_y \rightarrow w_{12}(2,3)$= +1.110728
	$c_1 \rightarrow w_{12}(3,3)$= -2.332082
	$c_2 \rightarrow w_{12}(4,3)$= +0.158489
	$c_3 \rightarrow w_{12}(5,3)$= -0.437483
	$s_1 \rightarrow w_{12}(6,3)$= +1.076385
	$s_2 \rightarrow w_{12}(7,3)$= -1.009113
	$s_3 \rightarrow w_{12}(8,3)$= +0.762563
	w_{12}(seuil,3)= -0.295441

neurone 4	$l_x \rightarrow w_{12}(1,4)$= +1.761554
	$l_y \rightarrow w12(2,4)$= -0.132334
	$c_1 \rightarrow w_{12}(3,4)$= -0.235067
	$c_2 \rightarrow w_{12}(4,4)$= -0.165563
	$c_3 \rightarrow w_{12}(5,4)$= +0.017149
	$s_1 \rightarrow w_{12}(6,4)$= +0.086462
	$s_2 \rightarrow w_{12}(7,4)$= +0.285524
	$s_3 \rightarrow w_{12}(8,4)$= -0.307973
	w_{12}(seuil,4)= +2.377649
neurone 5	$l_x \rightarrow w_{12}(1,5)$= +0.580207
	$l_y \rightarrow w_{12}(2,5)$= -1.545673
	$c_1 \rightarrow w_{12}(3,5)$= +1.246955
	$c_2 \rightarrow w_{12}(4,5)$= +0.096072
	$c_3 \rightarrow w_{12}(5,5)$= -0.018808
	$s_1 \rightarrow w_{12}(6,5)$= +0.145747
	$s_2 \rightarrow w_{12}(7,5)$= -0.097270
	$s_3 \rightarrow w_{12}(8,5)$= +0.137156
	w_{12}(seuil,5)= +0.154870
neurone 6	$l_x \rightarrow w_{12}(1,6)$= -1.571191
	$l_y \rightarrow w_{12}(2,6)$= +2.023641
	$c_1 \rightarrow w_{12}(3,6)$= -2.187135
	$c_2 \rightarrow w_{12}(4,6)$= +1.371148
	$c_3 \rightarrow w_{12}(5,6)$= -2.610556
	$s_1 \rightarrow w_{12}(6,6)$= +3.737988
	$s_2 \rightarrow w_{12}(7,6)$= -0.068950
	$s_3 \rightarrow w_{12}(8,6)$= +0.470451
	w_{12}(seuil,6)= +0.497652

Tableau 4.1: *Valeur des poids de la couche 1 vers la couche 2*

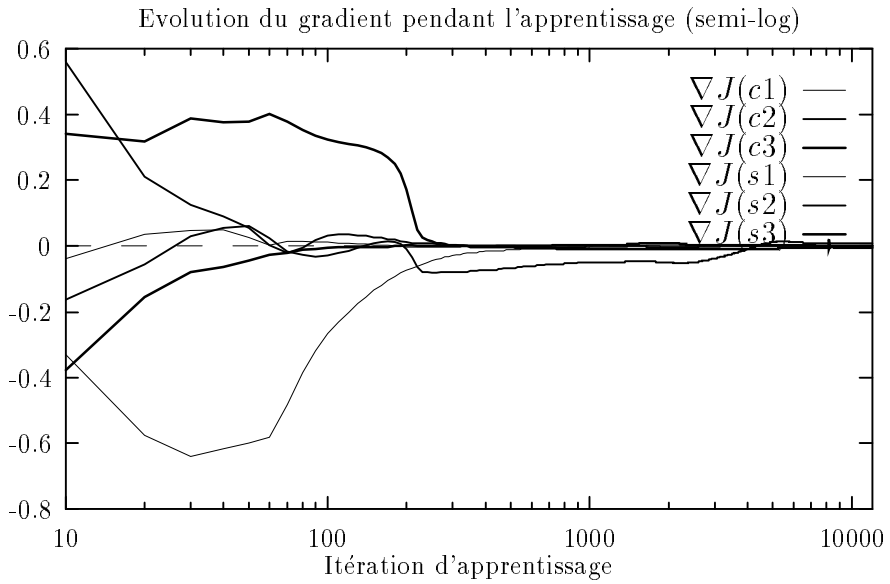

Fig. 4.9: *Apprentissage*

neurone 1 ($cosq_1$)	w_{23}(1,1)= -0.020313
	w_{23}(2,1)= -0.750752
	w_{23}(3,1)= +0.058581
	w_{23}(4,1)= -1.944958
	w_{23}(5,1)= -2.234124
	w_{23}(6,1)= +0.366955
	w_{23}(seuil,1)= -1.839861
neurone 2 ($cosq_2$)	w_{23}(1,2)= -0.074084
	w_{23}(2,2)= -0.735903
	w_{23}(3,2)= -0.876270
	w_{23}(4,2)= +0.136011
	w_{23}(5,2)= +0.374547
	w_{23}(6,2)= +0.027124
	w_{23}(seuil,2)= +0.124186
neurone 3 ($cosq_3$)	w_{23}(1,3)= -1.754799
	w_{23}(2,3)= +0.380603
	w_{23}(3,3)= +0.485490
	w_{23}(4,3)= +0.766356
	w_{23}(5,3)= -0.868180
	w_{23}(6,3)= -0.100061
	w_{23}(seuil,3)= +1.570377

neurone 4 ($sinq_1$)	w_{23}(1,4)= -1.323278
	w_{23}(2,4)= +1.071499
	w_{23}(3,4)= +1.817966
	w_{23}(4,4)= -0.000129
	w_{23}(5,4)= +0.831654
	w_{23}(6,4)= +3.150444
	w_{23}(seuil,4)= +0.390342
neurone 5 ($sinq_2$)	w_{23}(1,5)= +1.175856
	w_{23}(2,5)= -0.089547
	w_{23}(3,5)= -1.172457
	w_{23}(4,5)= -0.563746
	w_{23}(5,5)= -0.693259
	w_{23}(6,5)= -4.009666
	w_{23}(seuil,5)= +1.015139
neurone 6 ($sinq_3$)	w_{23}(1,6)= +1.799211
	w_{23}(2,6)= -0.259919
	w_{23}(3,6)= -0.743580
	w_{23}(4,6)= -0.353303
	w_{23}(5,6)= -0.121251
	w_{23}(6,6)= -2.442023
	w_{23}(seuil,6)= +0.115901

Tableau 4.2: *Valeur des poids de la couche 2 vers la couche 3*

la cible est trop éloignée pénalisent l'apprentissage; les contraintes empêchent d'annuler la distance, donc de bien apprendre le critère. Par contre, les bons résultats pour des situations non présentes dans la base d'exemples, confirment la généralisation du critère à toutes ces situations. La figure 4.10 montre l'évolution du critère pour quelques changement de cible. L'étude de la valeur des poids reliant la couche d'entrée à la couche cachée a montré que

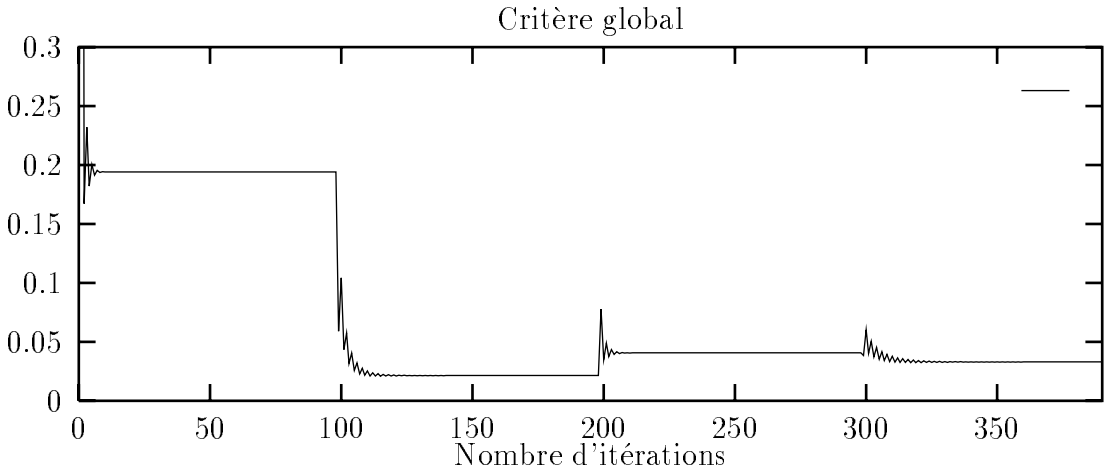

Fig. 4.10: *Résultat après l'apprentisage Hors-Ligne*

la plupart des connexions affectées aux sinus et cosinus étaient faiblement pondérées. Nous avons donc supprimé ces entrées (en les forçant à zéro), puis testé le réseau de la même façon que ci-dessus. Les résultats obtenus sont les mêmes. Le réseau n'utilise donc pas l'information de position courante. Il opère une inversion du modèle géométrique. On peut ainsi supprimer du vecteur d'entrée les entrées correspondants aux sinus et cosinus. La première couche n'a donc plus que 2 entrées l_x et l_y.

Si on considère cette structure comme une boucle ouverte au sens de l'automatique, l'asservissement sur la position cartésienne désirée impose le retour d'erreur :

$$\begin{cases} \epsilon_x &= l_x - X_p \\ \epsilon_y &= l_y - Y_p \end{cases}$$

Cette approche sera utilisée dans le cas de la commande en position d'un Robot Mobile (Partie II). Pour le problème que nous venons d'étudier, elle remet en cause la base d'apprentissage puisque l'entrée devient une erreur à minimiser. Le réseau que l'on obtiendrait alors serait complètement différent, pour la valeur des poids, de celui que nous avons calculé ici.

Conclusion

L'apprentissage Hors Ligne du critère a permis sa généralisation mais dans un domaine limité par les contraintes. Nous pouvons affirmer que notre méthode est validée. La minimisation

n'est cependant pas tout à fait optimale puisqu'il existe une faible erreur en fin d'apprentissage. Cependant, il est possible de corriger cette erreur par l'apprentissage En-Ligne. C'est l'objet de l'étude du paragraphe suivant.

4.4.6 Apprentissage en ligne

C'est une méthode qui doit permettre d'affiner le calcul du réseau précédemment obtenu Hors-Ligne. Nous reprenons la structure de principe 4.4 et l'appliquons à notre problème en générant d'une manière aléatoire les positions de la cible. On peut comparer l'évolution du critère sur les figures 4.11 et 4.10. Les résultats sont intéressants puisque à chaque position

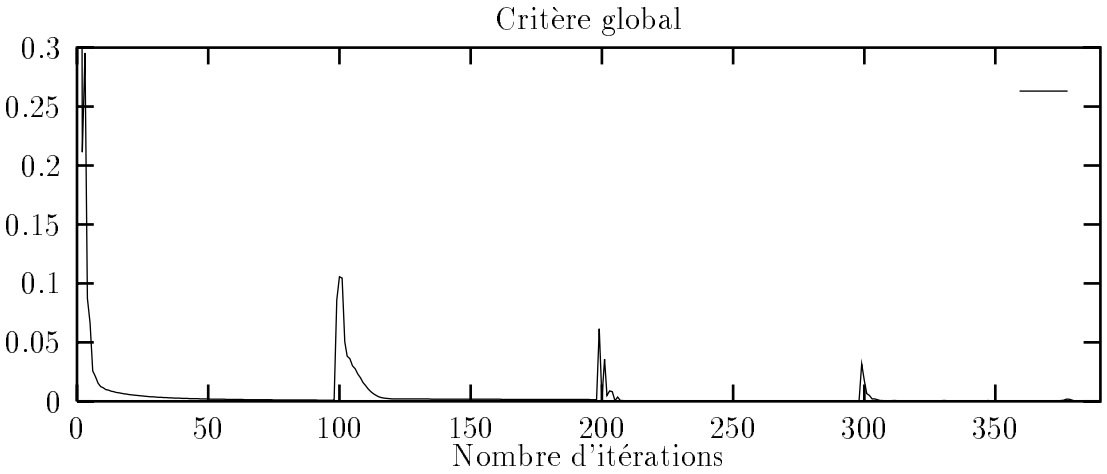

Fig. 4.11: *Résultat pendant l'apprentisage En-Ligne*

de la cible, l'algorithme converge et minimise presque totalement le critère. Si l'on poursuit plus longtemps l'apprentissage En Ligne, on remarque que le réseau "répond" de plus en plus vite. C'est à dire que l'adaptation lui permet d'améliorer sa réponse.

Ceci est confirmé si on recommence le test du paragraphe 4.4.5 avec le réseau qui s'est affiné en-ligne. Le figure 4.12 montre en effet que le critère est mieux minimisé qu'avant l'apprentissage en-ligne. On ne peut donc conclure que l'utilisation en-ligne de la rétropropagation indirecte permet d'améliorer nettement les performances du réseau.

Remarque

L'utilisation de cette structure avec un réseau vierge (i.e. n'ayant subit aucun apprentissage) ne fonctionne pas. On constate en effet qu'il y a trop d'oscillations dues à une trop forte valeur du gradient à chaque instant. Il n'y a pas de convergence.

Conclusion

L'apprentissage En-Ligne améliore beaucoup les performances d'un RNF obtenu par l'apprentissage Hors-Ligne. Cependant, on ne peut pas encore parler de généralisation mais

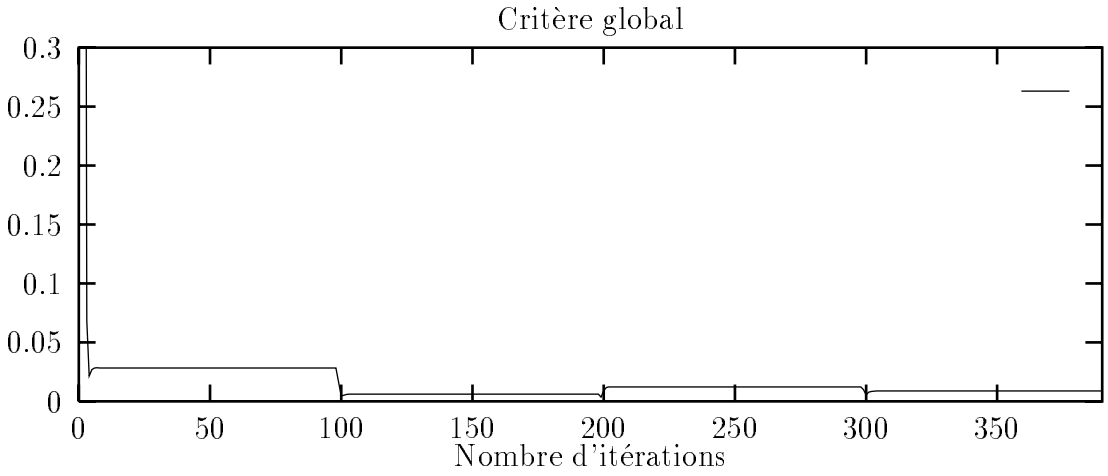

Fig. 4.12: *Résultat après l'apprentissage En-Ligne*

plutôt d'adaptation du réseau. C'est de plus une méthode qui doit permettre de corriger les imperfections du système contrôlé.

4.4.7 Conclusion

L'étude que nous venons de mener avait un but illustratif. Elle a permis de valider la rétropropagation indirecte. C'est une méthode qui permet de calculer un contrôleur par une description analytique de l'objectif à atteindre, et le calcul de son gradient par rapport aux paramètres de commande. Il y a deux concepts d'utilisation :

- **l'apprentissage Hors-Ligne:**
  qui permet d'obtenir le réseau optimal (ou proche de l'optimal), et de l'utiliser pour la commande tel qu'il est;
- **l'apprentissage En-Ligne:**
  qui permet d'affiner le calcul du réseau en l'adaptant périodiquement. La période d'adaptation est choisie par l'utilisateur. Ce concept de contrôle adaptatif est intéressant, puisqu'à notre connaissance, il n'existait pas avec les méthodes habituelles d'apprentissage par rétropropagation limitées par le calcul En-Ligne de la sortie désirée.

Partie II

Application à la commande d'un Robot Mobile

Chapitre 5

Introduction à la commande Cartésienne d'un Robot Mobile

Nous nous proposons dans cette partie d'étudier le contrôle de la position et de l'orientation d'un robot mobile par les techniques d'apprentissage direct et indirect présentées en première partie.

C'est un travail préliminaire, nécessaire à toute investigation future dans la commande par réseaux de neurones de systèmes robotiques plus complexes.

Le problème posé est donc de piloter le robot mobile ROMEO vers une position et une orientation désirées. Les moteurs sont asservis en vitesse. C'est un problème cinématique (commande en vitesse) pour lequel les contraintes cinématiques du robot, et son caractère physique (bande passante faible, saturation des consignes) sont importants.

Nous présentons au prochain chapitre le robot mobile, et les conditions expérimentales. Le chapitre suivant présente une solution basée sur l'apprentissage direct. Ensuite, l'apprentissage indirect (hors-ligne et en-ligne) est utilisé en fin de partie.

5.1 Description du Robot Mobile

Le robot mobile ROMEO (photo 5.2) est doté à l'arrière de deux roues motrices indépendantes asservies en vitesse, et de deux roues folles à l'avant.
Sous l'hypothèse de roulement sans glissement, le modèle cinématique du robot exprime la vitesse absolue du point M, situé au centre de l'axe des roues motrices, par les équations:

$$\left\{ \begin{array}{rcl} \dot{X}_M & = & \frac{r}{2}(\dot{q_1} + \dot{q_2})cos\theta \\ \dot{Y}_M & = & \frac{r}{2}(\dot{q_1} + \dot{q_2})sin\theta \\ \dot{\theta} & = & \frac{r}{2R}(\dot{q_1} - \dot{q_2}) \end{array} \right. \tag{5.1}$$

avec:

- r: rayon des roues.
- R: demi-longueur de l'axe des roues.

La caractéristique essentielle de ce système est d'être non-holonome (les deux premières équations de 5.1 ne sont pas intégrables).

5.2 Problème de la commande en position et en orientation

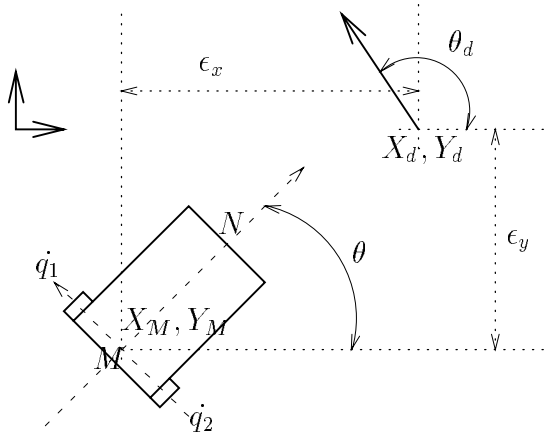

Fig. 5.1: *Commande en position du robot mobile*

La commande de la position et de l'orientation du robot mobile consiste à déterminer une suite temporelle de consignes $(\dot{q}_1, \dot{q}_2)$ pour positionner dans le repère absolu, le point M de coordonnées (X_M, Y_M), situé au centre de l'axe des roues motrices, sur le point de coordonnées (X_d, Y_d), avec l'orientation désirées θ_d.

L'étude, par les méthodes académiques de l'automatique non-linéaire, de la commande de la position et de l'orientation du robot mobile nécessite des appuis théoriques importants qui ne seront pas développés dans ce rapport.

K. Ait-Abderrahim ([*Ait 93*]) et C. Samson, montrent *qu'il n'existe pas de retour d'état stabilisant continu $U(X, Y, \theta)$ pour le robot mobile à deux roues motrices indépendantes, c'est à dire qu'il n'existe pas de commande continue $U(X, Y, \theta)$ faisant converger en même temps X, Y et θ à zéro, quelques soient les conditions initiales.* Il propose *un retour d'état continu instationnaire périodique.*

Nous proposons d'utiliser les méthodes connexionnistes. Les résultats obtenus sont à notre avis intéressants, tant pour la simplicité de leur mise en oeuvre, que pour les explications théoriques qu'elles pourraient susciter du point de vue de l'automatique.

5.3 Description du Site expérimental

Le site expérimental décrit figure 5.3 comprend :

- le robot mobile ROMEO. Il pèse environ $100kg$. Ses moteurs sont asservis en vitesse, et les consignes sont saturées pour limiter sa vitesse à $0,1$ m/s. Il a par conséquent peu d'effets dynamiques.
- une station SUN 4 connectée via une liaison série à un modem. La vitesse de transmission est de 9600 bauds.

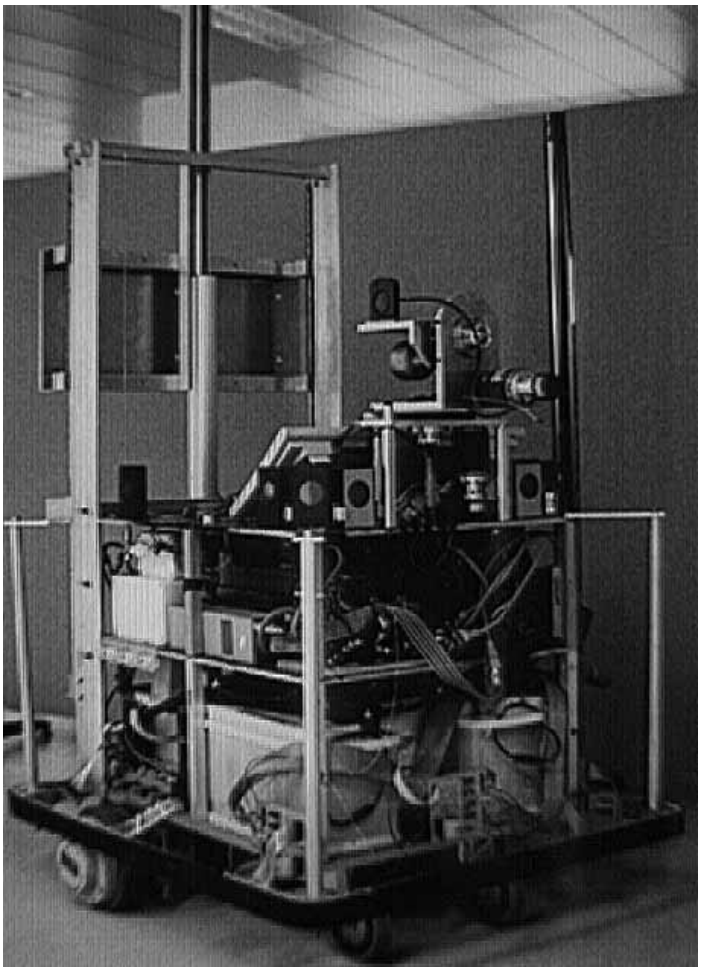

Fig. 5.2: *Robot mobile ROMEO*

Le simulateur de réseaux de neurones "neuro4" a été développé en langage C sous UNIX par E. Desjardin ([*Desj 93*]) pour des applications de reconnaissance de formes. Nous l'avons modifié et adapté pour des applications de commande.

La transmission radio est uniquement utilisée dans un but de simplicité et de rapidité de mise en oeuvre. La station de travail sur laquelle est exécuté "neuro4" émet les consignes en

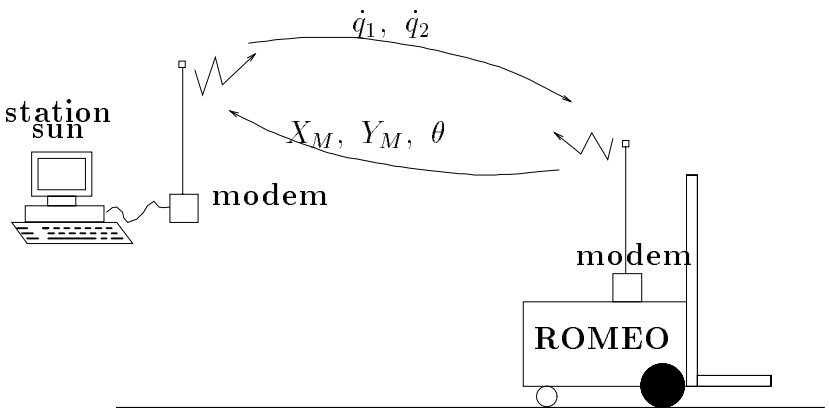

Fig. 5.3: *Site expérimental utilisé*

vitesses des roues. Dès qu'elles sont reçues par le robot, celui-ci réémet les mesures de positions et d'orientation (X_M, Y_M, θ). Ces mesures sont calculées par odométrie ([*Dela 91*]). Le temps d'émission/réception, que l'on peut considérer équivalent à une période d'échantillonnage, est d'environ $0,5$ secondes.

Chapitre 6

Commande par Apprentissage Direct

Le problème de la commande cartésien d'un robot mobile non-holonome peut être en partie résolu par la technique de rétropropagation directe. La méthode présentée se rapproche beaucoup de l'apprentissage de règles floues par un RNF[1]. Cependant, les comportements désirés présentés au réseau sont très élémentaires puisque l'univers des consignes est discrétisé en trois états : $-max$, nul, $+max$.

D'autres méthodes d'apprentissage de manoeuvres existent, comme le "clonage" [*Daxw 94*], qui consiste à apprendre les manoeuvres effectuées par un opérateur qui pilote le robot. Cependant, toutes ces techniques n'apportent pas autant d'autonomie au système pour l'apprentissage et la commande, que l'apprentissage indirect étudié aux chapitres suivants.

6.1 Analyse du problème

L'idée est de décomposer la tâche globale en sous tâches regroupant chacune des états similaires du robot. A chacun d'eux correspond une manoeuvre élémentaire pour se rapprocher de l'objectif. La succession de ces manoeuvres déplace le système dans une suite de configurations appartenant à des sous tâches différentes.

L'apprentissage de ces manoeuvres, donc la reconnaissance de ces sous tâches, est un problème de classification au sens de la classification par RNF. Pour déterminer ces sous tâches, nous utilisons la position $(X',\ Y')$ de l'origine dans le repère du robot.

6.1.1 Structure de Contrôle

La structure de commande du robot est schématisée figure 6.1. Le vecteur d'entrée du réseau est de la forme :

[1] Réseau de Neurones Formels

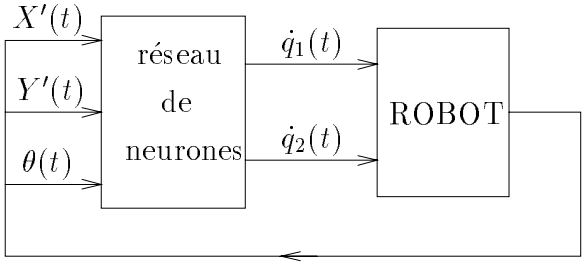

Fig. 6.1: *Commande du chariot mobile*

$$\begin{cases} X' &= -X_M cos\theta - Y_M sin\theta \\ Y' &= -Y_M cos\theta + X_M sin\theta \\ \theta' &= \theta \end{cases}$$

6.1.2 Classes de manoeuvre

Il y a au minimum 9 cas de figures (décrits figure 6.2), donc 9 manoeuvres différentes, pour neuf positions de l'origine quelque soit l'orientation du robot. De ces 9 situations nous

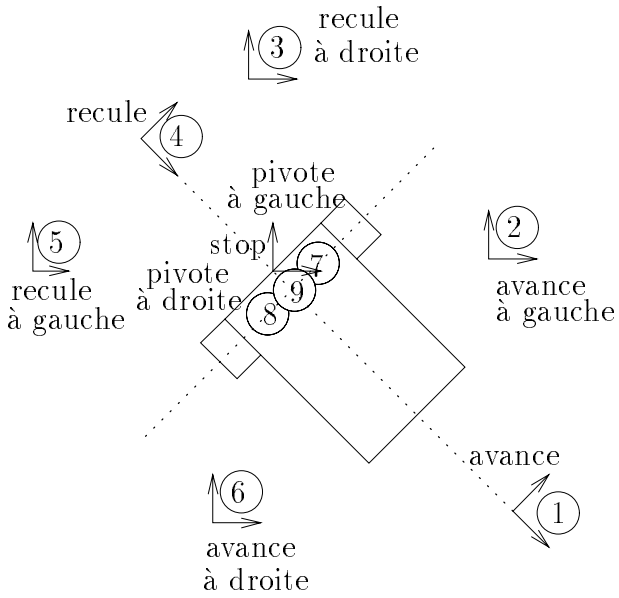

Fig. 6.2: *Les différentes classes de manoeuvre*

déduisons sur le tableau 6.1 les états en vitesses du robot. A partir de l'analyse des signes des vitesses V et $\dot{\theta}$, on déduit les valeurs désirées maximales de $\dot{q_1}$ et $\dot{q_2}$ (tableau 6.2). En présentant ainsi au réseau de neurones les valeurs maximales des commandes désirées, on espère que l'apprentissage lui permette d'interpoler la commande nécessaire pour passer d'un cas à un autre.

	X'	Y'	θ	$V = \frac{r}{2}(\dot{q_1} + \dot{q_2})$	$\dot{\theta} = \frac{r}{2R}(\dot{q_1} - \dot{q_2})$
cas 1	> 0	0	$\forall$	> 0	0
cas 2	> 0	> 0	$\forall$	≥ 0	> 0
cas 3	< 0	> 0	$\forall$	≤ 0	≤ 0
cas 4	< 0	0	$\forall$	< 0	0
cas 5	< 0	< 0	$\forall$	≤ 0	≥ 0
cas 6	> 0	< 0	$\forall$	≥ 0	≤ 0
cas 7	0	0	< 0	0	> 0
cas 8	0	0	> 0	0	< 0
cas 9	0	0	0	0	0

Tableau 6.1: *Signes des vitesses tangentielle et angulaire*

	cas 1	cas 2	cas 3	cas 4	cas 5	cas 6	cas 7	cas 8	cas 9
$\dot{q_1}$	+M	+M	-M	-M	+M	-M	+M	-M	0
$\dot{q_2}$	+M	-M	+M	-M	-M	+M	-M	+M	0

Tableau 6.2: *Valeurs des consignes pour chaque classe*

6.2 Apprentissage

Construction de la base d'exemples

On génère N exemples par classe, où X' et Y' sont aléatoires dans les limites définissant la classe. Ces données sont normalisées de la façon suivante:

$$\begin{cases} \theta_{norm} & = & \frac{\theta}{\pi} \\ X'_{norm} & = & \frac{X'}{D} \\ Y'_{norm} & = & \frac{Y'}{D} \end{cases}$$

Où D est le demi-coté d'un carré centré sur l'origine. Nous prendrons $D = 4$ mètres.

Remarque

Nous utilisons ici en entrée un angle dont l'amplitude est π. Cela peut paraître contraire à ce que nous avons dit au chapitre 2 . Cependant, ici la démarche est totalement différente puisque les valeurs positives et négatives de θ n'ont pas la même signification et entraînent des manœuvres différentes.

Choix du réseau

Les meilleurs résultats ont été obtenu par un réseau à une couche cachée de 9 neurones, c'est à dire un réseau 3-9-2. Il y a donc $3 \times 9 + 9 \times 2 = 45$ connexions.

Protocole d'apprentissage

Base d'apprentissage: 270 exemples uniformément repartis dans chaque classe (30 par classe). Viscosité = 0 . $\lambda = 0,5$. Nombre d'itérations: 500 .

6.2.1 Surfaces de commande avant apprentissage

Nous allons évaluer la qualité de l'apprentissage sous forme de surfaces. C'est à dire que nous tracerons les valeurs de $\dot{q_1}$, $\dot{q_2}$, $\dot{\theta}$ et V en fonction de X' et Y', cela pour deux valeurs de θ (0 et $\frac{\pi}{2}$). Ceci permet de représenter en 3 dimensions le tableau 6.2 appris par le réseau. On peut tracer ces surfaces avant l'apprentissage pour observer leurs modifications. La figures 6.3 montre les consignes de vitesse calculées par le réseau initial en fonction des erreurs cartésiennes pour une orientation de $\theta = 0$. La figure 6.4 montre la vitesse angulaire $\dot{\theta}$ et la

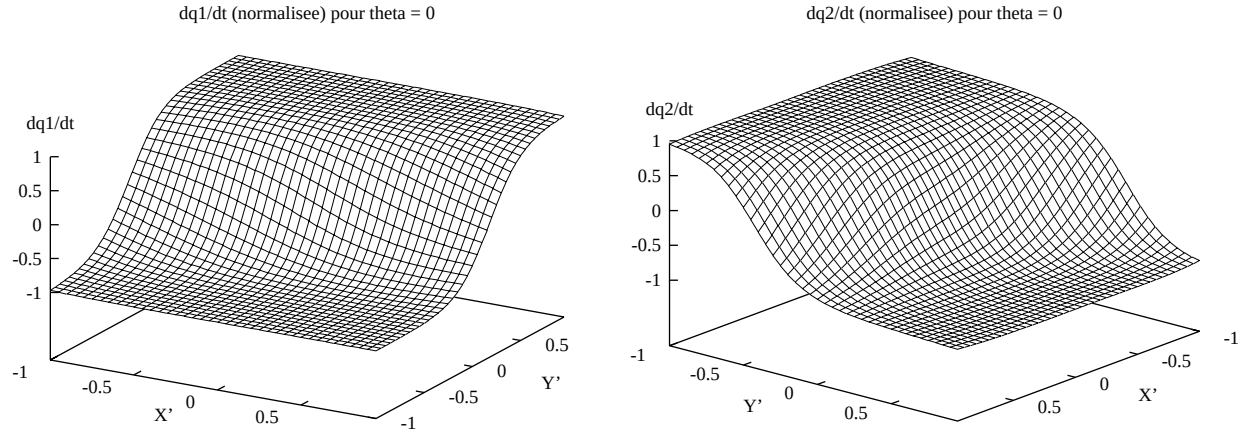

Fig. 6.3: *Vitesses des roues avant apprentissage ($\theta = 0$)*

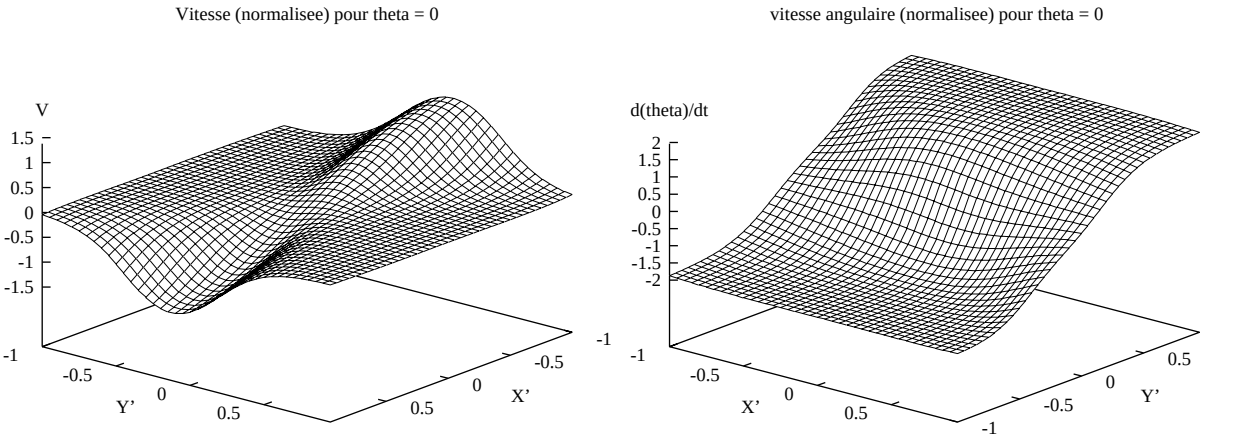

Fig. 6.4: *Vitesses linéaire et angulaire avant apprentissage ($\dot{\theta} = 0$)*

vitesse tangentielle à la trajectoire déduitent au point (X_M, Y_M) pour la même orientation.

6.2.2 Surfaces de commande après apprentissage

L'erreur finale moyenne est assez importante (de l'ordre de $0,2$) pour une valeur de départ d'environ 2. Ceci veut dire que l'identification des classes est difficile pour le réseau, sûrement à cause du recouvrement contradictoire de certaines d'entre elles. Cependant, les surfaces de commande après apprentissage montrent clairement que les différentes classes (donc manoeuvres) ont été reconnues et apprises.

Nous avons tracé les valeurs des vitesses des roues $\dot{q_1}$ et $\dot{q_2}$, de V et de $\dot{\theta}$ en fonction des valeurs de $X^{'}$ et $Y^{'}$, pour $\theta = 0$ (6.5 et 6.6), et $\theta = 180^o$) (figures 6.7 et 6.8). On remarque

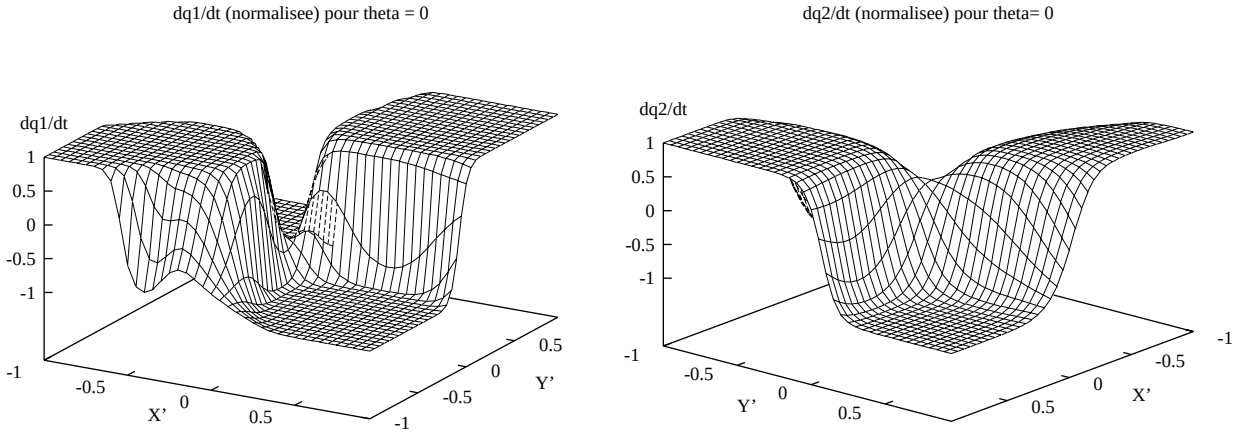

Fig. 6.5: *Vitesses des roues après apprentissage ($\theta = 0^o$)*

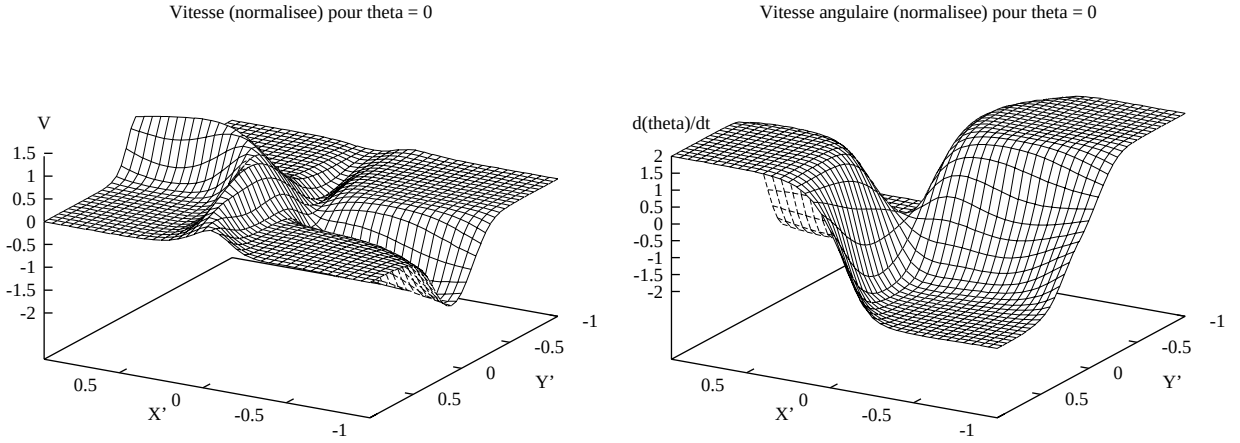

Fig. 6.6: *Vitesses linéaire et angulaire après apprentissage ($\theta = 0^o$)*

que la généralisation est bien faite par le réseau, puisqu'il sait construire une sortie pour des valeurs d'entrées non connues, c'est à dire, non présentes dans la base d'apprentissage.

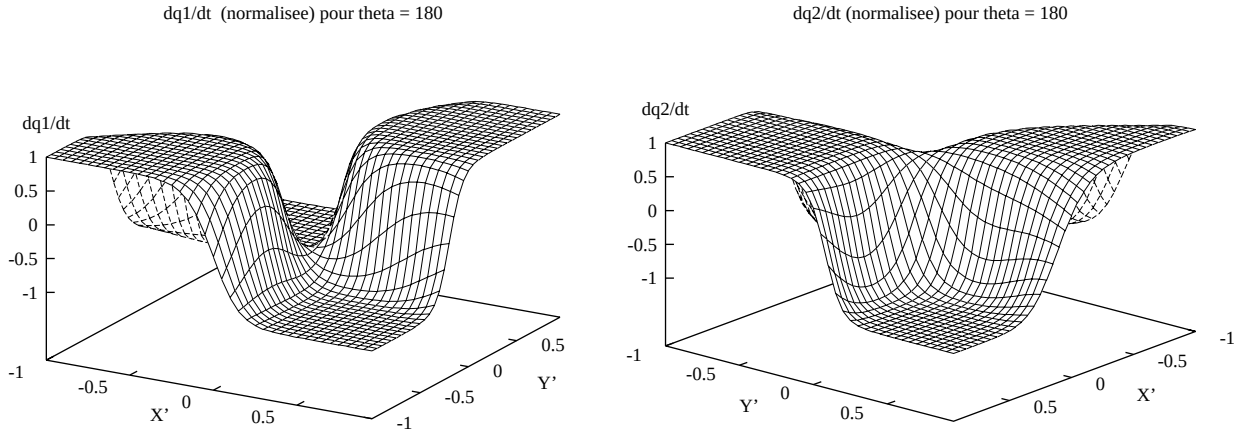

Fig. 6.7: *Vitesses des roues après apprentissage* ($\theta = 180^o$)

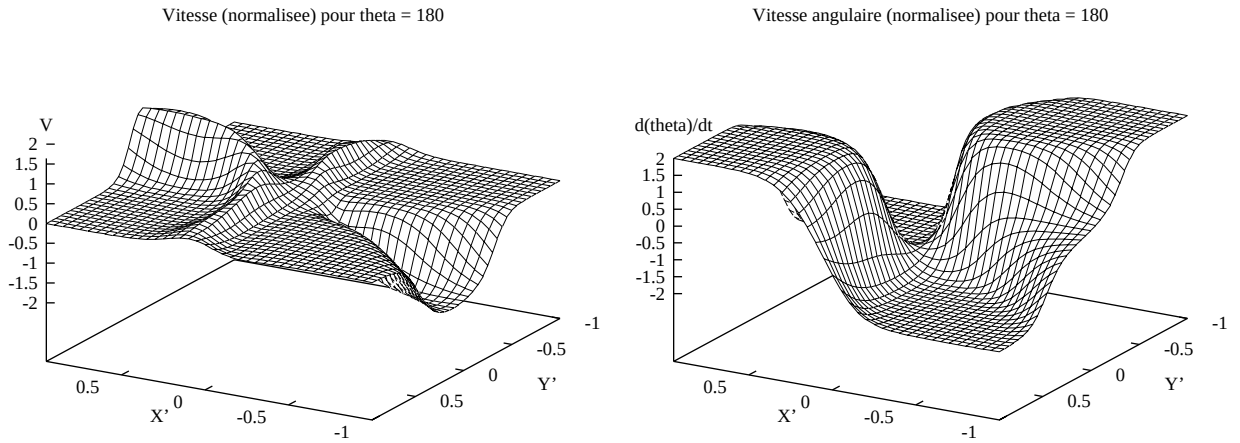

Fig. 6.8: *Vitesses linéaire et angulaire après apprentissage* ($\theta = 180^o$)

6.2.3 Valeur des poids du réseau

Les poids sont donnés en annexe A.1 tel que l'affiche le logiciel que nous utilisons (neuro4). Si on étudie les valeurs des poids du réseau obtenu par l'apprentissage, on constate qu'elles sont soit forte (de l'ordre de 1 à 2), soit presque nulles. On peut l'expliquer par le fait que l'apprentissage oblige le réseau à calculer une valeur maximale en sortie.
On ne peut toutefois pas tirer de conclusion plus précise sur ces paramètres.

6.3 Simulation du Contrôle du robot

A partir d'une configuration initiale du robot, on laisse évoluer la boucle Réseau+robot jusqu'à un état final où la vitesse des roues devient nulle. On obtient les courbes suivantes (6.9)

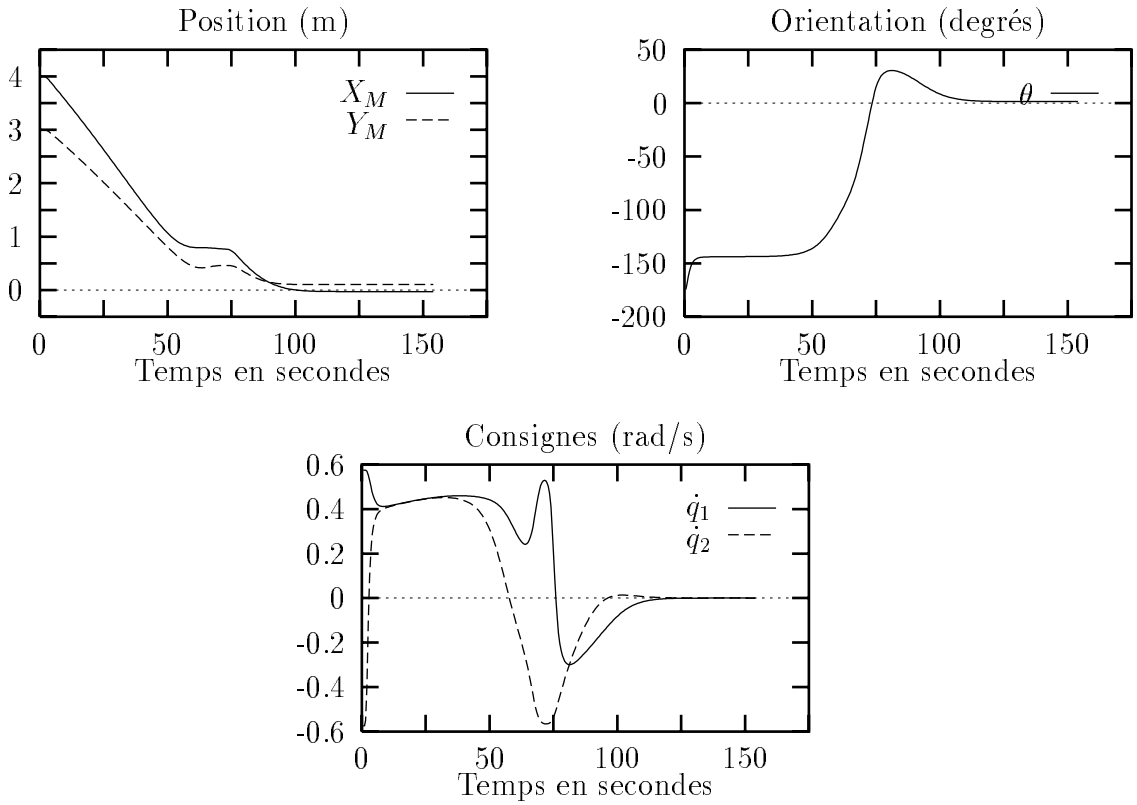

Fig. 6.9: *Simulation pour $X_M = 4m$, $Y_M = 3m$, $\theta = 180^o$*

Analyse du comportement du robot

La stratégie adoptée par le système est la suivante :

1. s'orienter pour avoir l'origine dans l'axe longitudinal du robot,
2. avancer (ou reculer suivant l'orientation) vers l'origine,

3. quand le point est assez proche (de l'ordre de $1m$), s'orienter vers $\theta = 0^o$, tout en manoeuvrant pour se rapprocher de l'origine.

4. minimiser au maximum l'orientation et la distance

Le contrôleur génère donc une commande qui fait exécuter au robot une manoeuvre de type demi-tour visible sur la figure 6.9 entre 50 et 75 secondes.
Cette manoeuvre existe quelque soit la position initiale. Ce phénomène est important pour deux raisons:

- il prouve que le réseau ne découple pas complètement les paramètres de position et d'orientation pour les commander.

- il prouve que les exemples de la base d'apprentissage ont été interprètés pour identifier un comportement général de contrôle de la position et de l'orientation.

Analyse des erreurs finales

On constate que les erreurs finales cartésiennes sont tout de même assez importantes tandis que l'erreur angulaire est faible ($X_M = 2cm, Y_M = 10cm, \theta = 1,51^o$). Cependant, en essayant des positions initiales plus singulières (pour lesquelles X_M tend vers 0 avec $Y_M \neq 0$), on constate, comme le montre la figure 6.10 (où $X_M = 0m$, $Y_M = 0,5m$, $\theta = 0^o$), que les performances sont assez médiocres. Néanmoins, nous pouvons être satisfait par ces résultats, puisque:

- le robot converge toujours vers la configuration finale désirée,

- l'erreur finale est acceptable compte tenu de la taille du robot.

Enfin le réseau utilisé est très petit, puisque seuls 45 paramètres (45 poids) suffisent à commander le système. Toutefois, après quelques observations sur la stratégie globale du neuro-contrôleur nous avons pu améliorer de beaucoup ces résultats. Ils sont présentés au paragraphe suivant.

6.4 Amélioration des performances du réseau

En observant, pour différentes conditions initiales, le comportement du robot, nous avons pu déduire qu'il tend d'abord à minimiser l'orientation au détriment de la position. On peut donc corriger ce comportement en privilégiant la commande de la position par rapport à la commande de l'orientation lorsque le point M est situé près de l'origine (vers $1m$). Pour ce faire, nous pondérons l'erreur en orientation par un coefficient qui devient faible devant les gains des erreurs cartésiennes lorsque la distance devient inférieure à $1m$. Après quelques essais, nous obtenons ainsi un gain variable k_θ de la forme:

$$k_\theta = 1 - 0,9e^{-10({X_M}^2+{Y_M}^2)}$$

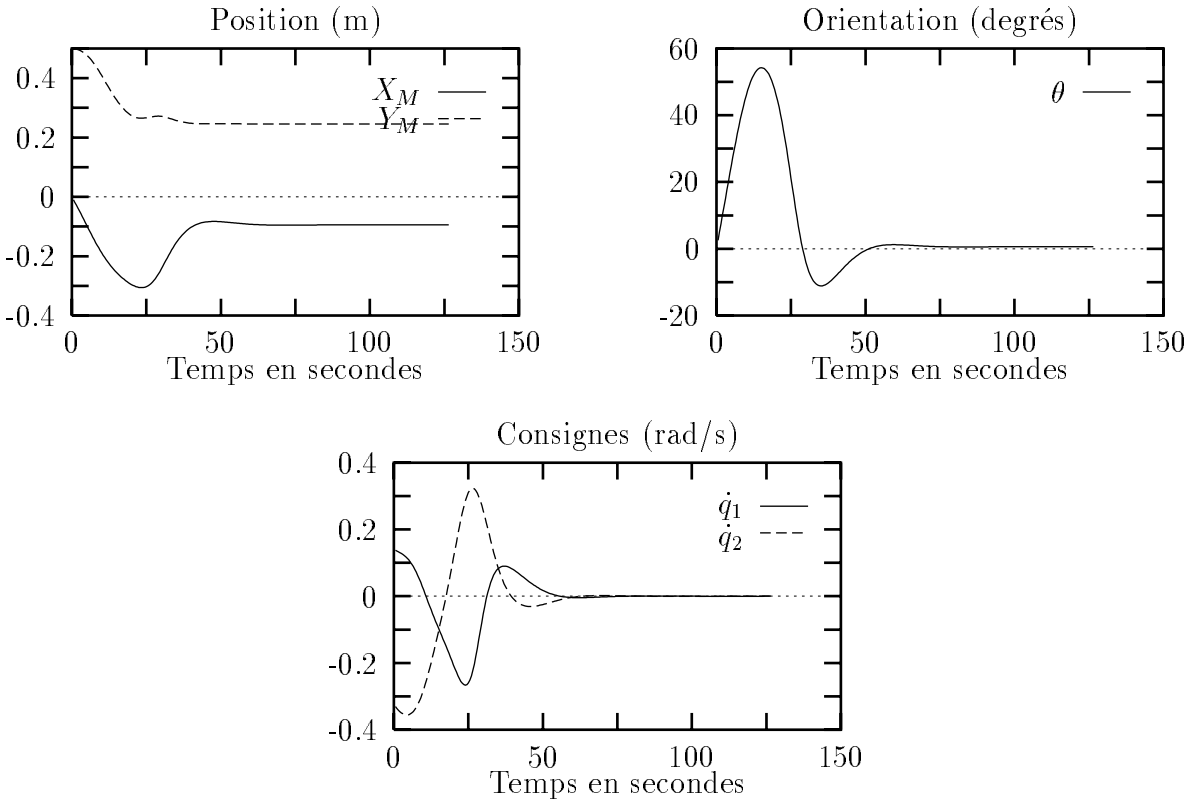

Fig. 6.10: *Simulation pour* $X_M = 0m$, $Y_M = 0,5m$, $\theta = 0^o$

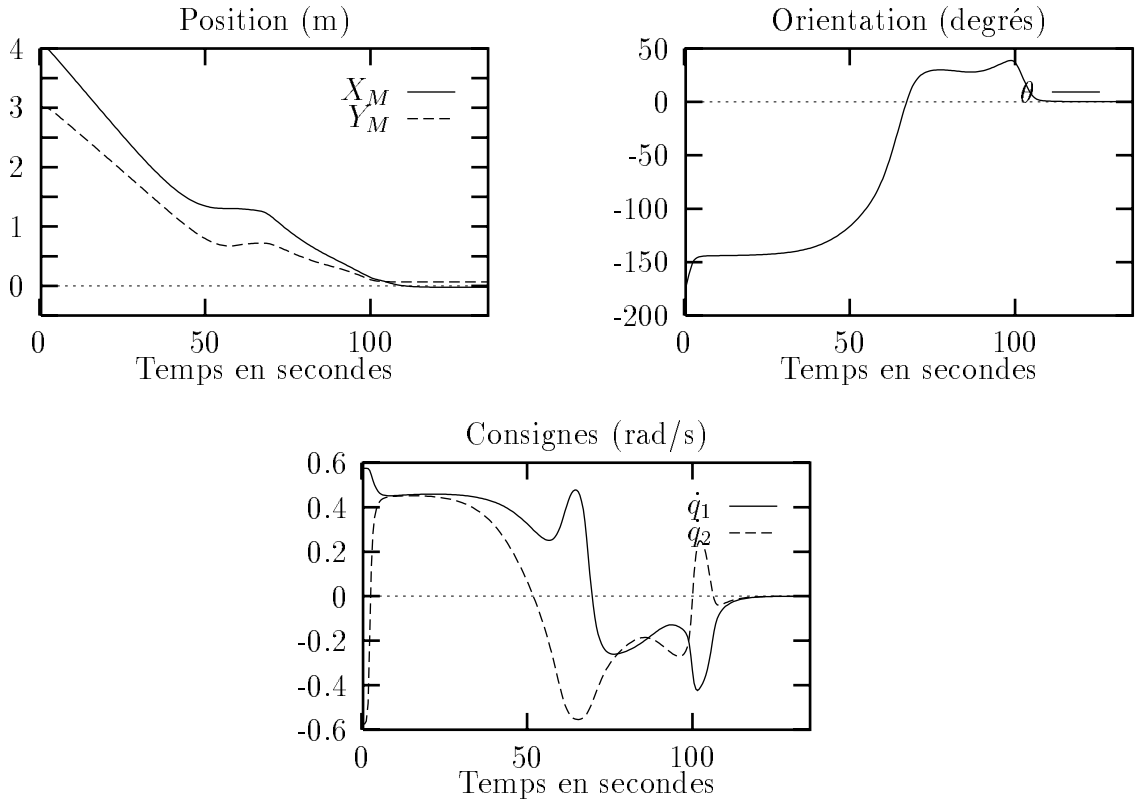

Fig. 6.11: *Simulation pour* $X_M = 3m$, $Y_M = 4m$, $\theta = 180^o$

Les résultats simulés, sont présentés sur les figures 6.11 et 6.12 pour des configurations initiales identiques aux précédentes. Les améliorations sont nettes, notamment pour la configuration singulière ($X_M = 0,0\ m$, $Y_M = 0,5\ m, \theta = 0^o$) pour laquelle la configuration finale vaut $X_M = 0,0018\ m$, $Y_M = 0,023\ m, \theta = 0,23^o$. Les résultats sont doc très bon en simulation.

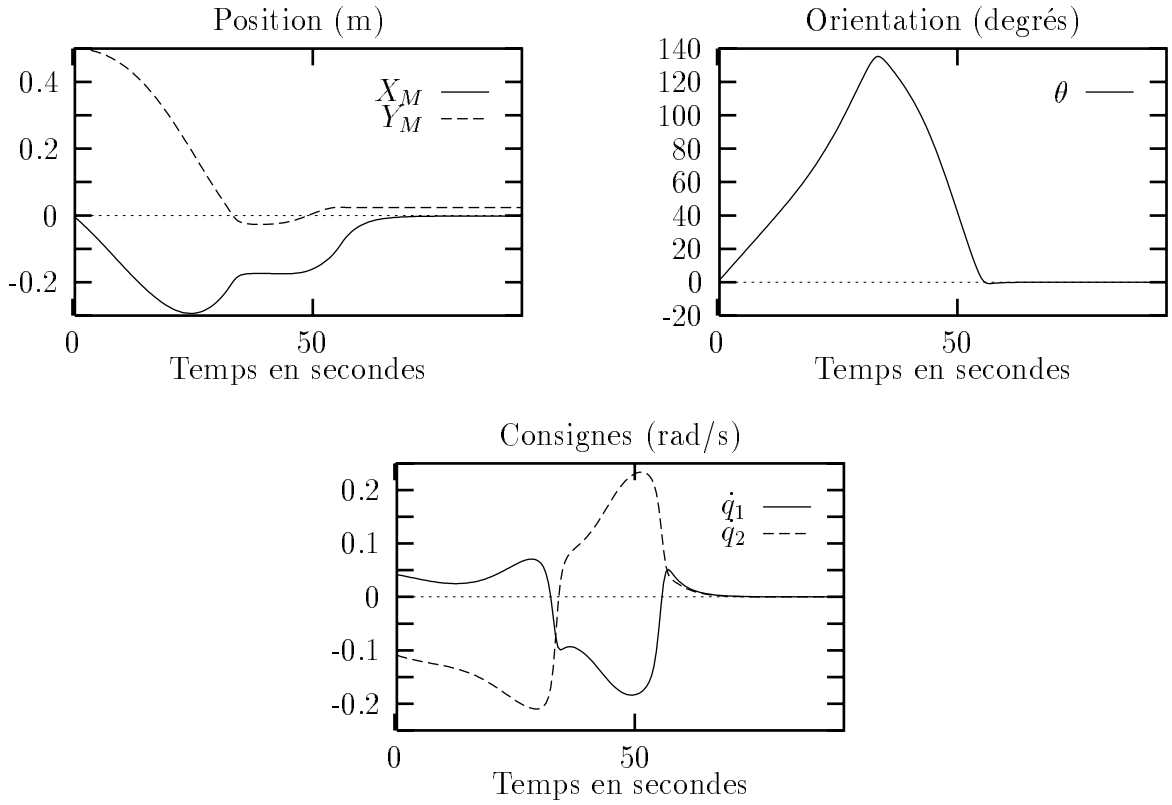

Fig. 6.12: *Simulation pour* $X_M = 0m$, $Y_M = 0.5m$, $\theta = 0^o$

6.5 Validation Expérimentale

Nous rappelons brièvement les conditions expérimentales décrites au chapitre précédent:

- Le robot est lent (0,1 m/s max).
- Les consignes sont saturées.
- La liaison entre le contrôleur et le robot est assurée par un modem.
- Le modem entraîne une perte de temps de 0,5 secondes, équivalente à une période d'échantillonage.

Les résultats des essais expérimentaux sont montrés sur les figures suivantes (6.13 et 6.14). Les erreurs évoluent avec la même allure et sont du même ordre de grandeur qu'en simulation. On remarque toutefois un phénomène de lissage des consignes du à la bande passante du robot.

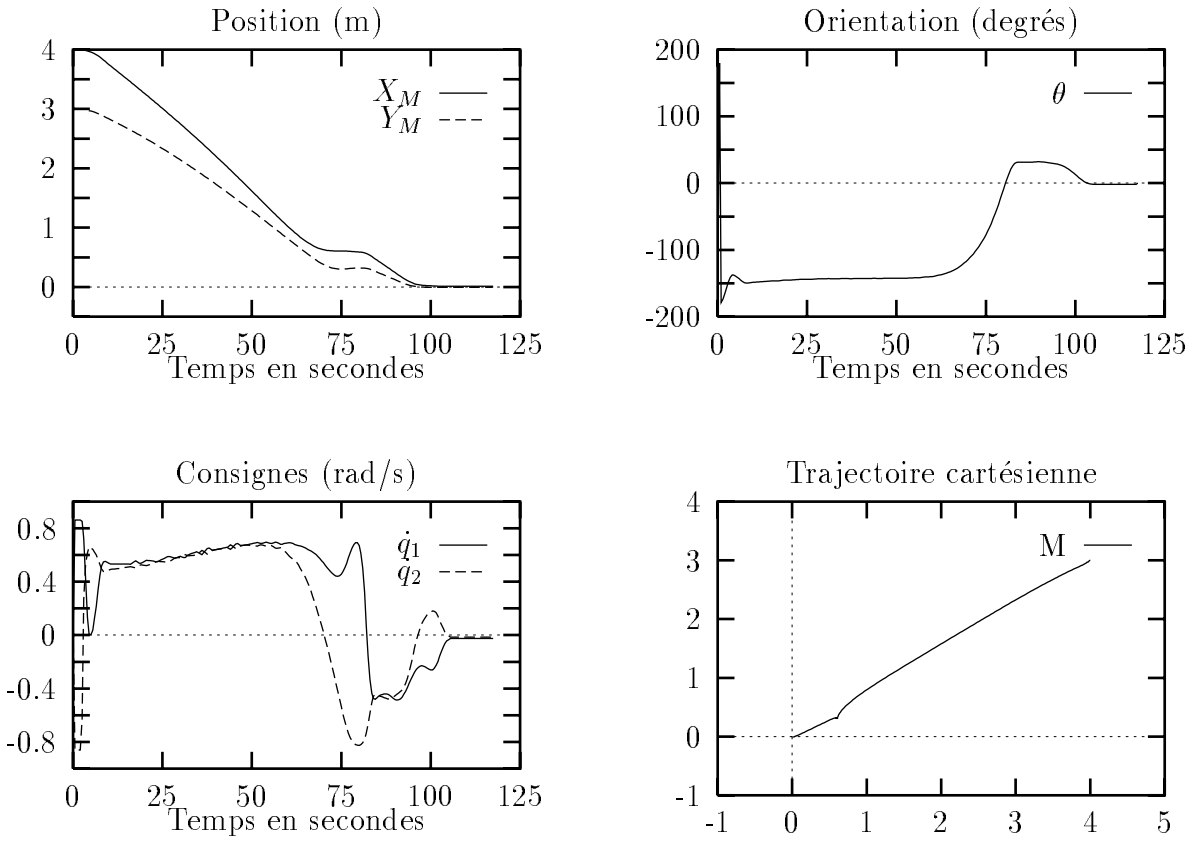

Fig. 6.13: *Expérimentation pour* $X_M = 4m$, $Y_M = 3m$, $\theta = -180^o$

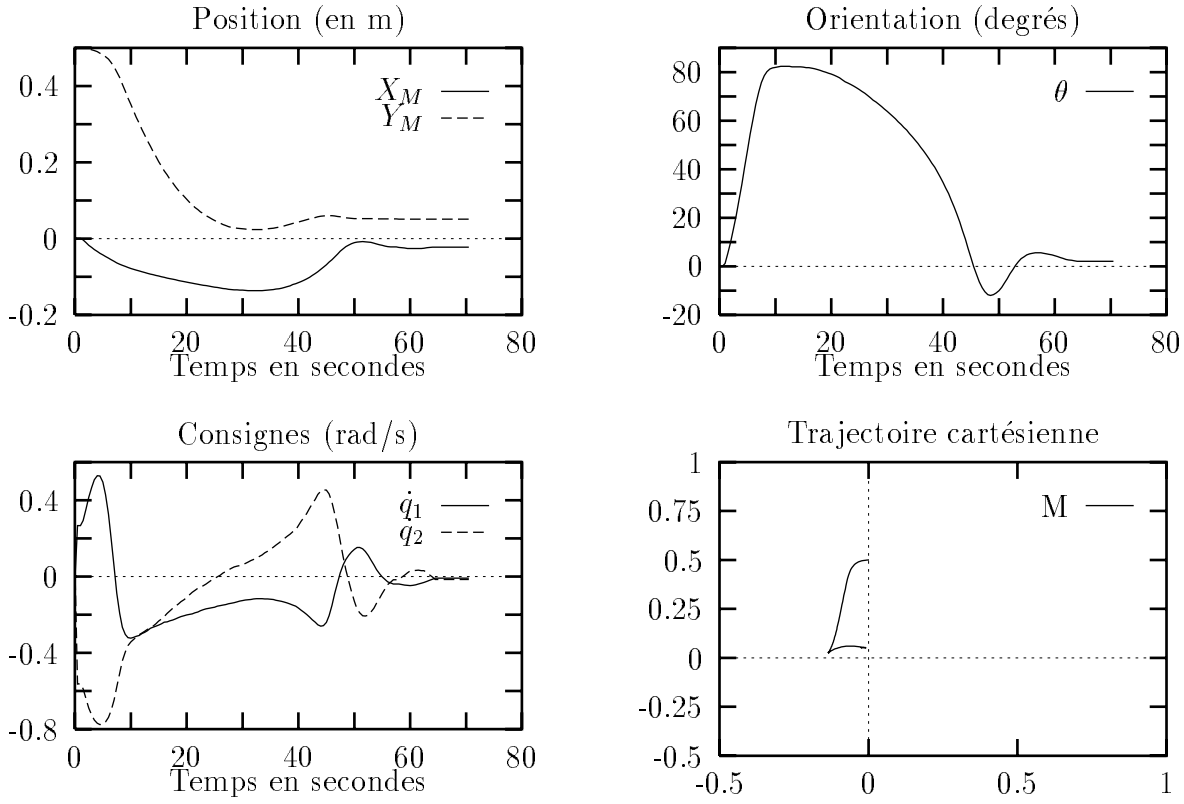

Fig. 6.14: *Expérimentation pour* $X_M = 0m$, $Y_M = 0,5m$, $\theta = 0^o$

6.6 Conclusion

Nous avons montré par l'expérience que l'on peut commander la position et l'orientation d'un robot mobile avec un réseau de neurones. Pour cela, nous avons utilisé la rétropropagation directe d'une erreur, entre la commande calculée par le réseau et une commande désirée. Cette commande désirée provient d'une expertise a priori du problème:

- nous avons analysé pour le réseau les différentes situations qui pouvaient se présenter,
- nous lui avons décrit grossièrement les différents états correspondant de sa sortie.

L'apprentissage a permis au réseau d'identifier et de généraliser une loi de commande qui utilise un retour d'état sur la position et l'orientation. Aucun modèle du système n'est utilisé pour la commande.

La méthode que nous avons utilisé ici s'apparente à l'apprentissage de règles floues. Les résultats sont intéressants. C'est une technique déjà utilisée par S-G. Kong et B.Kosko dans [*Kong 92*], et qui méritera d'être signalée dans les perspectives[2]

[2]Des travaux préliminaires actuellement en cours au L.R.P sur la commande dynamique d'un manipulateur sont prometteurs [*Ouss 94*].

Chapitre 7

Commande par Apprentissage Indirect Hors-Ligne

Pour minimiser, dans l'espace des configurations (X, Y, θ), la distance du point M au point désiré, nous devons minimiser les erreurs :

$$\begin{cases} \epsilon_x = X_M - X_d = X_M \\ \epsilon_y = Y_M - Y_d = X_M \\ \epsilon_\theta = \theta - \theta_d = \theta \end{cases}$$

Si, pour simplifier l'écriture, nous prenons $X_d = Y_d = \theta_d = 0$.

Le vecteur d'erreur devient donc le vecteur de configuration courante du robot. C'est cet état que doit observer le RNF pour le commander. L'architecture de commande est alors la suivante :

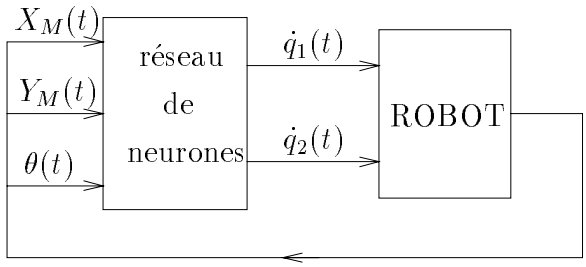

Fig. 7.1: *Structure du contrôle en position et en orientation*

7.1 Formulation Mathématique

D'un point de vue géométrique, pour rejoindre la configuration nulle, l'apprentissage doit minimiser le critère :

$$J = \alpha_1 X_M^2 + \alpha_2 Y_M^2 + \alpha_3 \theta^2 \tag{7.1}$$

où α_1, α_2, α_3 sont des coefficients positifs de normalisation (voir paragraphe 7.2.1).

D'un point de vue robotique, le réseau de neurones doit, à l'instant (t), à partir de la configuration observée, calculer les consignes pour qu'à l'instant $(t+1)$, le robot se rapproche de la configuration finale.
Le critère devient donc :

$$J = \alpha_1 X^2(t+1) + \alpha_2 Y^2(t+1) + \alpha_3 \theta^2(t+1) \tag{7.2}$$

(l'indice M est volontairement oublié dans un souci de clarté)
Pour minimiser ce critère, nous devons calculer son gradient par rapport aux sorties du réseau $\dot{q}_1(t)$ et $\dot{q}_2(t)$, c'est à dire les expressions :

$$\frac{\partial X(t+1)}{\partial \dot{q}_i(t)}, \quad \frac{\partial Y(t+1)}{\partial \dot{q}_i(t)}, \quad \frac{\partial \theta(t+1)}{\partial \dot{q}_i(t)}.$$

Or, ces expressions ne sont pas analytiquement calculables. On peut, toutefois, contourner cette difficulté en calculant un gradient approché à partir d'un modèle d'intégration. En effet, à l'instant (t), on a ([*Dela 91*]) :

$$\left\{ \begin{array}{lcl} \Delta\theta(t+1) & = & \frac{r}{2R}(\dot{q}_1(t) - \dot{q}_2(t))\Delta t \\ \Delta U(t+1) & = & \frac{r}{2}(\dot{q}_1(t) + \dot{q}_2(t))\Delta t \\ \theta(t+1) & = & \theta(t) + \Delta\theta(t+1) \\ X(t+1) & = & X(t) + \Delta U(t+1) cos(\theta(t) + \frac{\Delta\theta(t+1)}{2}) \quad = \quad X(t) + \Delta X(t+1) \\ Y(t+1) & = & Y(t) + \Delta U(t+1) sin(\theta(t) + \frac{\Delta\theta(t+1)}{2}) \quad = \quad Y(t) + \Delta Y(t+1) \end{array} \right.$$

Avec

- Δt la période d'échantillonnage
- $\theta(t+1), X(t+1), Y(t+1)$ la configuration du robot mesurée après application des consignes.
- $\Delta\theta(t+1)$ et $\Delta U(t+1)$ les variations de l'orientation du robot et la distance parcourue par le point M, entre les instants (t) et $(t+1)$.

L'équation 7.1 devient alors :

$$J = \alpha_1 (X(t) + \Delta X(t+1))^2 + \alpha_2 (Y(t) + \Delta Y(t+1))^2 + \alpha_3 (\theta(t) + \Delta\theta(t+1))^2 \tag{7.3}$$

Et on obtient :

$$\frac{\partial J}{\partial \dot{q}_i(t)} = 2\alpha_1 X(t+1) \frac{\partial \Delta X(t+1)}{\partial \dot{q}_i(t)} + 2\alpha_2 Y(t+1) \frac{\partial \Delta Y(t+1)}{\partial \dot{q}_i(t)} + 2\alpha_3 \theta(t+1) \frac{\partial \Delta\theta(t+1)}{\partial \dot{q}_i(t)} \tag{7.4}$$

avec:

$$\begin{cases} \frac{\partial \Delta X(t+1)}{\partial \dot{q}_i(t)} & = & \Delta t \frac{r}{2} cos(\theta(t) + \frac{\Delta\theta(t+1)}{2}) - \frac{1}{2}\frac{\partial \Delta\theta(t+1)}{\partial \dot{q}_i(t)} \Delta U(t+1) sin(\theta(t) + \frac{\Delta\theta(t+1)}{2}) \\ \frac{\partial \Delta Y(t+1)}{\partial \dot{q}_i(t)} & = & \Delta t \frac{r}{2} sin(\theta(t) + \frac{\Delta\theta(t+1)}{2}) + \frac{1}{2}\frac{\partial \Delta\theta(t+1)}{\partial \dot{q}_i(t)} \Delta U(t+1) cos(\theta(t) + \frac{\Delta\theta(t+1)}{2}) \\ \frac{\partial \Delta\theta(t+1)}{\partial \Delta q_i(t)} & = & \frac{r}{2R}\Delta t \ si \ i = 1; \ -\frac{r}{2R}\Delta t \ si \ i = 2. \end{cases} \quad (7.5)$$

Les contraintes cinématiques du robot apparaissent alors dans le gradient.

Remarque

Il est clair que cette approche est délicate à manipuler. En effet, l'apprentissage de ce critère avec de grandes erreurs $X_M(t), Y_M(t), \theta(t)$ risque de provoquer une saturation de la sortie de réseau. C'est à dire que les valeurs de sortie vont devenir maximales (proche de ± 1). Or la rétropropagation multiplie le gradient ci-dessus par la dérivée de la sigmoïde au point d'activité du neurone. Un neurone saturé a donc une "dérivée nulle". La décroissance du critère au cours de l'apprentissage va donc ralentir jusqu'à tendre vers zéro. Cependant, une sortie maximale (donc un déplacement à vitesse maximale), pour de grandes erreurs correspond à "une bonne réponse du réseau" sous condition de décroissance du critère. La non modification du réseau du fait de la rétropropagation d'un gradient non nul multiplié par une dérivée nulle ne doit donc pas altéré le résultat de l'apprentissage.

7.2 Apprentissage

7.2.1 Architecture

Les entrées du réseau (figure 7.2) sont normalisées entre -1 et $+1$ par rapport à leur valeur maximale :

$$\begin{cases} X_M(t) & \rightarrow & X_{norm} & = & \frac{X}{\alpha_1} = \frac{X}{D} \\ Y_M(t) & \rightarrow & Y_{norm} & = & \frac{Y}{\alpha_2} = \frac{Y}{D} \\ \theta(t) & \rightarrow & \theta_{norm} & = & \frac{\theta}{\alpha_3} = \frac{\theta}{\pi} \end{cases}$$

Où D est le demi-coté d'un carré centré sur l'origine.
Pour travailler à l'échelle de la salle de manipulation, nous prendrons $D = 4$ mètre.

7.2.2 Base d'Apprentissage

Des exemples aléatoires

Dans un premier temps, nous avons construit une base d'exemples aléatoires. Il s'avère que cette méthode n'est pas la meilleure. En effet, en "tirant au hasard" un nombre fixé

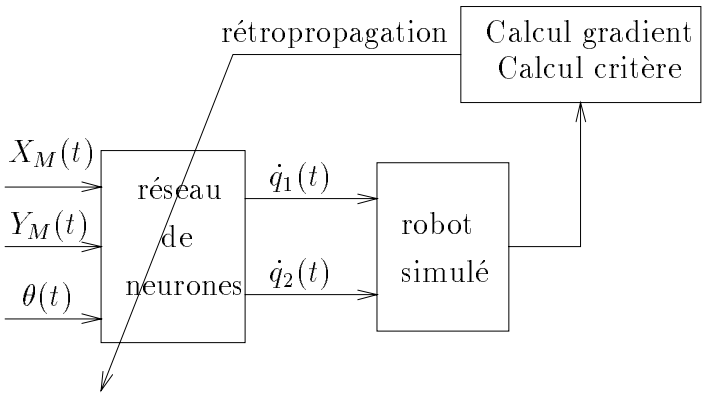

Fig. 7.2: *Apprentissage Hors-ligne*

d'exemples, nous ne savons pas si nous remplissons tout l'espace d'entrée du réseau. C'est à dire que nous ne savons pas si le réseau "voit" toutes les configurations possibles du robot. Pour en être certain, il faudrait un très grand nombre d'exemples, ce qui nous pénalise sur le temps d'apprentissage. Nous verrons de plus au chapitre 9 que ce problème sur la méthode de construction de la base d'apprentissage est très important. On s'aperçoit en effet, qu'une base d'apprentissage totalement aléatoire fait apparaître des configurations du robot qui n'ont aucun sens physique. Le réseau ne peut alors pas trouver de solution cohérente entre les "bons" et les "mauvais" exemples. Nous proposons donc de construire la base d'apprentissage par une répartition uniforme des exemples dans l'espace des configurations.

Des exemples uniformément répartis

Les erreurs sont normalisées par rapport aux distances maximales entre -1 et $+1$. Nous fixons un pas de répartition qui détermine le nombre d'exemples. Ainsi pour un pas de $0,25$, nous avons 9 états possibles de -1 à $+1$, ce qui donne $9 \times 9 \times 9 = 729$ exemples. Compte tenu de la taille du réseau[1] (45 connections), cela fait environ 16 exemples par poids. Ceci peut paraître important, mais il faut se rappeler que l'entrée est normalisée en distance maximale. Un pas de $0,25$ sur les entrées normalisées X_M et Y_M du réseau correspond donc à un "échantillonnage" de l'espace cartésien par un incrément de 1 m. C'est à dire qu'**il n'y aura pas d'exemple tel que l'erreur soit inférieure à** 1 m**, sauf l'erreur nulle**. Comme nous souhaitons une précision bien inférieure à 1 m (de l'ordre du cm), nous misons sur la capacité de généralisation de l'apprentissage.

7.2.3 Résultats

La figure 7.3 montre l'évolution du critère et de son gradient pour les 500 premières itérations de l'apprentissage. Le critère tend vers une asymptote de l'ordre de $0,4$ alors que sa valeur initiale est de $0,42$. Sa variation est donc très faible.

[1] Les meilleurs résultats ont été obtenu par un réseau de type $3-9-2$ (une couche cachée de 9 neurones).

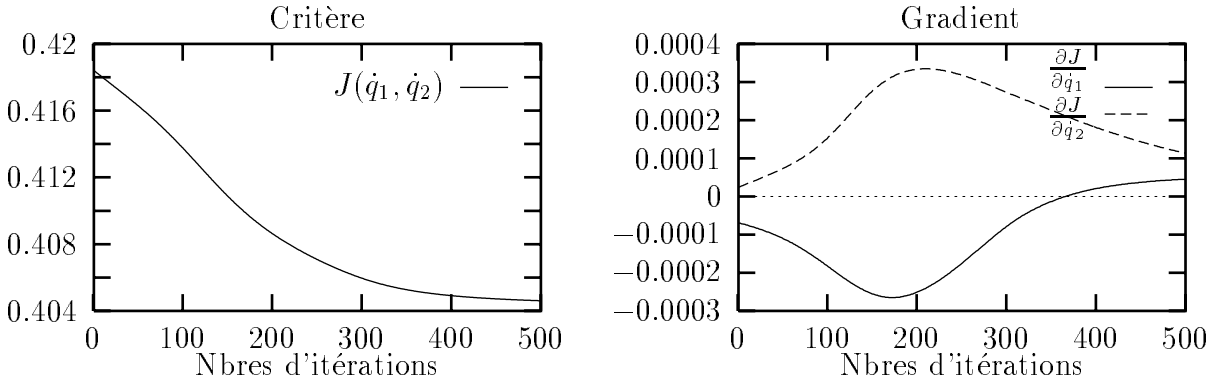

Fig. 7.3: *Evolution de l'apprentissage*

De même, l'amplitude du gradient est Cependant, si on compare cette évolution avec celle du gradient en sortie de réseau (même figure), on s'aperçoit que l'apprentissage ne permet pas d'atteindre un minimum puisque le gradient n'est pas nul. C'est le phénomène de saturation décrit à la fin de la section 7.1.

7.2.4 Valeur des poids du réseau

Les poids du réseau (voir annexe A.2) ont une valeurs assez proche de 1. En moyenne, ils sont plus faibles que ceux obtenus par l'apprentissage Direct.

7.3 Simulation de la Commande

La figure 7.4 montre les erreurs cartésiennes et angulaires à partir de la configuration initiale $X_M = 4m, Y_M = 3m, \theta = 180^o$. Les erreurs sur X_M et θ sont bien annulées, mais l'erreur sur

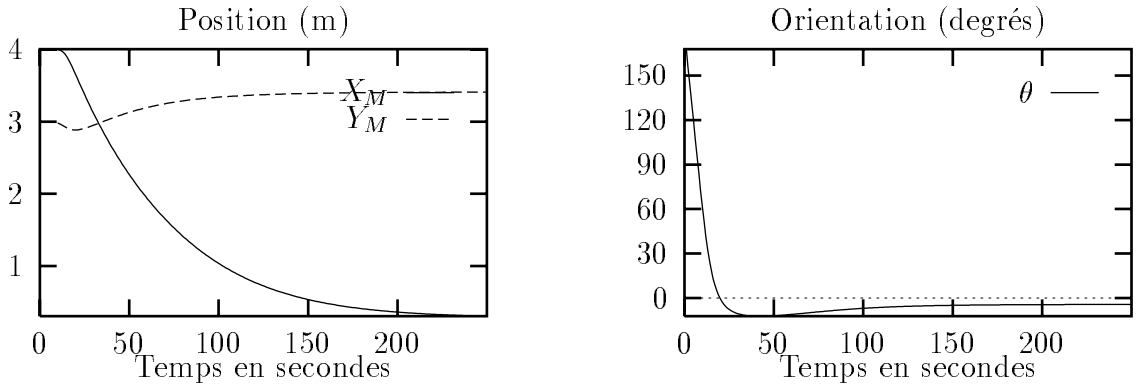

Fig. 7.4: *Evolution des Erreurs*

Y_M n'est pas commandée. Ceci est confirmé si la configuration initiale est telle que $X_M = 0m$ et $\theta = 0^o$. Les consignes en vitesse calculées par le réseau sont alors nulles, le robot ne bouge pas. Ce résultat n'est pas surprenant car, lorsque $\theta = 0^o$, le gradient du critère est quasiment

nul sur l'axe $X = 0$ puisqu'il ne dépend alors que de $sin\Delta\theta(t+1)$ (équation 7.5).
Pour tenir compte de ce problème, nous proposons de biaiser le retour d'état.

7.4 Amélioration par modification du retour d'état angulaire

L'idée de base est d'utiliser le réseau précédent et de modifier le retour d'état. On constate en effet que le RNF contrôle en priorité l'orientation du robot. En remplaçant l'erreur angulaire $\epsilon_\theta = \theta$ par $\epsilon_\theta = \theta - \theta_s$ où θ_s est une une contrainte de stratégie fonction de la position cartésienne, nous pouvons obtenir un contrôle du robot tel qu'il se dirige vers la configuration finale en exécutant une manoeuvre de type "créneau".

Il faut donc que θ_d soit nul si Y_M est nul, et proche de $\pm\frac{\pi}{2}$ si Y_M est grand. θ_d peut par exemple suivre la pente de la fonction $u = v^n$, c'est à dire $\theta_s = arctg(nv^{n-1})$.
Une fonction simple pour θ_s peut être la pente de la fonction Y_M^2, c'est à dire :

$$\theta_s = arctg(2\frac{Y_M}{D}).$$

La figure 7.5 montre les simulations correspondantes pour une configuration initiale identique à celle de la figure 7.4. L'amélioration est nette puisque la position finale est $X_M = 5cm$, $Y_M = -3cm$, $\theta = -3.5^o$. Cependant, sur la figure 7.6, pour une configuration initiale proche de l'axe singulier $X_M = 0$, c'est à dire pour $-0,5m < X_M < 0,5m$, le contrôle est difficile et la position finale peut satisfaisante.

Il y a deux explications possibles :

- le réseau n'a pas pu apprendre à contrôler le robot lorsque $X_M = 0$ et $\theta = 0$ à cause d'un gradient nul pour ces configurations;
- c'est un problème d'erreur statique, donc de gain sur l'entrée. On peut penser qu'un gain plus fort améliorerait le résultat. Cependant, des tests ont montré que le réseau étant fortement non linéaire, cela est délicat à manipuler.

Il nous parait intéressant à ce stade, d'expérimenter le neuro-contrôleur obtenu.

7.5 Expérimentation de la Commande

Les résultats expérimentaux des figures 7.7 et 7.8, ont une allure *globalement* identique à ceux de la figure 7.5. Les effets transitoires sont plus importants. On aperçoit même des comportements du type second ordre amorti pour Y_M et θ. Une manœuvre de demi-tour est montrée figure 7.9.
Le comportement physique du robot, notamment sa faible bande passante, ne compromet donc pas la commande. Les erreurs statiques sont du même ordre qu'en simulation.

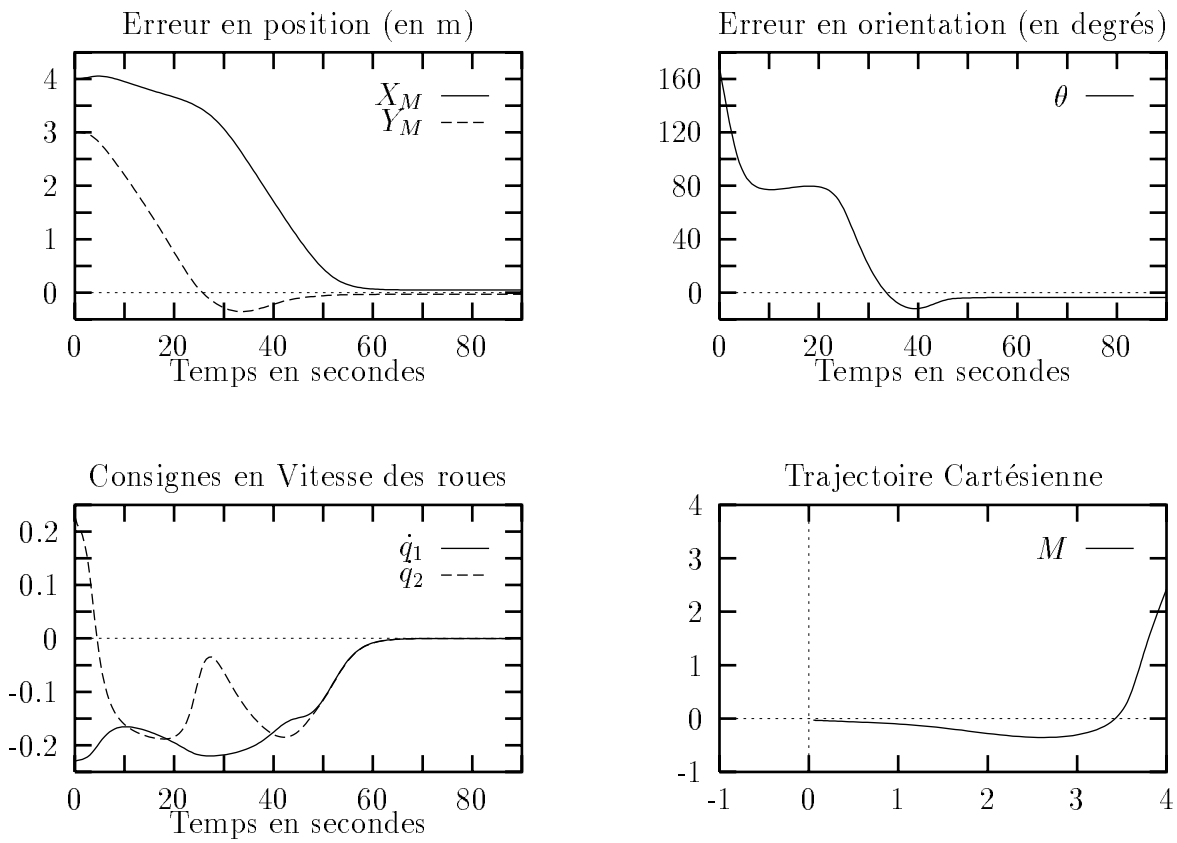

Fig. 7.5: *Simulation pour* $X_M = 4m$, $Y_M = 3m$, $\theta = 180^o$

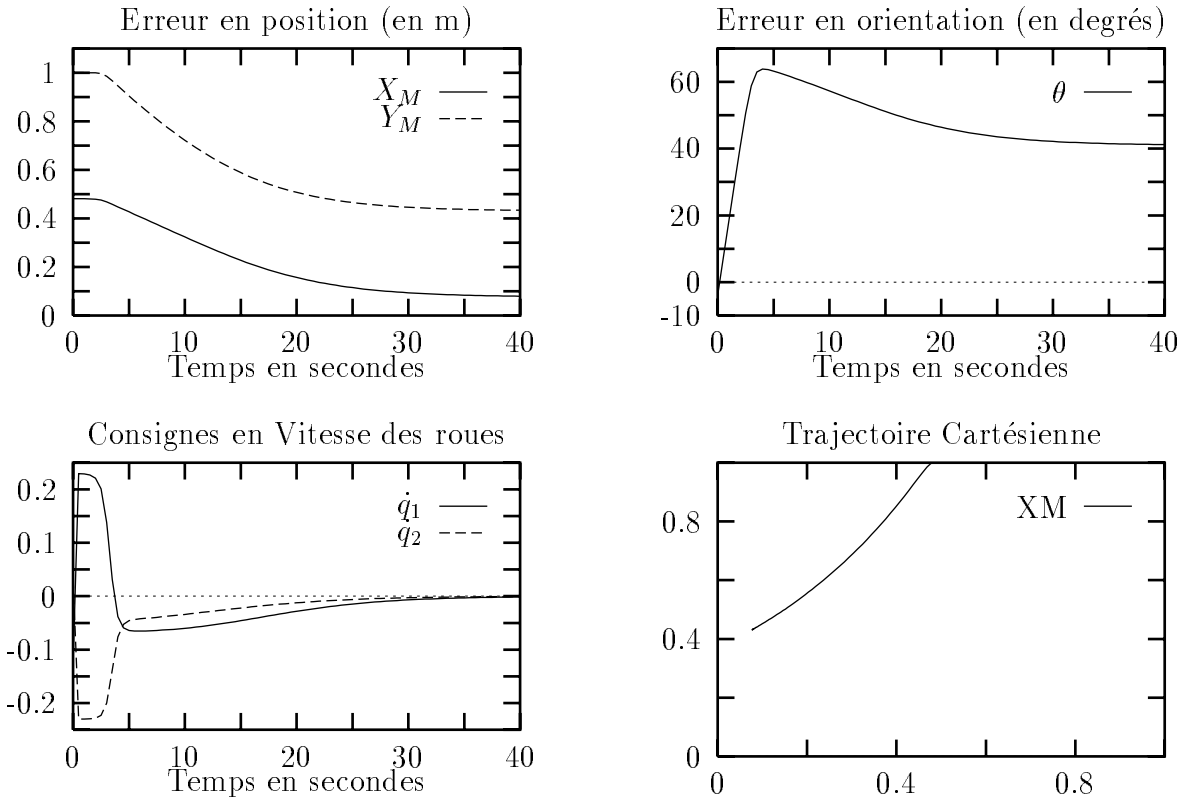

Fig. 7.6: *Simulation pour* $X_M = 0.5m$, $Y_M = 1m$, $\theta = 0^o$

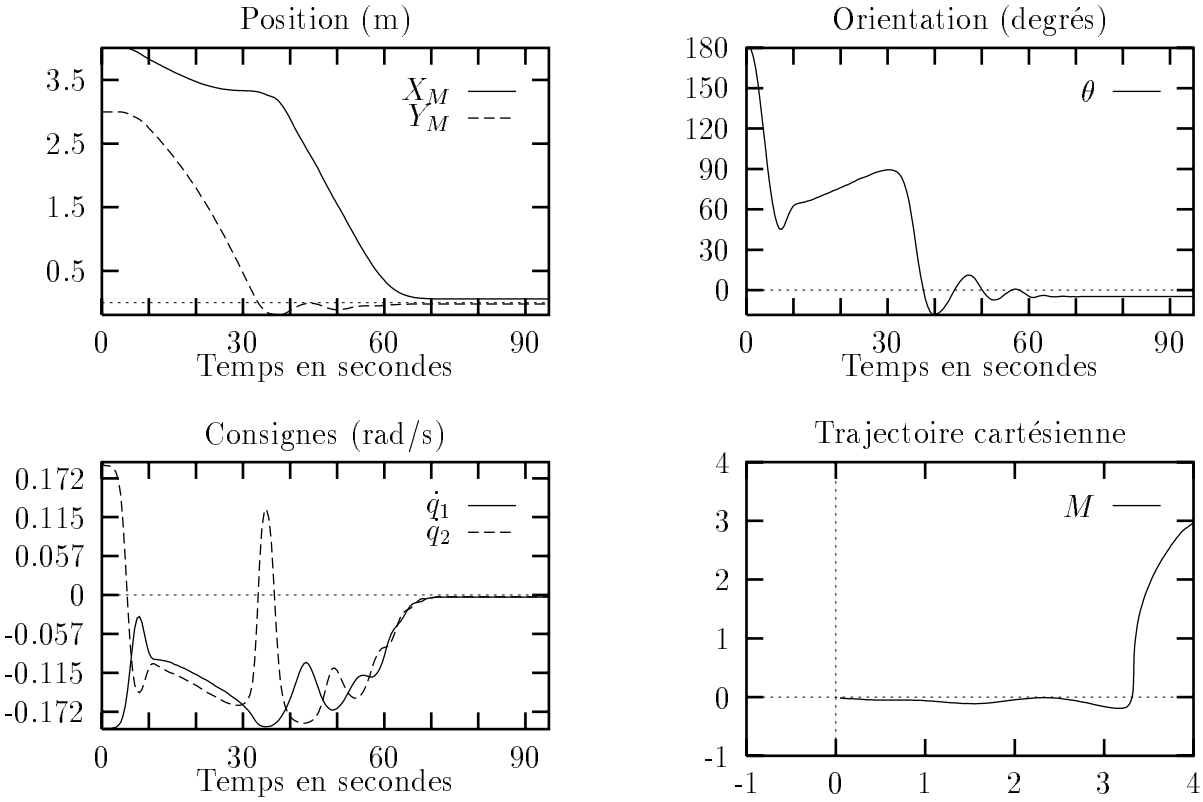

Fig. 7.7: *Expérimentation pour* $X_M = 4m$, $Y_M = 3m$, $\theta = 180^o$

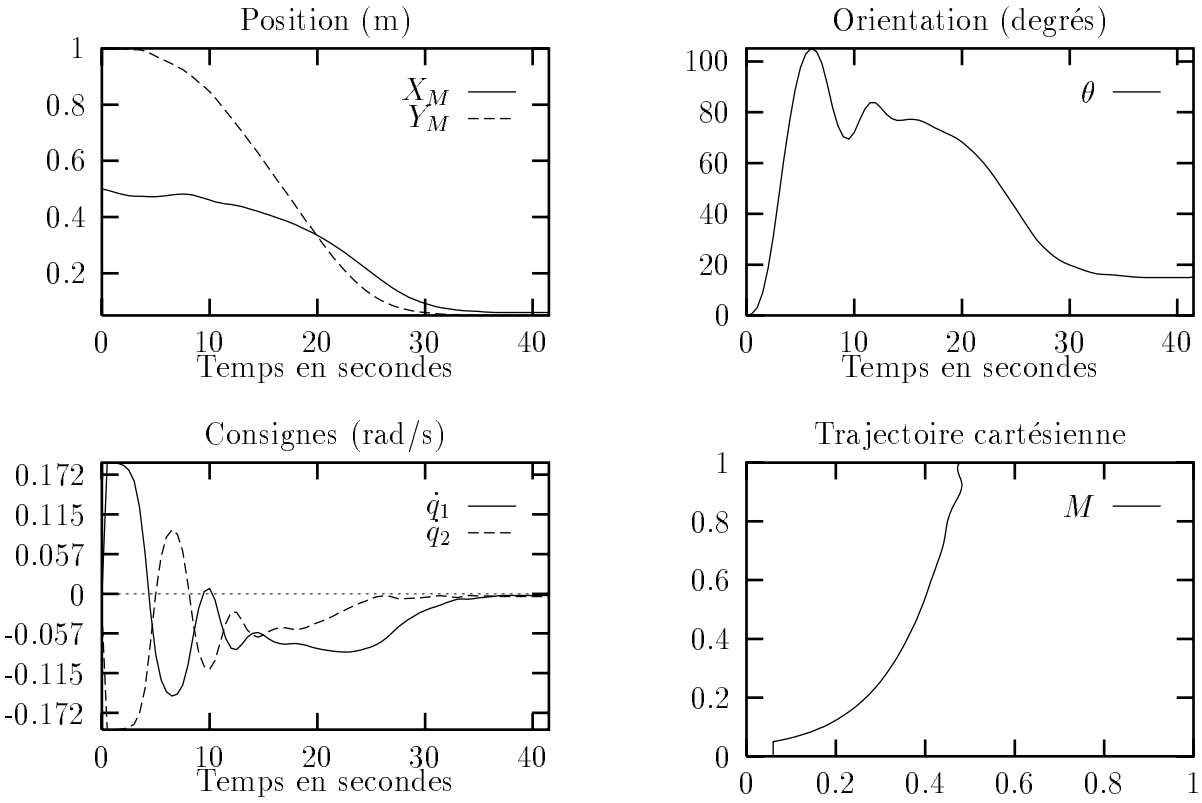

Fig. 7.8: *Expérimentation pour* $X_M = 0,5m$, $Y_M = 1m$, $\theta = 0^o$

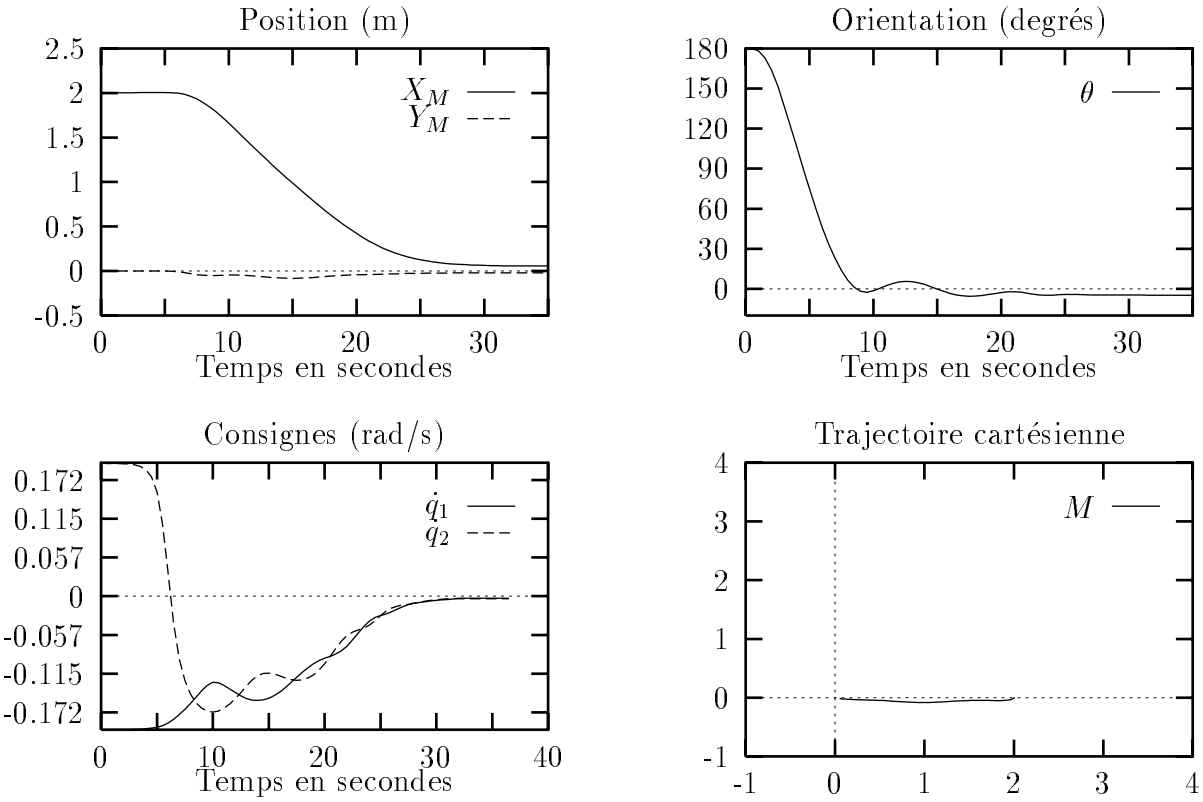

Fig. 7.9: *Expérimentation pour un demi tour*

7.6 Conclusion

Dans ce chapitre,

- nous avons montré par expérimentation que la rétropropagation d'un critère permet l'apprentissage de la commande d'un système sans sorties désirées.
- Le critère que nous avons utilisé exprime très simplement dans l'espace opérationnel le but à atteindre .
- En utilisant le modèle cinématique du robot, nous avons calculé le gradient du critère par rapport aux paramètres de commande, ce qui nous a permis d'introduire les contraintes non-holonomes du système.
- Nous n'avons pas utilisé de modèle inverse, et le neuro-contrôleur obtenu calcule directement les consignes des actionneurs.
- Après avoir modifié l'observation de la configuration courante du robot, nous avons validé expérimentalement la commande.

L'expérimentation nous a toutefois montré que l'apprentissage Hors-ligne ne prend en compte qu'un modèle approché du robot. Les relevés montrent en effet des phénomènes de réponses transitoires qui sont liés, sans aucun doute, à sa faible bande passante.
Le chapitre suivant présente les résultats sur l'apprentissage En-Ligne. On montre par l'expérience que cette technique permet d'adapter le réseau au comportement réel du robot.

Chapitre 8

Commande par Apprentissage Indirect En-Ligne

Nous présentons rapidement dans le premier paragraphe la structure d'apprentissage/commande. Dans la seconde, nous simulons l'apprentissage et la commande. Dans la troisième, nous expérimentons :

- le contrôleur obtenu par apprentissage sur simulateur,
- l'apprentissage sur le robot,
- le contrôleur obtenu par apprentissage sur le robot.

8.1 Architecture

Pour notre application, l'apprentissage En-Ligne est décrit par la figure 8.1. Le retour d'état

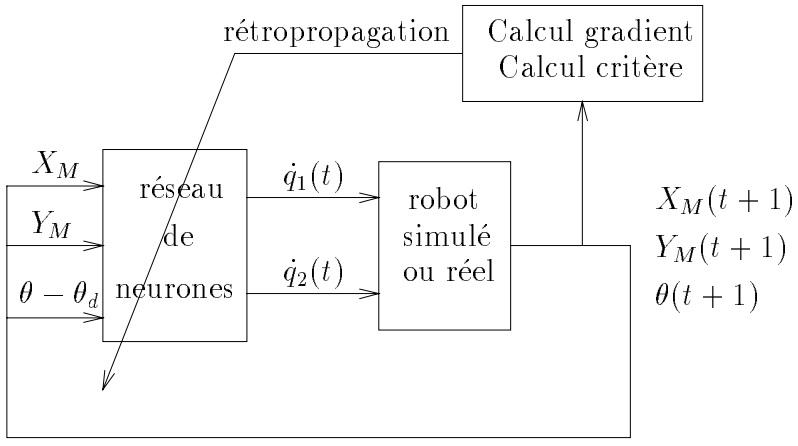

Fig. 8.1: *Structure d'apprentissage En-ligne de la commande en position de ROMEO*

tient compte des remarques du chapitre précédent. Le critère est donc :

$$J = \alpha_1 X^2(t+1) + \alpha_2 Y^2(t+1) + \alpha_3(\theta(t+1) - \theta_d)^2 \tag{8.1}$$

où

$$\theta_d = arctg(2Y_M(t)).$$

Le gradient est calculé comme au chapitre précédent par la formule :

$$\frac{\partial J}{\partial \dot{q}_i(t)} = 2\alpha_1 X(t+1)\frac{\partial \Delta X(t+1)}{\partial \dot{q}_i(t)} + 2\alpha_2 Y(t+1)\frac{\partial \Delta Y(t+1)}{\partial \dot{q}_i(t)} + 2\alpha_3(\theta(t+1) - \theta_d)\frac{\partial \Delta\theta(t+1)}{\partial \dot{q}_i(t)} \tag{8.2}$$

Nous rappelons que la mise à jour des poids est effectuée après chaque calcul de la sortie de la manière suivante (section 2.4) :

- En fonction du gradient du critère si celui-ci décroît.
- Aléatoirement avec une faible variance s'il croît.

Il est important de remarquer qu'il n'y a pas de base d'apprentissage en tant que telle. Chaque exemple est issu de la nouvelle configuration du robot elle même obtenue de la commande calculée par le réseau et appliquée au robot. C'est donc le réseau qui, à partir d'une configuration initiale du robot, génère lui même les exemples.

8.2 Simulation

Nous effectuons l'apprentissage en-ligne avec un réseau $3 - 9 - 2$ initialisé aléatoirement. Le robot est positionné à une configuration quelconque, si possible loin de la configuration finale désirée de manière à ce qu'il passe par un maximum de configurations différentes afin d'explorer le mieux possible l'espace des états.

Nous choisissons deux configurations initiales :

1. $X_M = 4m$, $Y_M = 4m$, $\theta = 180^o$
2. $X_M = 4m$, $Y_M = -4m$, $\theta = 180^o$.

Le protocole d'apprentissage est le suivant :

1. Premier apprentissage en-ligne à partir de la première configuration initiale jusqu'à arrêt complet du robot (consignes nulles),
2. Deuxième apprentissage en-ligne à partir de la même configuration initiale jusqu'à arrêt complet du robot (consignes nulles),
3. Troisième apprentissage en-ligne à partir de la seconde configuration initiale jusqu'à arrêt complet du robot (consignes nulles),

4. Quatrième apprentissage en-ligne à partir de la même configuration initiale jusqu'à arrêt complet du robot (consignes nulles),

Les figures 8.2 et 8.3 montrent l'évolution du critère, du gradient et du robot pendant les deux premières étapes. Ces courbes montrent la simultanéité entre la convergence du réseau

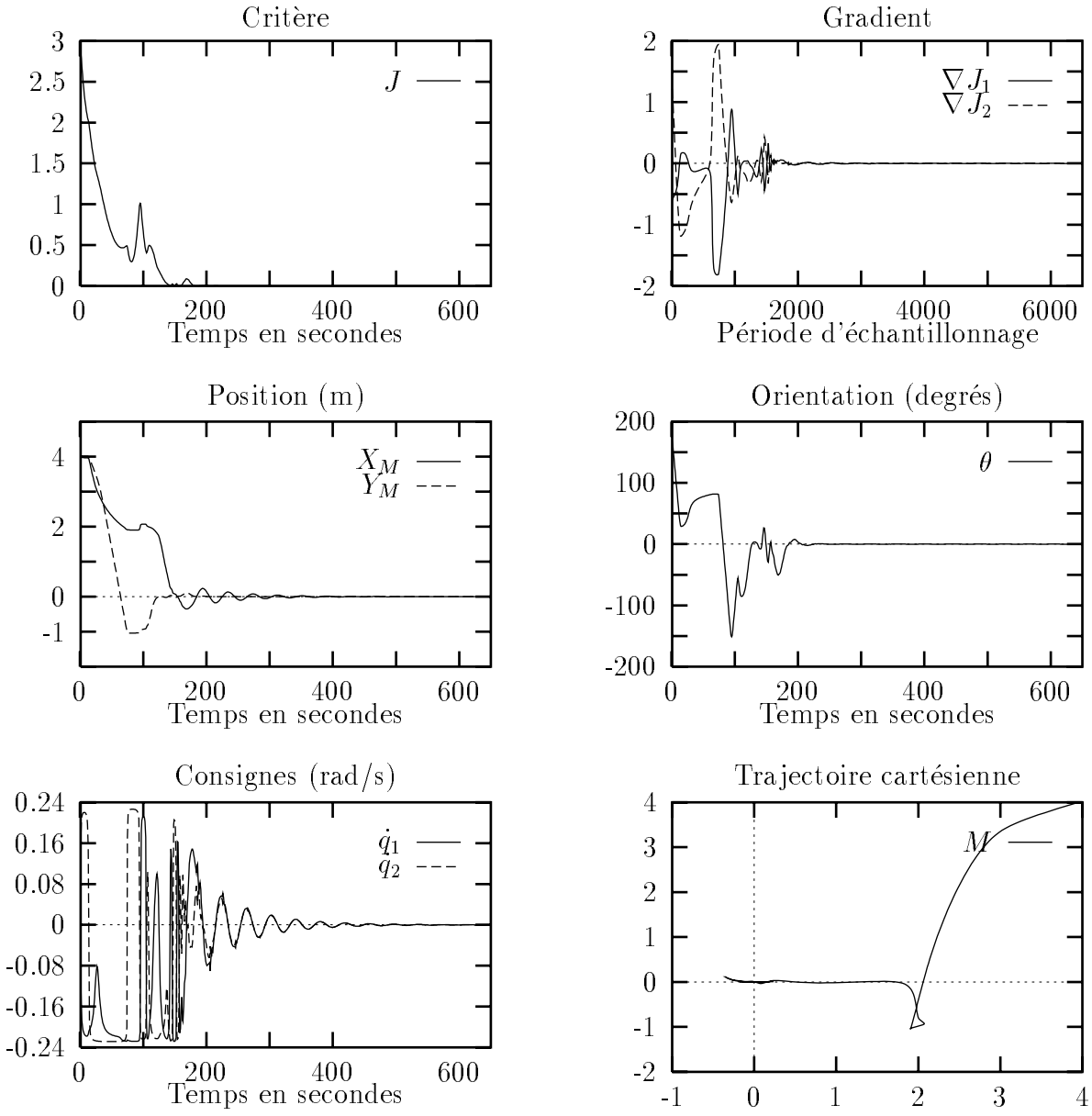

Fig. 8.2: *Premier apprentissage en-ligne sur simulateur*

vers le minimum global, et la convergence du robot vers l'état désiré. On aperçoit sur la figure 8.3 l'effet du premier apprentissage sur le second. Le critère et les erreurs décroissent plus vite. La trajectoire est plus courte et le robot atteint directement la configuration finale.

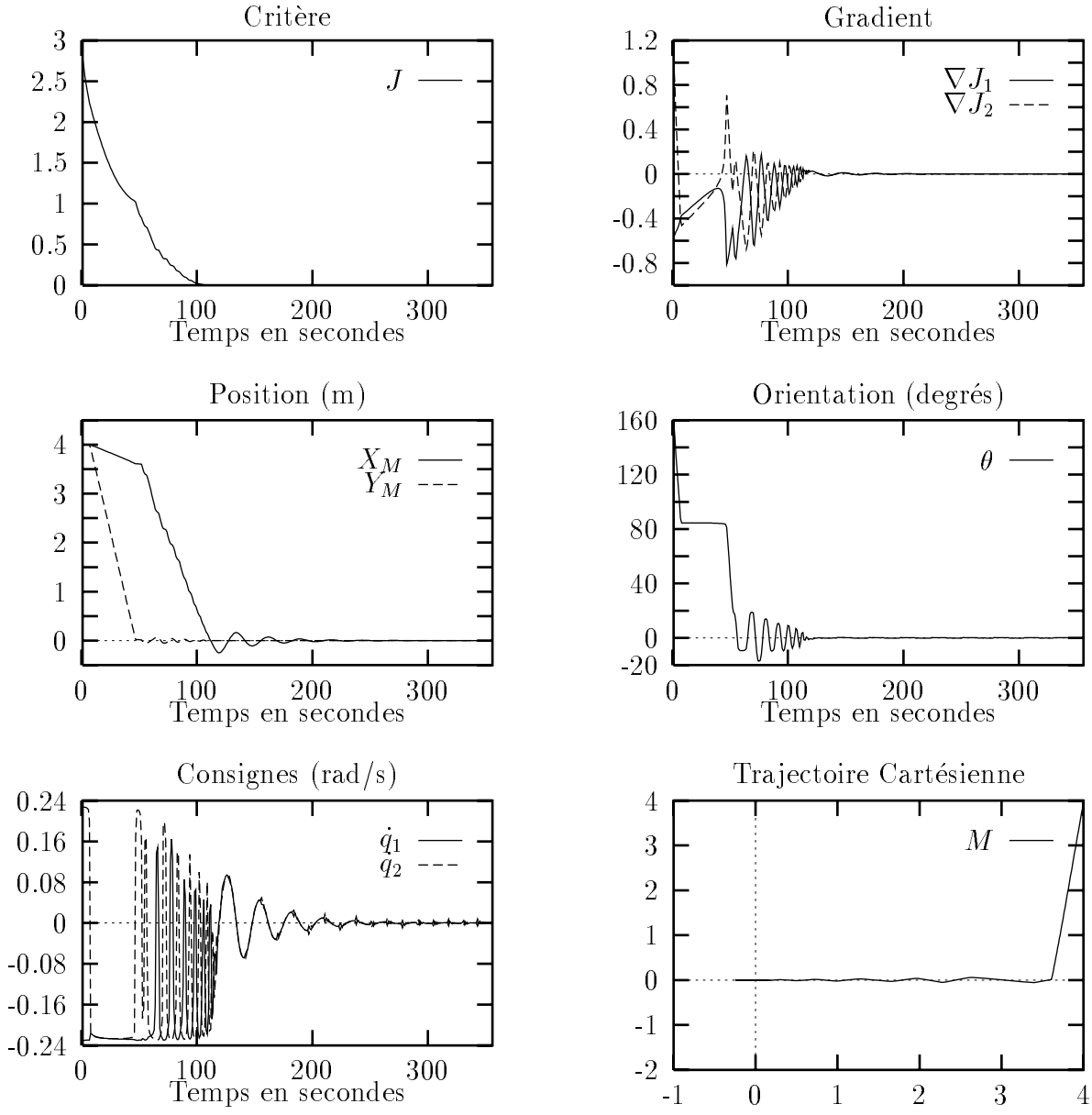

Fig. 8.3: *Second apprentissage en-ligne sur simulateur*

8.2.1 Valeur des poids du réseau après apprentissage en-ligne

Les valeurs des poids des couches cachées (annexe A.3) sont plus faibles que pour les autres méthodes d'apprentissage (rétropropagation directe et indirecte hors-ligne).

8.2.2 Influence des paramètres d'apprentissage

La figure ?? donne un exemple l'influence de la valeur du pas d'apprentissage λ sur la trajectoire du robot pendant le premier apprentissage en-ligne pour une valeur du paramètre de viscosité de 0,15. Plus λ tend vers 1, plus vite le robot atteind la configuration désirée, mais moins il explore son espace des configuration. Les valeurs de *lambda* (0,25) et de la viscosité (0) ont été choisies aprés quelques essais, car elles permettent un comportement acceptable du robot pour les expérimentations.

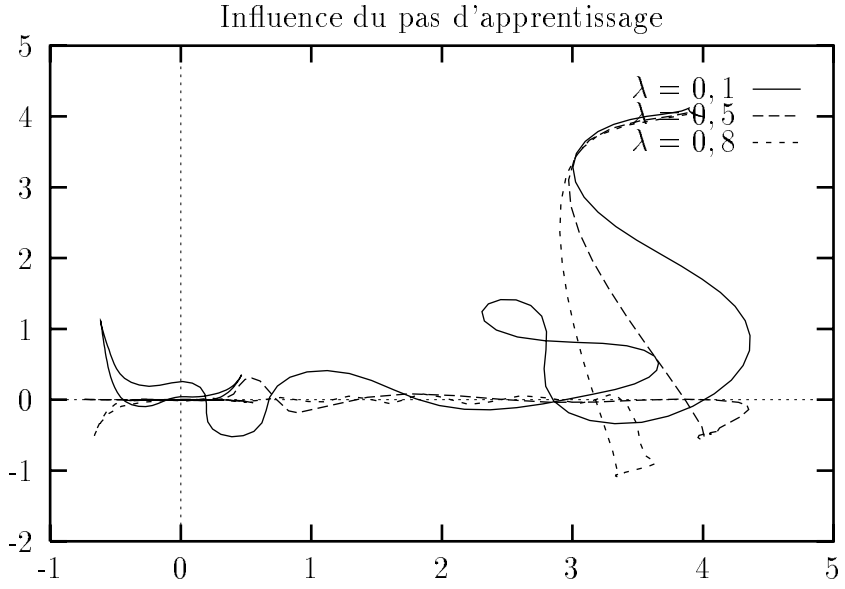

Fig. 8.4: *Influence du pas d'apprentissage sur la trajectoire du robot pendant l'apprentissage en-ligne*

8.2.3 Comportement du robot après apprentissage

On peut considérer que l'apprentissage en-ligne sur le simulateur a permis au réseau d'acquérir une expérience, donc d'apprendre à commander le robot. Ceci est confirmé par la figure 8.5. On a tracé quelques trajectoires obtenues pour un ensemble significatif de configurations. Elles convergent toutes vers la configuration désirée. Ce résultat est très satisfaisant car il prouve qu'à partir de seulement deux configurations initiales, la rétropapagation du critère en-ligne permet :

- le contrôle du robot avec un retour d'état simple (stationnaire).

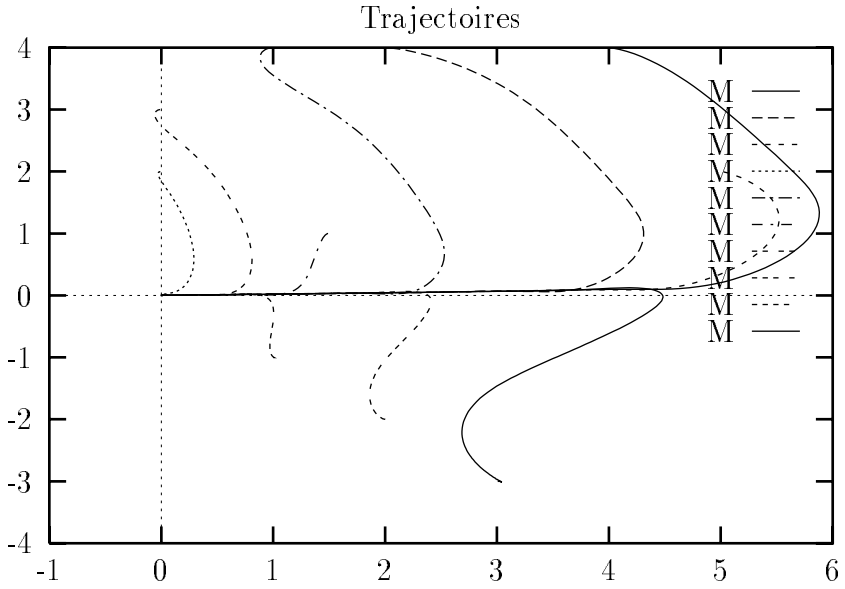

Fig. 8.5: *Trajectoires cartésiennes du robot après apprentissage sur simulateur*

- la généralisation du contrôle à tout l'espace des configurations.

Deplus, les erreurs finales sont très faibles (de l'ordre du millimètre). La figure 8.11 donne un aperçu des variations d'erreurs et des consignes pour la configuration initiale $X_M = 3m$, $Y_M = 3m$, $\theta = 180^o$.

En comparant ces courbes avec celles de la figure 8.2, on remarque que les consignes sont moins discontinues. Les oscillations sont donc un phénomène uniquement liées à l'apprentissage en-ligne, sûrement du fait des modifications permanentes du réseau.

Enfin, l'évolution de l'erreur en X montre que le réseau minimise d'abord Y et θ au détriment de X. Le robot commence par s'éloigner du but pour se rapprocher de l'axe X. On constate figure 8.5 que cet éloignement croît avec l'abcisse initiale. On peut éliminer ce problème en "dilatant" l'espace d'entrée, par un gain inférieur à 1 sur l'erreur X. C'est un inconvénient du réseau obtenu par l'apprentissage en-ligne.

8.3 Expérimentations

8.3.1 Commande par un réseau formé sur simulateur

En comparant la figure 8.7 à la précédente (8.11), on constate que le robot s'éloigne moins du but, car les variations des consignes sont lissées par sa faible bande passante. L'erreur est du même ordre de grandeur (10^{-3}), mais des oscillations amorties sur les consignes apparaissent.

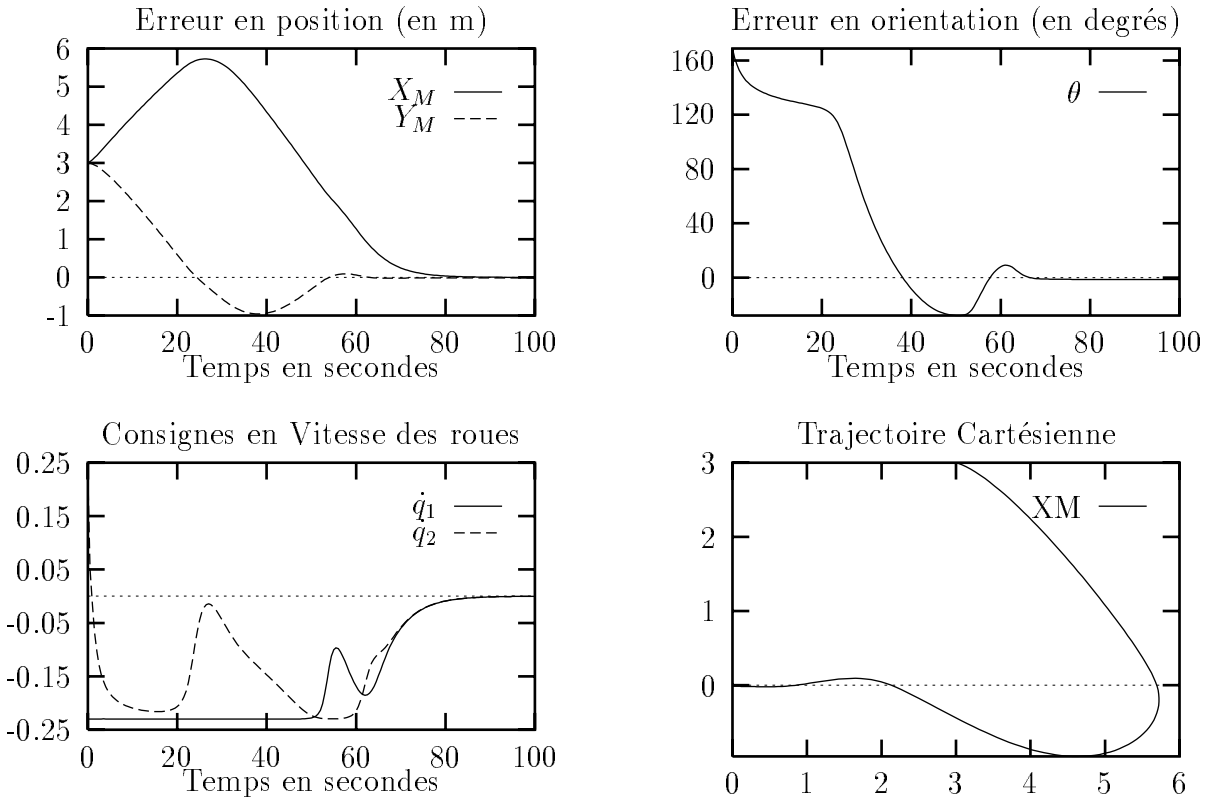

Fig. 8.6: *Simulation après apprentissage sur simulateur*

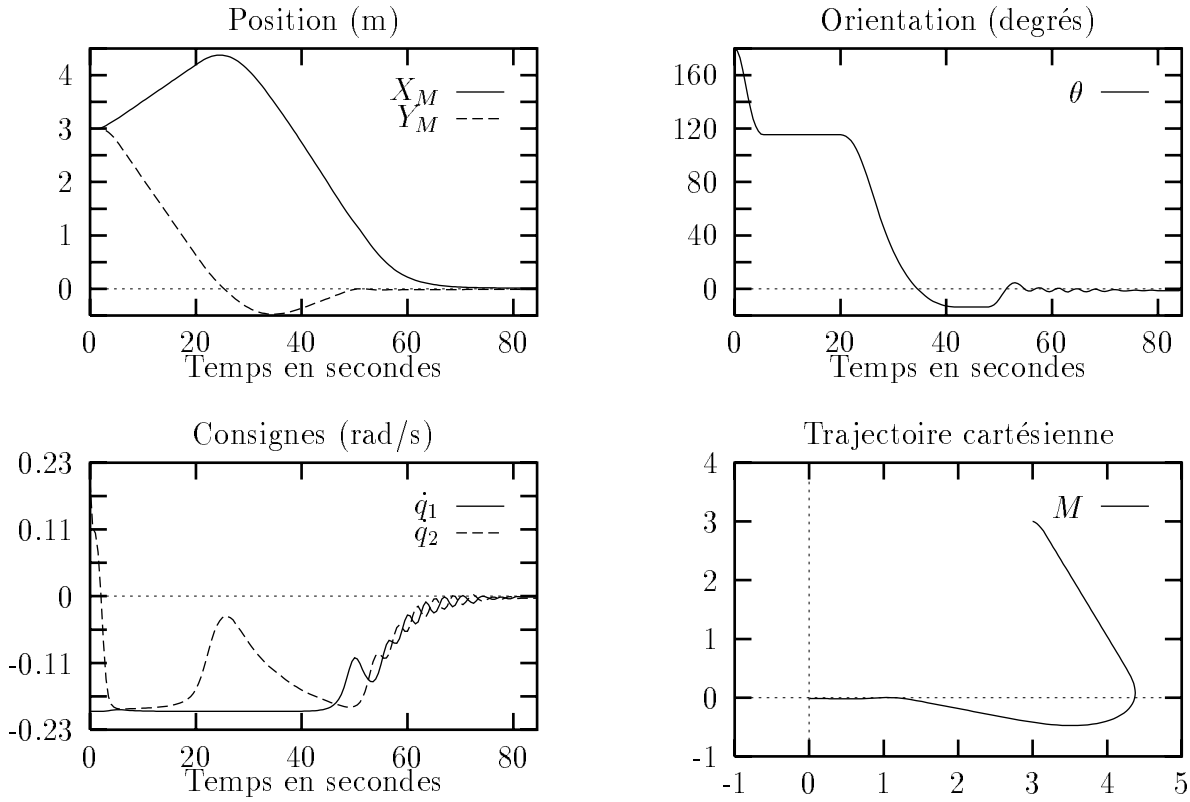

Fig. 8.7: *Expérimentation après apprentissage sur simulateur*

8.3.2 Apprentissage en-ligne sur le robot réel

Pour des problèmes d'espace disponible dans la salle d'expérimentation, nous choisissons deux configurations initiales différentes de la simulation :

1. $X_M = 3m$, $Y_M = 3m$, $\theta = 0^o$
2. $X_M = 3m$, $Y_M = 0m$, $\theta = 180^o$.

La première est utilisée pour deux apprentissages successifs (figures 8.8 et 8.9) jusqu'à arrêt complet du robot. La seconde est utilisée pour l'apprentissage suivant (figure 8.10) jusqu'à arrêt complet du robot.
Nous utilisons ici un réseau 3-9-2 initialisé aléatoirement. Le système que nous contrôlons est en effet lent et sa bande passante très faible. Il va donc filtrer les discontinuités des consignes apparues figure 8.2. Nous nous trouvons ainsi dans le cas le plus défavorable pour l'apprentissage.

Les trois étapes de l'apprentissage sont montrées figures 8.8, 8.9 et 8.10.
On remarque pour la première étape une plus grande difficulté de convergence qu'en simulation. La réalité physique du système est donc prise en compte par l'apprentissage. Cependant, la trajectoire du robot n'est plus la même qu'en simulation et le robot se retrouve dans une configuration difficile. L'apprentissage lui permet toutefois de converger vers le but par une succession de manœuvres.

La deuxième phase d'apprentissage est plus rapide que la première, malgré un comportement similaire[1] avec des amplitudes plus faibles. Il y a donc acquisition d'une connaissance sur le contrôle du robot réel.

La troisième phase d'apprentissage est instable mais non divergente. Le robot est à la position désirée, mais l'orientation n'est pas stabilisée. Nous avons dû stopper l'apprentissage. La section suivante montre toutefois que le réseau obtenu permet de contrôler le robot. Plusieurs phénomènes peuvent expliquer cette oscillation :

- du coté de la rétropropagation :
  - Le pas d'apprentissage est trop grand. On peut le diminuer et recommencer l'apprentissage, ou l'adapter en fonction des variations du critère.
  - La pondération α_3 du critère n'est pas assez faible.
- du coté de la propagation ou de la commande :
  - C'est un problème de frottements secs sur les moteurs.
  - C'est un problème de "pompage" lié aux saturations des consignes sur les moteurs.
  - C'est un problème de "pompage" lié à la saturation de la sigmoïde sur la couche de sortie.

[1] L'allure de la trajectoire est tout à fait identique à celui obtenu par [*Ait 93*] avec la commande continue instationnaire

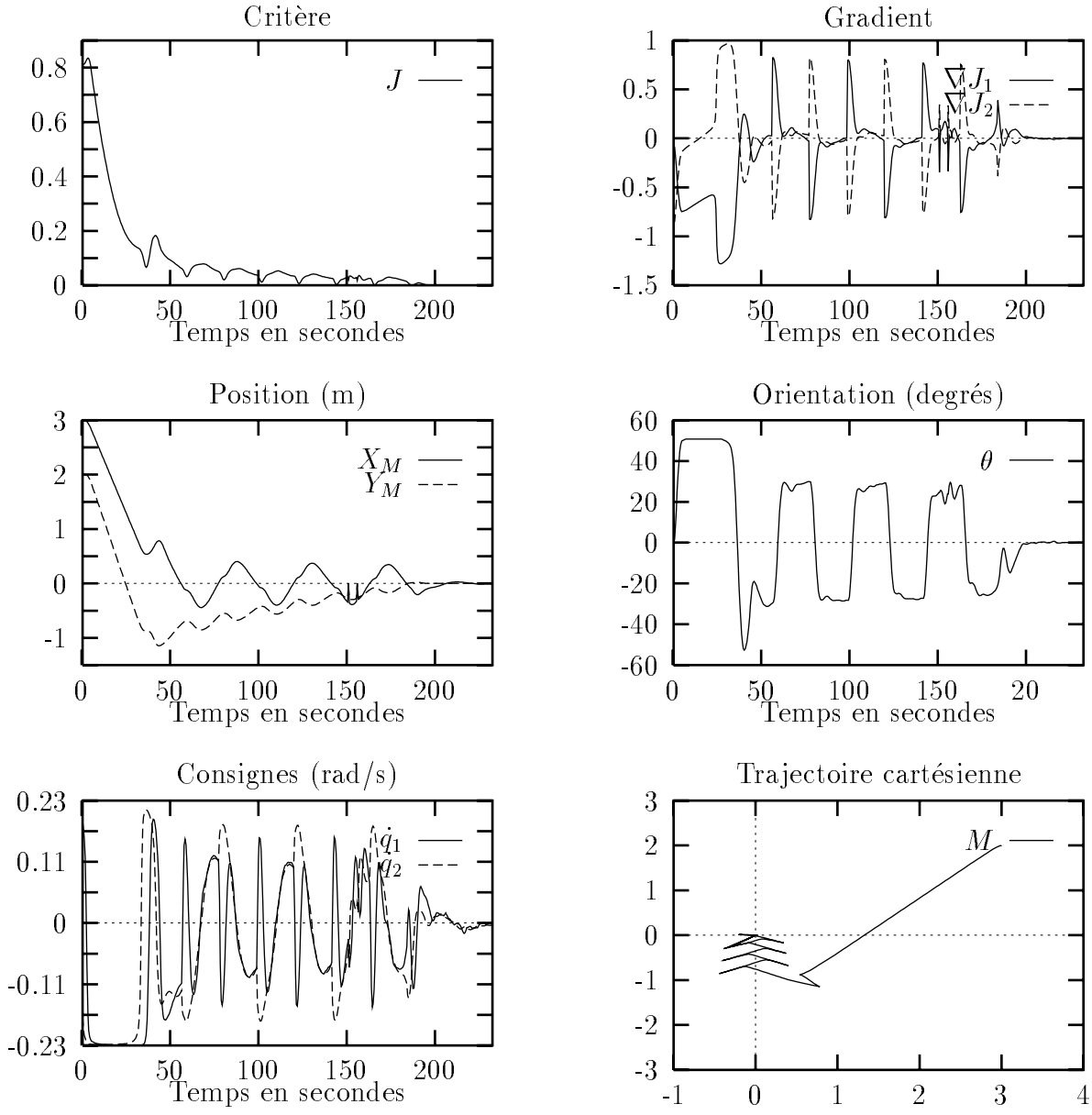

Fig. 8.8: *Premier apprentissage réel en-ligne*

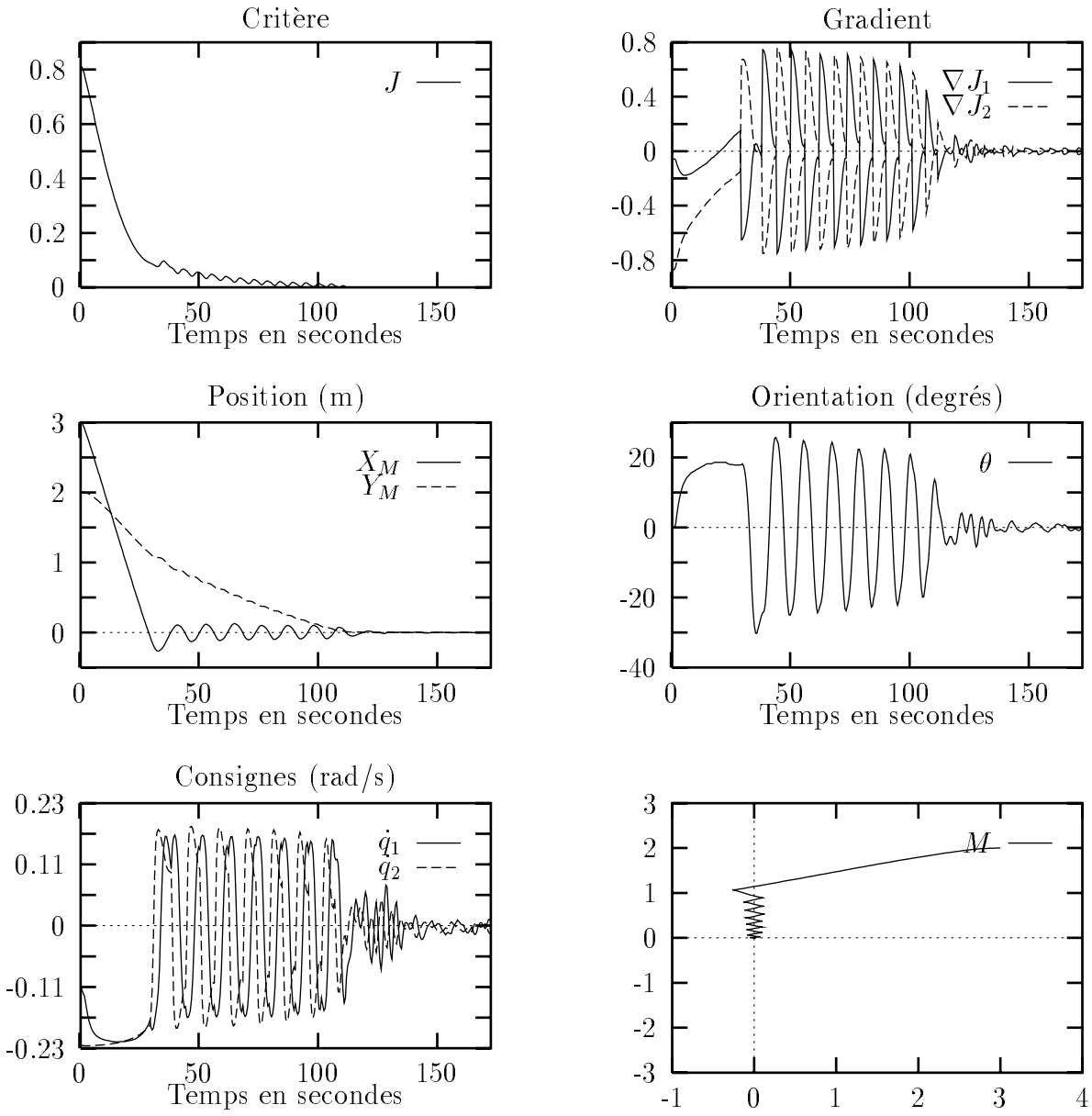

Fig. 8.9: *Second apprentissage réel en-ligne*

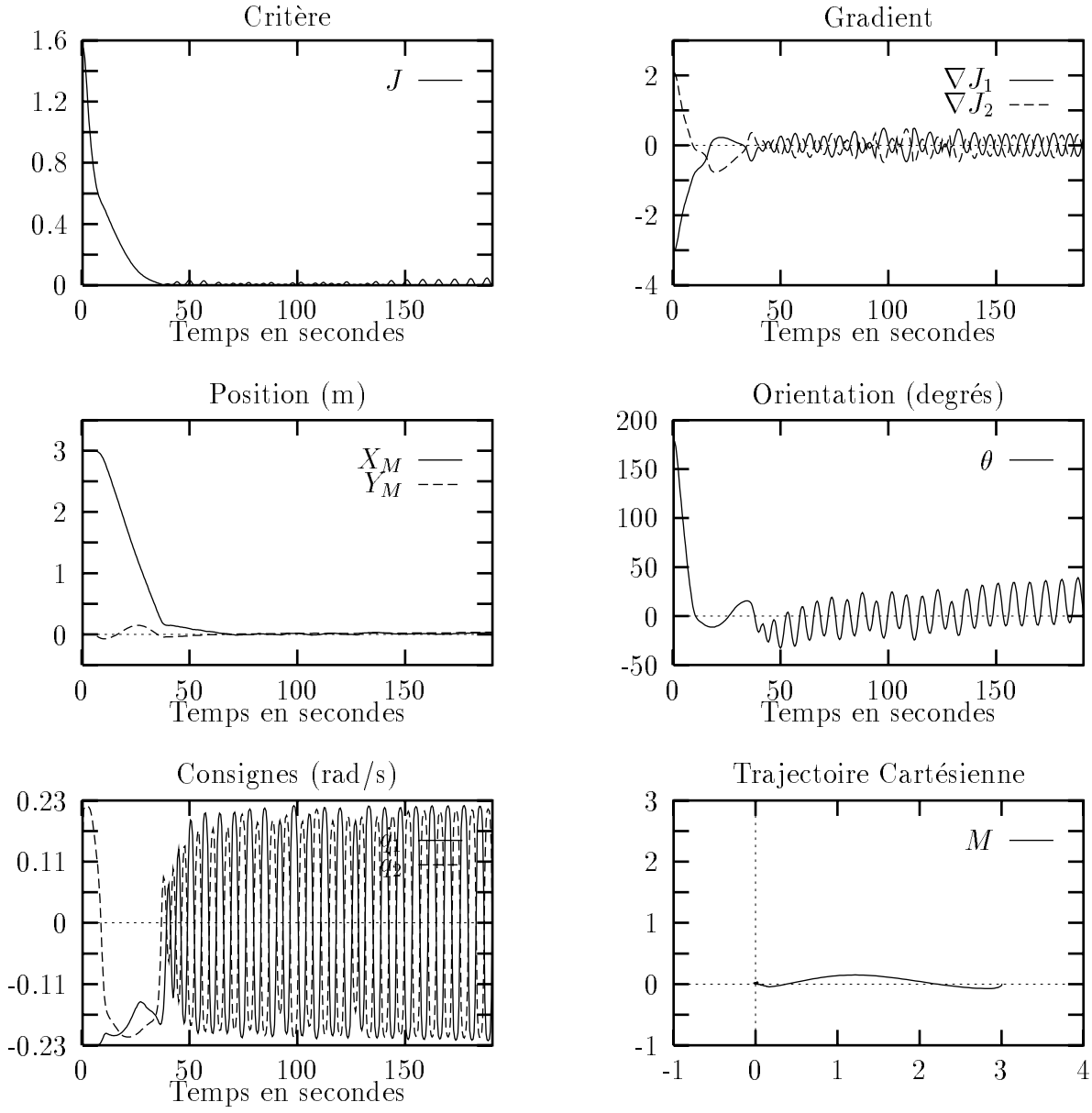

Fig. 8.10: *Troisième apprentissage réel en-ligne*

8.3.3 Valeur des poids

On ne peut pas conclure aisément en observant l'évolution des poids (annexe A.4.1) après chaque apprentissage. Cependant, on peut remarquer qu'ils sont plus faibles que ceux obtenus par rétropropagation directe. D'autres part, on remarque que le neurone 7 de la couche cachée semble plus actif que les autres. En effet, les poids de ses connexions qui le relient aux deux neurones de la couche de sortie (w23(7,2) et w23(7,1)) sont plus important que les poids qui le relient aux autres neurones.

8.3.4 Commande après apprentissage sur le robot réel

Le réseau obtenu par les trois précédentes phases d'apprentissage est utilisé sans adaptation. La configuration initiale du robot est celle de la troisième phase. La trajectoire n'est pas la même que pendant l'apprentissage. Le réseau évite la configuration singulière $X_M = 0$. La commande est stable, mais une erreur statique subsiste.

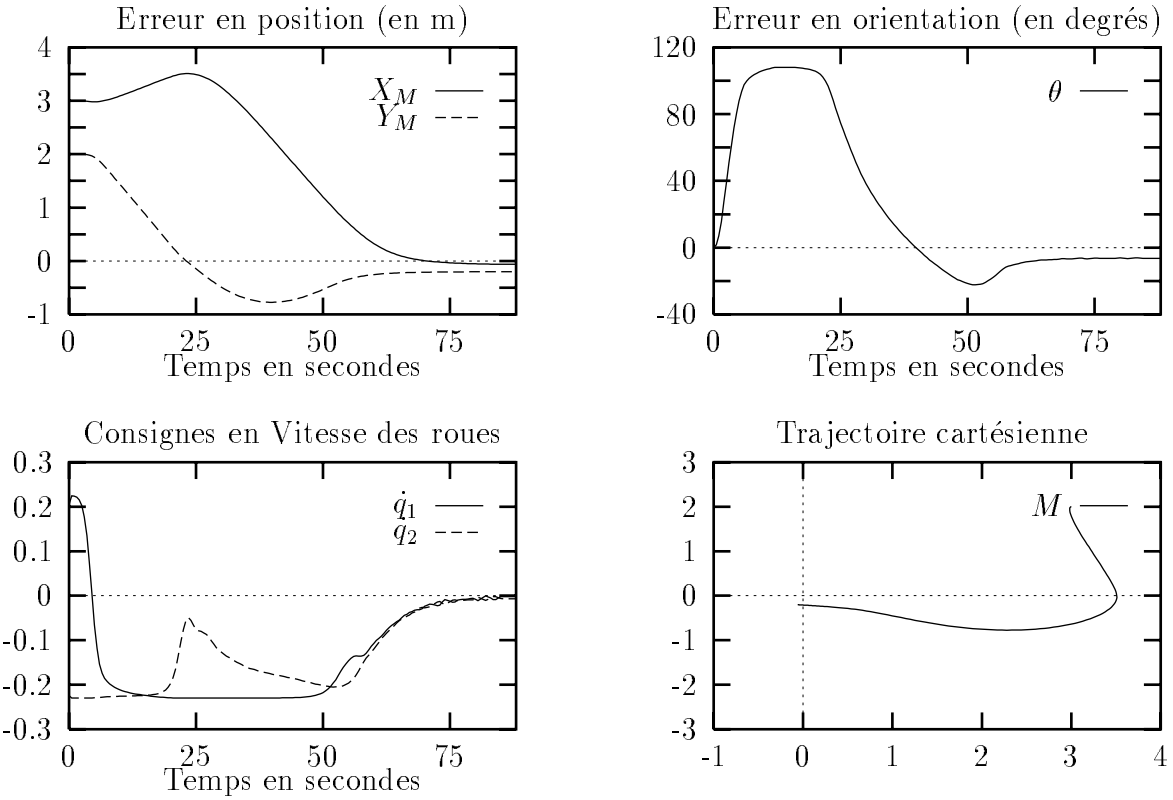

Fig. 8.11: *Expérimentation après apprentissage réel en-ligne*

8.4 Conclusion sur l'apprentissage En-Ligne

Les résultats obtenus par cette méthode sont très satisfaisants. Nous avons en effet montré par l'expérience que pour le contrôle d'un robot mobile :

- Un réseau de neurones généralise la commande avec l'apprentissage En-Ligne, avec le simulateur et avec le robot.
- Cette généralisation est rapide (trois apprentissages).
- Il y a adaptation au comportement réel du robot.
- la simplicité et l'autonomie (il n'y a pas de supervision de l'apprentissage) de la structure d'apprentissage/commande.

Nous avons toutefois observé des phénomènes d'oscillations pendant l'apprentissage sur le robot réel. Celà n'a aucune incidence sur le chariot mobile que nous contrôlons puisque c'est un système à très faible bande passante.

Cependant, ll faudra faire attention dans le cas de la commande de système plus rapide (manipulateur, petit véhicule...). Les brusques variations de consignes pourraient devenir destructrices. Pour l'apprentissage sur le robot réel, il faudra utiliser un réseau formé sur un simulateur.

8.5 Conclusion de la seconde partie

Nous avons montré que des RNF peuvent contrôler un robot mobile avec un simple retour d'erreur stationnaire.

Nous avons utilisé deux techniques différentes d'apprentissage :

- La première utilise, sans aucun modèle, une expertise qualitative du comportement désiré en fonction de la configuration du robot. Elle est supervisée et s'apparente à la fois à de l'apprentissage de règles floues et à de la classification. Cette méthode ne permet pas un apprentissage en-ligne.

- La seconde utilise un critère qui exprime mathématiquement le but à atteindre en tenant compte des contraintes cinématiques. Elle est autonome et se rapproche des techniques de renforcement. Elle n'utilise aucune connaissance a priori sur les trajectoires. Elle permet un apprentissage autonome en-ligne qui peut être relancé à tout moment pour adapter le réseau aux paramètres réels du robot. Elle s'apparente dans ce cas au contrôle adaptatif.

Dans les deux cas, les réseaux optimaux sont les mêmes. Ils sont de petites tailles. Le calcul de la commande est donc rapide. Les précisions obtenues sont acceptables compte tenu de la taille du robot. Les erreurs statiques, quand elles existent, peuvent êtres diminuées en changeant le facteur d'échelle à l'entrée du réseau.

Nous n'avons toutefois pas résolu entièrement le problème du "créneau", c'est à dire des configurations singulières pour lesquelles $X_M = \theta = 0.0$.
La RPD est toutefois une solution assez acceptable, qui découple les variables à commander.
La RPI Hors-Ligne, telle que nous l'avons utilisée ne propose pas de solution satisfaisante. Cependant, sa souplesse d'utilisation permet de modifier le critère afin d'y inclure une contrainte de comportement face à ces situations.
Enfin, la RPI En-Ligne est une solution tant que l'apprentissage est appliqué. Les résultats montrent en effet qu'ensuite, le RNF ne sait pas commander le robot pour ces configurations. Nous pouvons cependant, comme pour la RPI Hors-Ligne, faire les mêmes remarques sur la modification du critère.

Pour parvenir jusqu'aux expérimentations, nous avons dû franchir toutes les difficultés liées à l'utilisation des techniques neuronales pour le contrôle des systèmes. Les réponses que nous avons proposé face à ces difficultés, nous ont permis de tisser une méthodologie et d'acquérir une expérience sur les problèmes suivants :

- Du point de vue connexionnisme :
  - La structure de commande :
    - Choix des paramètres d'entrées,
    - Choix de leur normalisation,
    - Influence de cette normalisation.
    - Choix de la taille du réseau
  - La structure d'apprentissage :
    - Comment construire la base d'apprentissage,
    - Quelle taille doit avoir cette base.
- Du point de vue robotique :
  - Valider expérimentalement notre méthode par un vrai problème de robotique où le temps intervient.
  - Montrer qu'on peut utiliser un réseau de neurones dans une boucle réelle de commande.
  - Comparer expérimentalement l'apprentissage direct (par sorties désirées) et indirect (par critère).
  - Montrer qu'il est possible de faire de la commande neuronal adaptatif.
  - Proposer une solution simple pour la commande en position et en orientation du chariot mobile.

Nous nous sommes attaqués au problème du robot mobile dans un but de comparaison des apprentissages et de validation expérimentale de la commande neuronale. Nous sommes toutefois convaincus, que l'utilisation de l'algorithme de rétropropagation indirecte (hors-ligne et en-ligne) reste plus intéressant dans un problème où un comportement réactif est nécessaire pour la commande du système. La partie suivante du rapport présente une application qui va dans ce sens.

Partie III

Vers la locomotion Articulée

Chapitre 9

Contrôle de l'équilibre dynamique d'un bipède

Le travail présenté dans ce chapitre a pour but de montrer l'aptitude des réseaux de neurones et de la rétropropagation indirecte pour l'apprentissage et la génération de comportements réflexes dans le cadre de la locomotion articulée. C'est pourquoi nous ne ferons pas de synthèse sur l'étude et le contrôle des systèmes robotiques à pattes. Pour plus d'informations dans ce domaine le lecteur lira [*Ouez 90*].

Nous avons vu au chapitre 4 que la RPI[1] permet de minimiser un critère analytique qui exprime l'objectif et les contraintes de la commande. Nous avons montré au chapitre 8 que son utilisation En-Ligne permet d'adapter le réseau obtenu Hors-Ligne aux conditions réelles d'utilisation. Nous sommes convaincus que cette technique doit être utilisée dans un cadre où la complexité de la dynamique du système à commander est telle, que lorsqu'il se trouve dans une configuration inhabituelle qui menace son intégrité, un comportement réactif est nécessaire. C'est le cas en locomotion articulée lorsque par exemple, une patte du robot glisse.

Afin d'illustrer cette affirmation, nous proposons ici d'étudier en simulation le contrôle de l'équilibre dynamique d'un bipède au cours d'une enjambée [*Schw 92*]. Le modèle utilisé est plan et le comportement dynamique est pris en compte. Un générateur d'allure impose les accélérations articulaires tandis qu'un réseau de neurones calcule l'accélération articulaire du tronc pour garantir l'équilibre dynamique de l'ensemble. Aucune connaissance a priori sur les trajectoires n'est utilisée. Seul, l'état articulaire du robot est pris en compte.

Le critère utilisé est la position du *Zero-Moment-Point* (ZMP) par rapport au point de contact avec le sol. Après l'apprentissage Hors-Ligne, pour lequel nous montrons l'importance de la méthode de construction de la base d'apprentissage, nous comparons le contrôle neuronal avec une commande analytique (possible pour un système plan).

[1] RétroPropagation Indirecte

Les simulations montrent que notre méthode permet d'anticiper l'état d'équilibre dynamique du système, et que l'apprentissage en-ligne permet de l'améliorer.

9.1 Introduction

L'idée de départ de notre travail s'inspire directement des travaux de Atsuo Takanishi et de son équipe à l'Université de Waseda à Tokyo ([*Taka 85*], [*Taka 88*], [*Taka 89*], [*Taka 90*], [*Taka 92*], [*Taka 93*]). En 1986, ils construisent un robot bipède, le WL-12, de la taille d'un humain. Il est muni d'un tronc dont la commande permet de contrôler la stabilité dynamique de l'ensemble du robot pendant son déplacement. Le mouvement du tronc est calculé en fonction de la trajectoire d'un point qui exprime un critère d'équilibre dynamique. Ce point, appelé "Zero-Moment-Point" ou ZMP, a été défini pour la première fois par A. Vukobratović dans [*Vuko 70*].

D'autre part, si l'on se réfère à la littérature pour l'utilisation des réseaux de neurones en locomotion articulée, on ne trouve à notre connaissance que des travaux sur la génération d'allures statiques. On peut citer [*Zhen 90*], [*Zoma 93*], [*Sala 92a*], [*Sala 92b*], [*Chie 92*], [*Holl 92*], [*Touz 92*], [*Min 92*] et [*Bass 93*].

Dans le domaine de la commande d'allure dynamique, en plus de [*Lee 92*] cité au chapitre 2, Helferty ([*Helf 89*]) propose toutefois d'utiliser un réseau de neurones pour contrôler la stabilité d'un monopède. L'apprentissage utilise une technique de renforcement qui évalue l'énergie totale du système et qui récompense ou pénalise le réseau suivant sa variation. Le réseau apprend ainsi à calculer une consigne qui minimise la fonction d'énergie.

9.2 Modélisation du bipède

9.2.1 Le robot Japonnais WL-12

Le robot WL-12 (voir photographie 9.1) pèse environ 100 kg et mesure 1,70 m. Il se déplace dans le plan sagital, mais les mouvements du tronc sont à deux dimensions. Le corps de chaque segment est en fibre de carbone renforcé et les actionneurs sont localisés sur les axes de liaisons. Ce qui permet à A. Takanishi de répartir la masse du système de façon ponctuelle sur les articulations comme le montre la figure 9.2.

9.2.2 Modèle utilisé

Dans un but de simplification, nous considérons le problème plan. Le tronc ne possède donc qu'un seul degré de liberté. Les pieds sont assimilés à des points et les masses sont plus faibles que celles du WL-12.

Notre bipède devient alors équivalent au modèle de la figure 9.3.

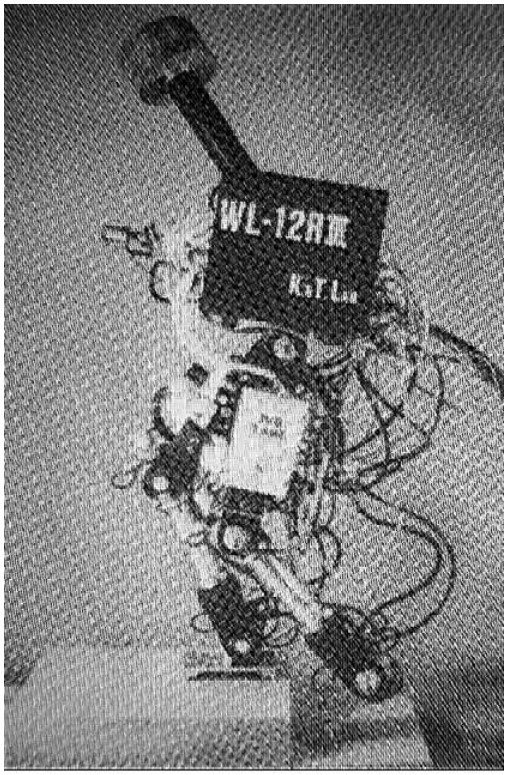

Fig. 9.1: *Bipède Japonais WL12*

Les longueurs et masses des segments sont les suivantes :

$$\begin{array}{lcllcl} & & & m_0 & = & 2 \text{ kg} \\ L_1 & = & 1.0 \text{ m} & m_1 & = & 3 \text{ kg} \\ L_1' & = & L_1'' = 0.7 \text{ m} & m_1' & = & m_1'' = 1 \text{ kg} \\ L_2' & = & L_2'' = 0.6 \text{ m} & m_2' & = & m_2'' = 0.5 \text{ kg} \end{array}$$

Les actionneurs sont supposés parfaitement asservis en accélérations. C'est à dire que nous les supposons capables d'exercer des couples proportionnels aux consignes en accélération.

9.2.3 Modèle de déplacement

Un bipède qui se déplace enchaîne deux phases de configuration distinctes ([*Ouez 90*]) :

- Celle dite *Double-Support-Phase* au cours de laquelle les deux pieds sont en contact avec le sol. Elle est d'autant plus brève que l'allure est rapide. Durant cette phase, le robot doit être en équilibre statique ([*Gama 94*]).
- Celle dite *Single-Support-Phase* au cours de laquelle un seul pied est en contact avec le sol. Elle est d'autant plus longue que l'allure est rapide. Durant cette phase, le robot doit être en équilibre dynamique.

Dans le cadre de notre étude, nous ne nous intéressons qu'à la seconde phase, au cours d'une seule enjambée, pour laquelle le pied de la patte 2 est en contact avec le sol. En effet,

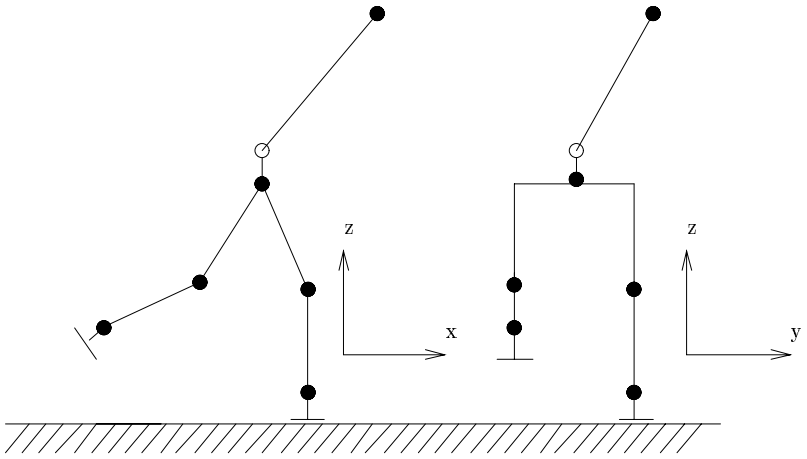

Fig. 9.2: *Modélisation de la répartion de masses du WL-12*

Lee et ElMaraghy ont montré dans [*Lee 92*] (voir chapitre 3) qu'il est possible d'utiliser séquentiellement deux réseaux de neurones pour une succession de SSP[2]. La trajectoire de la patte 1 sera cycloïdale, afin de finir l'enjambée avec une vitesse de posé nulle. Ceci nous permet de négliger la force d'impact au contact.

9.3 Equilibre Dynamique du Bipède

Nous rappelons brièvement dans le paragraphe qui suit les notions d'équilibre d'un système multicorps soumis à des efforts extérieurs dans le plan.

9.3.1 Rappels sur l'équilibre d'un système : ZMP

Soit un solide indéformable, sans masse, dans le plan soumis à une force $\vec{F} = (F_x\ F_y)^T$ au point N de coordonnées (x, y) dans un repère cartésien d'origine O. Le moment par rapport à l'origine du repère s'écrit :

$$\vec{M} = \vec{ON} \times \vec{F} = (F_y x - F_x y)\vec{z}$$

Si le système est soumis à i forces et j moments extérieurs, alors le moment total s'écrit :

$$\vec{M}_{tot} = (\sum_i (F_{i,y} x_i - F_{i,x} y_i) + \sum_j M_{j,z})\vec{z} \tag{9.1}$$

En appliquant le principe fondamental de la statique au point $P\ (x_P\ y_P)$:

$$\vec{M}_O + \sum_i \vec{PO} \times \vec{F}_i = \vec{0} \tag{9.2}$$

[2]Single-Support-Phase

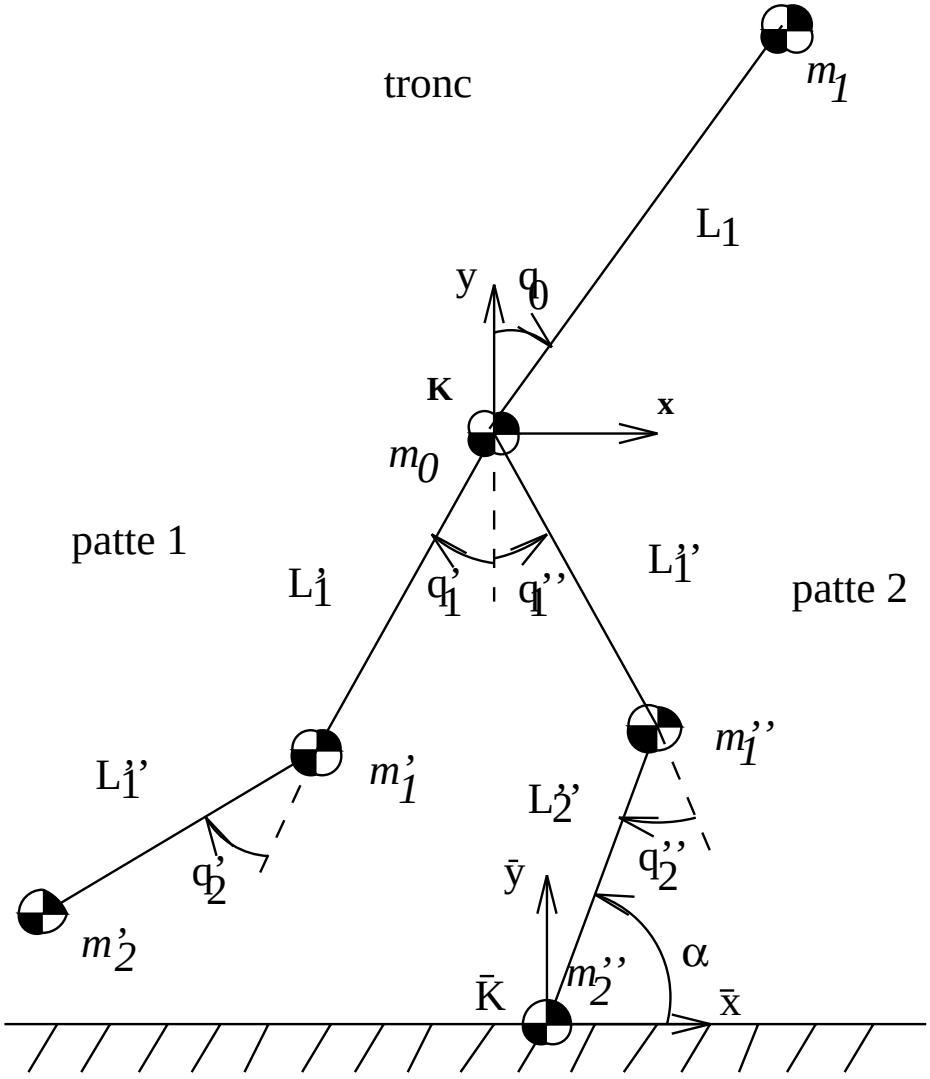

Fig. 9.3: *Modèle de bipède que nous utilisons*

d'où

$$\left(\vec{M}_O + \sum_i (-x_P\vec{x} - y_P\vec{y}) \times (F_{i,x}\vec{x} + F_{i,y}\vec{y})\right)\vec{z} = \vec{0} \tag{9.3}$$

ce qui donne en choisissant $y_P = 0$

$$\sum_i (F_{i,y}x_i - F_{i,x}y_i) + \sum_j M_{j,z} - \sum_i x_P F_{i,y} = 0 \tag{9.4}$$

On trouve ainsi l'abcisse d'un point pour lequel la somme des forces et des moments extérieurs est nulle. Ce point appelé *Zero-Moment-Point* est donné par :

$$x_{zmp} = \frac{\sum_i F_{i,y}x_i - \sum_i F_{i,x}y_i + \sum_j M_j}{\sum_i F_{i,y}} \tag{9.5}$$

9.3.2 Application à l'équilibre du bipède

Considérons le bipède de la figure 9.3 qui est un système multicorps en mouvement et soumis à la gravité. Le principe fondamental de la dynamique permet d'écrire qu'il y a égalité entre son moment dynamique et le moment total des forces auquel il est soumis.

Calcul du moment dynamique

Nous devons d'abord calculer son moment cinétique dans le repère $\bar{K}$ au point O. Il s'écrit pour chaque corps i :

$$\vec{\sigma}_{O(ib/\bar{K})} = J_{i,\bar{K}}\vec{\Omega}_{i/\bar{K}} + m_i\vec{OP}_i \times \vec{V}_{P_i/\bar{K}} \tag{9.6}$$

où $J_{i,\bar{K}}$ est la matrice d'inertie de chaque corps et P_i leur centre de masse. Or le modèle choisi a des masses ponctuelles situées aux liaisons. Le moment cinétique du bipède entier s'écrit donc :

$$\vec{\sigma}_{O(bip/\bar{K})} = \sum_i m_i(x_i\dot{y}_i - y_i\dot{x}_i)\vec{z} \tag{9.7}$$

Par dérivation, nous obtenons le moment dynamique selon l'axe $\vec{z}$:

$$\vec{\delta}_{O(bip/\bar{K})} = \sum_i m_i(x_i\ddot{y}_i - y_i\ddot{x}_i)\vec{z} \tag{9.8}$$

On obtient finalement l'expression du moment dynamique du bipède entier selon l'axe $\vec{z}$, exprimé au point $P\ (x_P\ , 0)$:

$$\vec{\delta}_{P(bip/\bar{K})} = \vec{\delta}_{O(bip/\bar{K})} + \vec{PO} \times \sum_i m_i\vec{\Gamma}_{P_i/\bar{K}} \tag{9.9}$$

d'où :

$$\vec{\delta}_{P(bip/\bar{K})} = \sum_i m_i(x_i\ddot{y}_i - y_i\ddot{x}_i)\vec{z} - x_P\sum_i m_i\ddot{y}_i\vec{z} \tag{9.10}$$

Calcul du moment résultant des forces

Le calcul est le même que pour le solide, mais en considérant la gravité. Le moment total des efforts et moments extérieurs appliqués au bipède, et exprimé au point P $(x_P,\ 0)$ s'écrit :

$$\vec{M}_P = \vec{M}_O + \vec{PO} \times \left(\sum_i F_{i,x}\vec{x} + F_{i,y}\vec{y} - m_i g \vec{y} \right) \tag{9.11}$$

finalement :

$$\vec{M}_P = \left(\sum_i (F_{i,y}x_i - F_{i,x}y_i) + \sum_j M_{j,z} - \sum_i m_i g x_i - x_P \sum_i (F_{y,i} - m_i g) \right) \vec{z} \tag{9.12}$$

Calcul du ZMP

On traduit l'état d'équilibre du système bipède par l'égalité :

$$\vec{\delta}_{P(bip/\bar{K})} = \vec{M}_P$$

En appliquant ainsi les équations 9.12 et 9.10, on trouve l'abcisse du ZMP pour le bipède considéré dans le repère $\bar{K}$:

$$\bar{x}_{zmp} = \frac{\sum_i m_i(\ddot{\bar{y}}_i + g)\bar{x}_i - \sum_i m_i \ddot{\bar{x}}_i \bar{y}_i}{\sum_i m_i(\ddot{\bar{y}}_i + g) - \sum_k F_{k,y}} - \frac{\sum_j M_{j,z} + \sum_k (F_{k,y}\bar{x}_k - F_{k,x}\bar{y}_k)}{\sum_i m_i(\ddot{\bar{y}}_i + g) - \sum_k F_{k,y}} \tag{9.13}$$

où

$\bar{x}_i, \bar{y}_i$	:	Coordonnées de la masse m_i dans le repère $\bar{K}$
$\bar{x}_k, \bar{y}_k$	:	Coordonnées du point d'application de la force F_k
$F_{k,x}, F_{k,y}$	:	Composantes des forces extérieures F_k
M_j	:	Moment extérieur
g	:	gravité
$\bar{x}_{zmp}, 0$	:	Coordonnées du ZMP

Le contact patte/sol étant supposé ponctuel dans le plan, il n'y a pas de moment exercé en ce point par la force de contact. Comme nous ne considérons pas les couples articulaires, le second terme de l'équation 9.14 est nul. De plus, si on considère un repère K attaché à une plateforme virtuelle au niveau de la hanche du bipède, et qui n'a pas de mouvement de rotation par rapport au repère $\bar{K}$, on peut utiliser les propriétés de symétrie des pattes par rapport à la hanche.

L'équation 9.14 peut alors être simplifiée, et on obtient l'expression du ZMP en fonction des accélérations absolues de la plateforme :

$$\bar{x}_{zmp} = \frac{\sum_i m_i(\ddot{y}_i + \ddot{\bar{y}}_{pl} + g)(x_i + \bar{x}_{pl}) - \sum_i m_i(\ddot{x}_i + \ddot{\bar{x}}_{pl} + g)(y_i + \bar{y}_{pl})}{\sum_i m_i(\ddot{y}_i + \ddot{\bar{y}}_{pl} + g)} \tag{9.14}$$

en posant : (9.15)

$$Fx_y = \sum_i m_i(\ddot{x}_i + \ddot{\bar{x}}_{pl} + g)(y_i + \bar{y}_{pl})$$

$$Fy_x = \sum_i m_i(\ddot{y}_i + \ddot{\bar{y}}_{pl} + g)(x_i + \bar{x}_{pl})$$

$$Fy = \sum_i m_i(\ddot{y}_i + \ddot{\bar{y}}_{pl} + g) \quad (9.16)$$

on a : (9.17)

$$\bar{x}_{zmp} = \frac{Fy_x - Fx_y}{Fy} \quad (9.18)$$

Le critère $\bar{x}_{zmp}$ est donc une fonction qui dépend, via le modèle géométrique et ses dérivées première et seconde, de l'état articulaire du robot : $q_i, \dot{q}_i, \ddot{q}_i$. Les mouvements de la plateforme sont supposés connus, c'est à dire mesurés par un capteur dans le cas d'un robot réel.

Critère d'équilibre dynamique

A. Vukobratović propose de prendre comme critère d'équilibre dynamique durant la SSP, la distance de la projection verticale du ZMP par rapport à la surface de contact. Ce critère est repris par A. Takanishi dans ses travaux. C'est une extension, au cas dynamique, de la notion d'équilibre statique, pour lequel le centre de masses du système doit se trouver à la verticale de la surface de sustentation. Plus cette distance est grande, plus le système est déséquilibré. Dans notre cas, la surface de contact se ramène à un point. Notre bipède sera donc dynamiquement stable si le ZMP se trouve à la verticale du point de contact, c'est à dire à la verticale du repère $\bar{K}$.

Le critère d'équilibre dynamique s'écrit donc :

$$(\bar{x}_{zmp} - \bar{x}_{contact})^2 = 0$$

C'est à dire, exprimé dans le repère $\bar{K}$:

$$\bar{x}^2_{zmp} = 0$$

L'apprentissage neuronal présenté plus loin aura pour but de minimiser ce critère.

9.4 Structure de Commande

A. Takanishi explique dans [*Taka 85*] qu'ils ont décidé de construire un bipède doté d'un tronc suite à un premier prototype sans tronc, qu'il était très difficile de stabiliser lors des changements d'allures. Avec un tronc, on peut ainsi découpler le problème du déplacement de la machine, du problème du contrôle de l'équilibre. Les pattes servent à déplacer le système et le mouvement du tronc à le stabiliser.

Nous appliquons ce concept de la manière suivante :

- Un générateur de consignes impose les accélérations articulaires aux pattes. Ces accélérations sont calculées à partir de trajectoires cycloïdales. Celles-ci ont l'avantage d'être définies uniquement par la longueur et le temps d'enjambée. De plus, la vitesse de posé d'une patte pour une cycloïde est nulle, ce qui permet de négliger la force d'impact.

- Le réseau de neurones contrôle le mouvement du tronc, en calculant son accélération articulaire à partir de l'observation de l'état articulaire du robot.

La structure de commande peut alors être schématisée par la figure 9.4. L'entrée du réseau

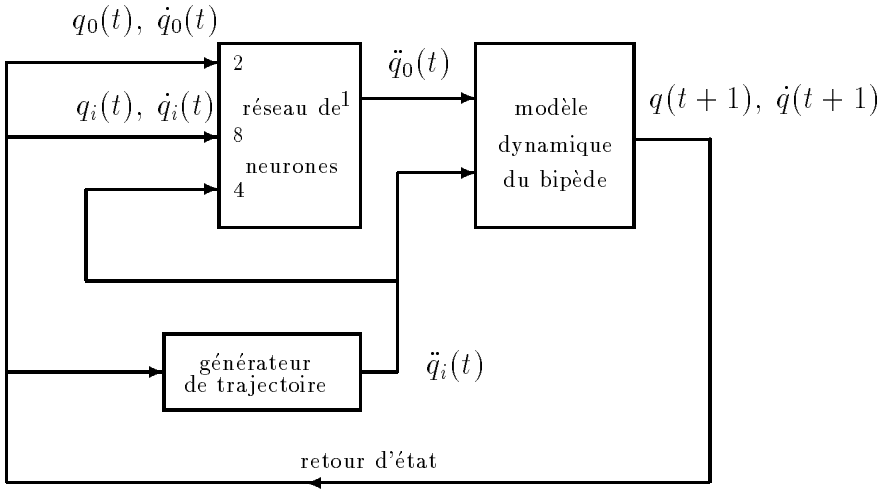

Fig. 9.4: *Structure de contrôle du bipède*

est constituée des positions, vitesses et accélérations articulaires des pattes ($q_i,\ \dot{q}_i,\ \ddot{q}_i$), et de la position et vitesse articulaire du tronc ($q_0, \dot{q}_0$). La couche d'entrée possède donc 14 neurones et la couche de sortie un seul.

La simulation du comportement dynamique du bipède est très simple compte tenu des hypothèses que nous avons posées. Il est inutile d'utiliser un modèle récursif. En effet, à l'instant t, on connait tous les paramètres angulaires (positions, vitesses et accélérations). Vu du point de contact (origine du repère $\bar{K}$), le système est rigide. On peut le considérer comme un corps tournant autour de ce point. La dynamique est très simple et se ramène au calcul de l'accélération $\ddot{\bar{\alpha}}$, à partir de laquelle on peut calculer par intégration $\dot{\bar{\alpha}}$ et $\bar{\alpha}$, puis en déduire les positions et vitesses des articulations.

9.5 Apprentissage Hors-ligne

9.5.1 Choix du critère et calcul du gradient

L'objectif du contrôle est de maintenir le robot en équilibre dynamique durant la SSP. Compte tenu des remarques précédentes, nous pouvons donc choisir comme critère pour la rétropropagation indirecte, la position du ZMP par rapport au point de contact. C'est à

dire :

$$J(\ddot{q}_0) = \bar{x}_{zmp}^2$$

L'apprentissage doit permettre au réseau d'apprendre à minimiser ce critère. Pour cela, nous devons calculer le gradient de $J(\ddot{q}_0(t))$ par rapport à la sortie du réseau, c'est à dire l'accélération articulaire $\ddot{q}_0(t)$ du tronc.

En se référant à l'équation 9.18, nous obtenons ainsi :

$$\frac{\partial J}{\partial \ddot{q}_0} = 2\,\bar{x}_{zmp}\frac{\partial \bar{x}_{zmp}}{\partial \ddot{q}_0}$$

c'est à dire

$$\frac{\partial J}{\partial \ddot{q}_0} = 2\,\bar{x}_{zmp} m_1 L_1 \left(\frac{Fy_x - Fx_y}{Fy^2}\sin q_0 + \frac{(y_1 + \bar{y}_{pl})\cos q_0 - (x_1 + \bar{x}_{pl})\sin q_0}{Fy} \right)$$

9.5.2 Construction de la base d'apprentissage

Nous devons créer pour l'apprentissage un ensemble d'exemples qui doit être représentatif des états possibles du robot.

Pour ce faire, une technique consiste à utiliser un processus aléatoire. On ne peut pas l'appliquer dans notre cas pour au moins deux raisons (des tentatives l'ont confirmé) :

1. La taille du vecteur d'entrée (14) imposerait un trop grand nombre d'exemples. A titre d'illustration, si l'on considère seulement 3 états (-1,0,+1) pour chaque entrée, on obtient $3^{14} = 4782969$ exemples.

2. On génèrerait des états qui n'ont aucun sens physique. et qui n'ont par conséquent pas de solution dans des limites atteignables par le réseau. Ces mauvais exemples empêcheraient le réseau de généraliser le contrôle.

Il convient donc de construire la base d'apprentissage en tenant compte des contraintes géométriques du robot (butées articulaires), et des limites du réseau (accélérations maximales). La dernière contrainte est une condition d'atteignabilité du ZMP par le réseau. C'est à dire que nous ne sélectionnerons que les exemples qui permettent au réseau de corriger l'équilibre du robot, compte tenu de l'accélération maximale qu'il peut calculer.

Face à ces remarques, nous présentons deux méthodes de construction de la base. La première calcule des configurations articulaires aléatoires dans les limites du robot.

La seconde construit une répartition uniforme de la position du pied de la patte 1, et par le modèle inverse géométrique en déduit la configuration articulaire.

Dans les deux cas, nous considérons pour l'enjambée les paramètres suivants :

y_{pl} : hauteur moyenne de la plateforme (hanche).
dy_{pl} : amplitude des variations de hauteur de la plateforme.
l_{max} : longueur maximale du pas.
h_{max} : hauteur maximale du pas.

Le réseau de neurones devra contrôler toutes les trajectoires de la patte 1 incluses dans le rectangle $l_{max} \times h_{max}$.

Dans l'espace articulaire

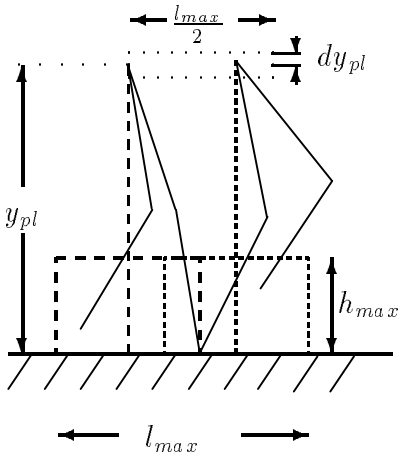

Fig. 9.5: *Shéma de construction des exemples dans l'espace articulaire*

Une fois fixés ces paramètres, on construit la base d'exemples avec l'algorithme suivant :

1. Générer aléatoirement, en tenant compte des butées, les positions articulaires de la patte 2 ($q_1^{"}$ et $q_2^{"}$). Ne retenir que les configurations pour lesquelles la hanche est située dans un rectangle de longueur $\frac{l_{max}}{2}$, de hauteur $2dy_{pl}$ et placé en $(0,\ y_{pl})$

2. Générer aléatoirement, en tenant compte des butées, les positions articulaires de la patte 1 ($q_1^{'}$ et $q_2^{'}$). Ne retenir que les configurations pour lesquelles l'extrémité de la patte est située dans un rectangle de hauteur h_{max} et de longueur l_{max}.

3. Générer aléatoirement, sur ces configurations, les vitesses et accélérations articulaires. Ne retenir que les exemples pour lesquelles le ZMP est atteignable par le réseau.

Les résultats montrent que cette méthode ne permet pas d'obtenir une distribution harmonieuse de la position cartésienne de l'extrémité de la patte 1. Ceci veut dire que certaines trajectoires incluses dans ce rectangle ne seront pas contrôlables. La figure 9.6 donne un aperçu de ces positions dans le rectangle $l_{max} \times h_{max}$.

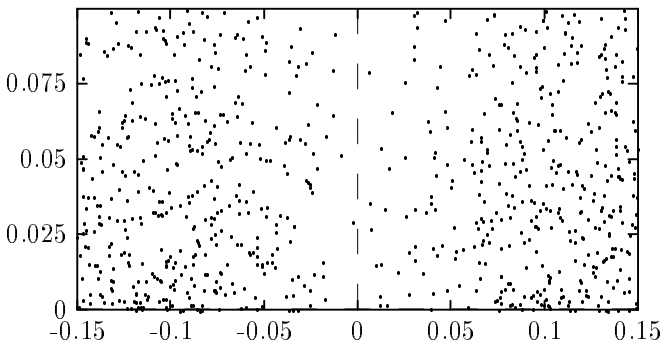

Fig. 9.6: *Distribution de l'extrémité de la patte 1 obtenue par le processus aléatoire*

Dans l'espace cartésien

La méthode utilisée ici est simple. On répartit uniformément l'extrémité de la patte 1 dans le rectangle $l_{max} \times h_{max}$. Un pas fixe le nombre de positions, donc le nombre d'exemples. A partir de la position cartésienne, on déduit par le modèle géométrique inverse la confi-

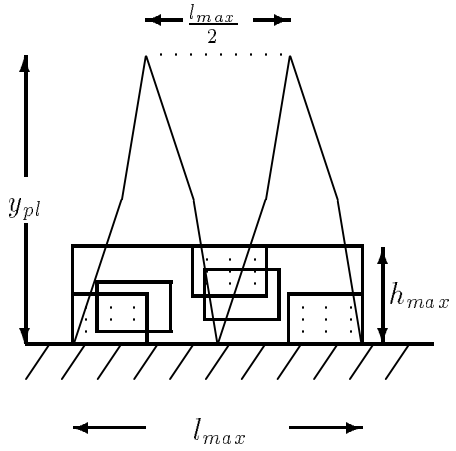

Fig. 9.7: *Schéma de construction des exemples dans l'espace cartésien*

guration articulaire. On génère ensuite aléatoirement sur ces configurations, les vitesses et accélérations articulaires. On ne retient que les états pour lesquels le ZMP est atteignable par le réseau.

9.5.3 Résultats de l'apprentissage

Nous avons utilisé, afin de les comparer, les deux méthodes de génération des exemples, pour une enjambée d'une hauteur maximale de 0,1 m, et d'une longeur maximale de 0.3 m. Le réseau choisi possède une couche cachée de 12 neurones. L'apprentissage est effectué sur 10000 itérations pour 800 exemples.

Après l'apprentissage Hors-Ligne, nous avons testé, pour une enjambée de 0,25m×0,075m, le contrôle de l'équilibre par les deux réseaux obtenus. La figure 9.8 montre l'évolution de l'abcisse du ZMP pendant cet essai. Le réseau n^o1 correspond à l'apprentissage sur les exemples

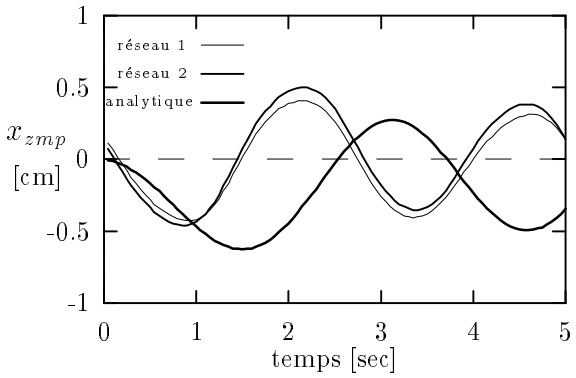

Fig. 9.8: *Trajectoire du ZMP au cours d'une enjambée.*

générés dans l'espace articulaire.

Le réseau n^o2 à l'apprentissage sur les exemples générés dans l'espace cartésien. On constate peu de différences. L'amplitude et l'allure sont quasiment identiques. D'autres essais montrent toutefois que le second réseau contrôle mieux l'équilibre pour des trajectoires extrèmes (proches des limites du rectangle).

A titre de comparaison, nous montrons le résultat d'un contrôle analytique simple. Les hypothèses simplificatrices faites sur le modèle, nous permettent en effet de calculer l'accélération instantanée du tronc pour contrôler l'équilibre.
On constate une différence fondamentale. Avec la solution analytique, la valeur moyenne de la position du ZMP diverge. L'apprentissage, et notamment la minimisation du critère $J(t+1)$ sur l'état résultant de la commande, permet donc d'anticiper et de corriger le déséquilibre.

9.6 Apprentissage En-Ligne

Nous appliquons avec le réseau n^o2 l'algorithme d'apprentissage en-ligne. Celui-ci doit affiner les valeurs des poids au cours du contrôle.

Il est important de noter ici, que contrairement aux expériences menées sur le robot mobile (des tentives l'ont montré), nous ne pouvons pas utiliser un réseau qui n'a pas appris Hors-Ligne. En effet, la taille de l'entrée, et la nature même du problème font que l'aspect incohérent du contrôle en début d'apprentissage amène le robot dans une configuration sans solution pour le réseau (par exemple, le bipède tombe à terre).

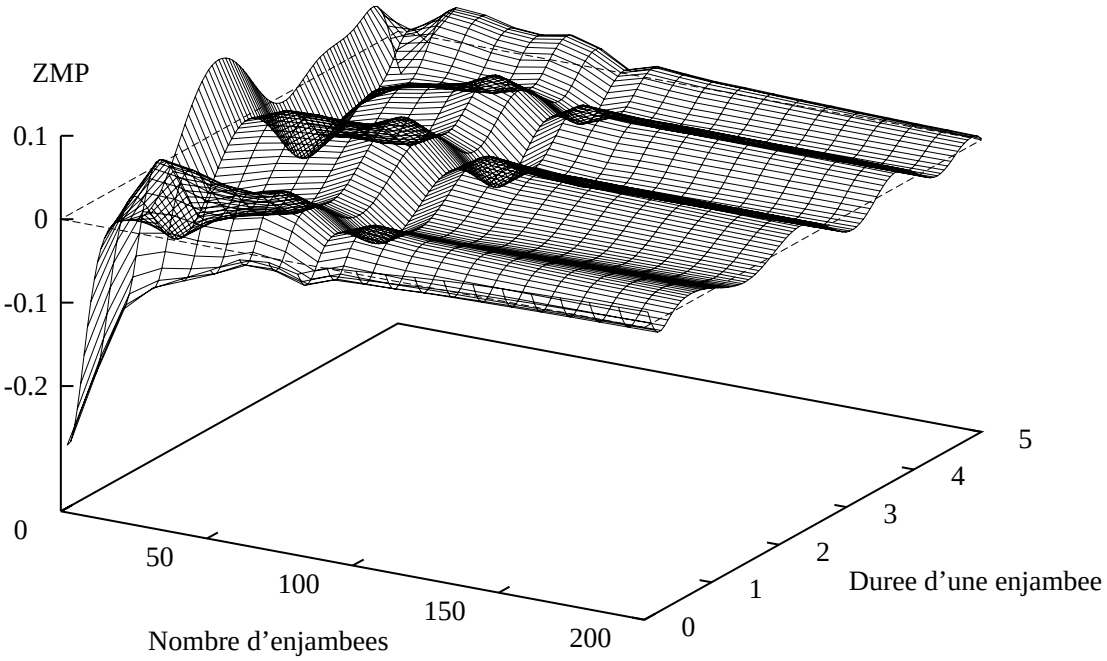

Fig. 9.9: *Evolution de la trajectoire du ZMP pendant l'apprentissage En-Ligne*

Compte tenu de l'aspect périodique du problème, nous procédons de la manière suivante :

- Le robot exécute un grand nombre de fois la même enjambée.
- La mise à jour des poids se fait à la fin de chaque enjambée.

Du point de vue de la rétropropagation, nous nous situons donc entre l'apprentissage Hors-Ligne et l'apprentissage En-Ligne, puisque la suite de configurations par lesquelles passe le robot pendant une enjambée, sert de base d'apprentissage (nous calculons un gradient moyen sur cette base).

Les résultats sont montrés sur la figure 9.9 où nous avons tracé l'évolution de la trajectoire du ZMP au cours de l'apprentissage.

On constate nettement une décroissance de l'amplitude du mouvement. Ceci veut dire, que le contrôle de l'équilibre est fortement amélioré avec cette technique. De plus, si on teste le réseau après l'apprentisasge En-Ligne, sur d'autres trajectoires les résultats sont aussi nettement améliorés. Il y a donc apprentissage et généralisation pendant cette phase.

9.7 Conclusion

Les résultats que nous présentons dans ce chapitre sont issus d'une première étude sur l'intérêt de la rétropropagation indirecte en locomotion articulée. Même si nous avons beaucoup simplifié le problème de départ, nous pouvons affirmer que le but recherché est atteint. En effet :

- Nous contrôlons l'équilibre dynamique du système sans aucune connaissance a priori sur les trajectoires.
- Toutes les trajectoires, quelques soient leurs formes sont contrôlées, pourvu quelles restent incluses dans le domaine qui a servi d'apprentissage.
- L'apprentissage En-Ligne a amélioré nettement les résultats tout en contrôlant le système.

De plus, nous avons montré l'importance de la méthode de construction des exemples. Il vaut mieux construire une base uniformément répartie car cela permet d'être plus représentatif des états possibles et par conséquent d'éliminer les mauvais exemples qui gênent l'apprentissage.

Partie IV

Conclusion Générale

Chapitre 10

Conclusion et Perspectives

Nous avons présenté dans ce mémoire de thèse des applications de la commande neuronale en robotique mobile à roues et à pattes. Nous nous étions fixé comme objectif d'utiliser les perceptrons multicouches et l'algorithme de rétropropagation du gradient afin d'évaluer leurs performances et d'approfondir leurs mises en œuvre par diverses expérimentations.

Nous avons pour cela (chapitre 2), décrit l'outil réseau de neurones et la RétroPropagation Directe (RPD) qui minimise une erreur quadratique calculée entre la sortie du réseau et une sortie désirée. Nous avons ensuite présenté (chapitre 3) quelques applications bibliographiques de la commande neuronale en robotique.

Nous avons montré (début du chapitre 4) que l'inconvénient principal de la RPD en robotique, est l'utilisation d'une commande désirée pour faire l'apprentissage, car pour la calculer, deux solutions sont en général utilisées:

- La première inverse un modèle (géométrique, cinématique, dynamique) du robot. Or dans un cadre de commande dynamique de systèmes mécaniques complexes, et face à leurs redondances cinématiques, l'inversion des modèles utilise souvent les techniques d'optimisation afin de choisir, pour la commande, une solution parmis une infinité. Ceci pose des problèmes d'implémentation temps réel de la RPD En-Ligne.

- La seconde utilise un modèle de comportement (correcteur PID, correcteur flou, opérateur...) qui sert de référence pour l'apprentissage, ce qui ne permet pas d'exploiter pleinement les capacités d'apprentissage et d'identification des RNF.

En tenant compte de ces remarques, et en soulignant que la rétropropagation du gradient n'est qu'un algorithme d'optimisation, nous avons proposé ensuite une solution simple et originale: la RétroPropagation Indirecte (RPI). Elle consiste à remplacer l'erreur quadratique en sortie par un critère qui exprime directement l'objectif robotique. La structure d'apprentissage obtenue devient alors très autonome puisque disparaît totalement la notion de "Maître" dans la manière d'atteindre le but, et qu'elle peut être appliquée en-ligne se rapprochant alors des techniques de commandes adaptatives.

Dans un but de validation expérimentale de la RPI, nous l'avons appliquée à la commande en orientation et en position d'un robot mobile à deux roues motrices indépendantes, problème non trivial compte tenu du caractère non holonome du système. Nous avons comparé les commandes neuronales obtenues par RPD, par RPI Hors-Ligne et RPI En-Ligne:

- La RPD (chapitre 6) est basée sur une expertise du problème qui consiste à définir des classes de manœuvres suivant la configuration du robot. Le réseau apprend à différencier ces classes et à commander le robot différemment pour chacune d'entre elles.

- La RPI appliquée Hors-Ligne (chapitre 7) utilise uniquement un critère qui est une somme pondérée des erreurs quadratiques cartésiennes et angulaires. Le réseau apprend à minimiser ce critère sur un horizon temporel très restreint (un pas de temps). Il détermine ainsi la commande qui permet , sur cet horizon, de rapprocher le robot de la configuration finale désirée.

- La RPI appliquée En-Ligne (chapitre 8) utilise la même technique, mais dans une boucle qui commande le robot réel. La faible bande passante de ce dernier nous a permis de tester cette méthode dans le cas le plus défavorable, c'est à dire avec un réseau n'ayant subit aucun apprentissage. Nous avons pu montrer alors les capacités d'apprentissage, de généralisation et d'adaptivité du neuro-contrôleur face aux contraintes cinématiques du robot.

Les résultats obtenus et l'expérience acquise dans le cas du robot mobile à roues, nous ont incité à poursuivre vers la commande dynamique des robots à pattes (chapitre 9). Le but de cette étude étant de montrer l'intérêt d'utiliser la RPI pour les problèmes où un comportement réflexe est requis.

Nous avons proposé pour cela de contrôler l'équilibre dynamique d'un bipède plan en simulation. Le réseau commande l'accélération articulaire du tronc tandis qu'un système plus classique impose des accélérations articulaires aux pattes. Nous avons montré qu'un critère d'équilibre dynamique (souvent utilisé en locomotion articulée) peut être minimisé par la RPI Hors-Ligne, et que celà permet d'obtenir un réseau qui maintient l'équilibre dynamique sans aucune connaissance a priori sur les trajectoires. L'apprentissage appliqué En-Ligne a montré qu'il affine les paramètres du réseau obtenu Hors-Ligne, améliorant ainsi nettement la condition d'équilibre.

Finalement, nous avons donc proposé dans ce mémoire une technique d'apprentissage des perceptrons multicouches qui est originale, simple et autonome. Nous l'avons validée à travers deux applications différentes mais complémentaires. En effet:

- La première est un problème de commande d'un système non holonome dont les effets dynamiques sont inexistants. Le réseau calcule deux paramètres de commande. Le critère est simple mais le calcul du gradient a nécessité un artifice d'intégration.
  Les expériences ont montré qu'il est possible de commander la position et l'orientation du robot mobile avec un réseau de neurones. La qualité des résultats est comparable à celle obtenue par les autres méthodes développées au L.R.P (commande floue,

commande par retour d'état instationnaire). La simplicité de la méthode permet de l'intégrer dans la bibliothèque de commande du robot ROMEO.

- La seconde est un problème de commande dynamique. Le réseau calcule un seul paramètre de commande. Le critère est assez complexe mais le calcul du gradient est exact. Les simulations ont montré les possibilités d'apprentissage de comportement réflexes par la minimisation d'un critère à l'aide de la RPI. Les résultats comparés à un calcul analytique de la commande montrent l'intérêt de notre méthode. Toutefois ce problème n'est qu'une première étape vers une mise en œuvre de commandes neuronales pour la locomotion articulée dans le cadre du projet RALPHY (Robot Autonome Longitudinal Pneumatique Hybride) du L.R.P.

Cependant, les inconvénients de la Rétropropagation Indirecte se situent au niveau de l'utilisation du critère qui peut être difficile à identifier, et au niveau de son gradient qui peut être délicat à calculer par rapport aux paramètres de commande.

Les différents points que nous avons abordés au cours de ce travail ont toutefois soulevés de nombreuses questions, tant sur l'analyse théorique de la rétropropagation indirecte que sur ses champs d'application. Nous proposons donc dans un cadre de travaux futurs d'approfondir les points suivants :

- **au niveau de l'application à la robotique :**
  - Pour la commande des robots mobiles :
    prendre en compte l'environnement en intégrant dans le critère des contraintes de distances minimales par rapport aux obstacles.
  - Pour la commande des robots à pattes :
    étendre les principes proposés au chapitre 9 à un bipède non plan, puis à un robot quadrupède.
  - Pour la commande des robots manipulateurs :
    un travail d'évaluation reste à faire au niveau de la commande dynamique dans l'espace opérationnel. Des perspectives dans l'apprentissage de commandes réactives (contrôle d'efforts au contact) ne doivent pas être oubliées.
- **au niveau de la commande neuronale :**
  - Chercher une analyse théorique de la RPI en s'inspirant par exemple des outils de la commande optimale.
  - Etudier le stabilité d'un système bouclé avec un réseau de neurones.
  - Chercher une corrélation entre la taille de la base d'apprentissage et la structure du réseau.
  - Appliquer la RPI aux réseaux dynamiques afin de prendre en compte le paramètre temporel. Le concept des "réseaux à temps" en cours de développement au Laboratoire PARC serait une finalité intéressante pour la Rétropropagation Indirecte.

Références bibliographiques

[*Ait 93*] C. K. Ait-Abderrahim, *Commande de Robots Mobiles*, Thèse de l'Ecole des Mines de Paris, Janvier 1993.

[*Bass 93*] D. Bassani, M. Chiaberge, D. Del Corso, G. Gentia, F. Zanetti *Simulation of an Hexapod Walking Machine Controlled by Neural Networks*, Proceedings of the Third International Symposium on Measurement and Control in Robotics,session As.II, page 27-32,Turin ,:1993

[*Beer 93*] J. R. Beerhold, C. Jansen, R. Eckmiller *First Results on Stable Adaptive Robot Control with RBF Networks Based*, Proceedings of International Conference on Neural Networks, pp 297-300, 1993

[*Bere 92*] H. R. Berenji and P. Khedkar,*Learning and Tunning Fuzzy Logic Controllers Through Reinforcements*, IEEE Transactions on Neural Networks,Vol 3., No5 pp 724-740, 1992

[*Chen 89*] Fu-Chuang Chen, Yoh-Han Pao, *Learning control with neural networks*, Proceedings of IEEE Internationnal Conference on Robotics and Automation, pp 1448-1453,1989

[*Chen 91*] Fu-Chuang Chen *A Dead-Zone Aproach in Nonlinear Adaptive Control Using Neural Networks*, Proceedings of the 30th Conference on Decision and Control. , pp 156-161, 1991

[*Chen 92*] Fu-Chuang Chen, H. K. Khalil *Adaptive Control of Non-Linear systems Using Neural Network*, International Journal Control,Vol. 55, No 6, pp 1299-1317, 1992

[*Chen 93*] R. M. H Cheng, J. W. Xiao, S. LeQuoc,*Neuromorphic Controller ot AGV Steering*, Proceedings of the IEEE Internationnal Conference on Robotics and Automation, pp 2057-2061, 1992

[*Chie 92*] H. J. Chiel, R. D. Beer, R. D. Quinn, K. S. Espenschied *Robustness of a Distributed Neural Network Controller for Locomotion in a Hexapod Robot*, IEEE Tansactions on Robotics and Automation, Vol. 8, No 3 pp 293-303,1992

[*Cili 92*] M. K. Ciliz, C. Işik *Stability and Convergence of Neurologic Model Based Robotics Controllers*, Proceedings of IEEE International Conference on Robotics and Automation, pp 2052-2056, 1992

[*CNRS 91*] C.N.R.S et M.R.T *Rapport sur les Architectures Nouvelles de Machines: Présentation de l'activité*, Programmes de Recherches Coordonnées , Edité par le C.N.R.S et le M.R.T, 1991

[*Coif 86*] P. Coiffet, *La Robotique: Principes et aplications*, Seventh CISM - IFToMM Sym. on Theory and Practice of Robots and Manipulators, pp.89-98, Udine 12-15 Sep 1988

[*Conn 93*] T. H. Connolly and F. Pfeiffer,*Neural Network Hybrid Position/Force Control*,Neural Networks,Vol. 5, pp 305-311, 1992

[*Daxw 94*] W. Daxwanger and G. Schmidt *Skill-based Vehicule Guidance by Use of Artificial Neural Networks*, Proceedings of International Symposium on Signal Processing, Robotics and Neural Networks, pp 207-210, 1994

[*Dela 91*] S. Delaplace, *Navigation à coût minimal d'un Robot Mobile dans un Environnement Partiellement Connu*, thèse de l'Université Paris 6,novembre 1991.

[*Derr 94*] D. A. Derradji and N. Mort *Application of neural Nerworks to Underwater Vehicule*, Proceedings of International Symposium on Signal Processing, Robotics and Neural Networks, pp 576-580, 1994

[*Desj 93*] E. Desjardin, *Expérimentations et Améliorations des Techniques Neuronales pour la Reconnaissance des Formes*, thèse de l'Université de Reims,Novembre 93.

[*Feld 92*] L.A Feldkamp, G.V Puskorius, L.I Davis, Jr and F. Yuan *Neural Control Systems Trained by Dynamic Gradient Methods for Automotive Aplications*, Proceedings of Internationnal Joint Conference on Neural Networks, Boston, Vol II, pp. 798-804,1992

[*Forn 94*] W. Fornaciari and F. Salice *A VLSI Macrocell Implementation for Digital Neural Nets*, Proceedings of International Symposium on Signal Processing, Robotics and Neural Networks pp 659-663, 1994

[*Gama 94*] A. El Gamah, *Implémantation de Commande Répartie pour un Robot Quadrupède á Locomotion Pneumatique*,Thèse de l'université Paris 6, juillet 1994

[*Gard 93*] J. F. Gardner, A. Brandt, G. Lueke*Applications of Neural Network for Coordinate Transformations in Robotics*,Journal of Intelligent and Robotic Systems 8,pp 361-373, 1993

[*Gas 94*] B. Gas, *Un Modèle Connexionniste non Supervisé pour l'Apprentissage et la Reconnaissance de Séquences Temporelles*,Thèse de l'université Paris XI, mai 1994

[*Gomi 93*] H. Gomi and M. Kawato,*Neural Network Control for a Closed-loop System Unsinf Feedback-Error-Learning*,NNeural Networks,Vol 6,pp 933-946, 1993

[*Gori 92*] M. Gori and A. Tesi*On the Problem of Local Minima in Backpropagation*, IEEE Transactions on Pattern Analysis and Machine Intelligence,Vol. 14, No 1, pp 76-86,1992

[*Guez 88*] A. Guez, J. Selinsky, *A neuromorphic controller with a human teacher*, Proceedings of IEEE International Conference on Neural Networks, pp II.595-II.602, 1988

[*Gull 92*] V. Gullapalli, R. A. Grupen and A. G. Barto *Learning Reactive Admittance Control*, Proceedings of International Conference on Robotics and Automation, pp 1475,1480 1992

[*Guo 92*] Q. Guo,*An Adaptive Robust Compensation Control Scheme Using ANN for a Redundant Robot Manipulator in the Task Space*,IEEE Internationnal Conference on Robotics and Automation, pp 1908-1913, 1992

[*Gurfin*] V.S. Gurfinkel, S.V. Fomin*Biomechanical Principlies of Constructing Artficial Walking Systems*,pp 133-141

[*Helf 89*] J. Helferty, J. B. Collins, M. Kam *A Neural Network Learning Strategy for the Control of a One-Legged Hopping machine*, Proceedings of IEEE International Conference on Robotics and Automation, pp 1604-1608, 1992

[*Hena 93*] P. Henaff, H. Schwenk, M. Milgram *A Neural Network Approach of the Control of Dynamic Biped Equilibrium*, Proceedings of the Third International Symposium on Measurement and Control in Robotics, Session As.II, pp 19-25, 1993

[*Hena 94*] P. Henaff, M. Milgram *Adaptive Neural Control with Backpropagation Algorithm*, Proceedings of IMACS International Symposium on Signal Processing, Robotics and Neural Networks, pp 395-398, 1993

[*Holl 92*] O. E. Holland, M. A. Snaith *Neural Control of Locomotion in a quadrupedal Robots*, IEE Proceedings-F,Vol. 139, No 6, pp 431-436, 1992

[*Homs 92*] L. Holmström, P.Koistinen, *Using Adaptative Noise in Back Propagation Training*, IEEE transactions on neural networks, vol 3, n1, janvier 1992

[*Ichi 92*] Y. Ichikawa and T. Sawa *Neural Network Application for Direct Feedback Controllers*, IEEE Transactions on Neural Networks, Vol 3, No2, 1992

[*Josi 88*] G. Josin, D. Charney, D. White, *Robot control using neural networks*, Internationnal Joint Conference on Neural Networks, pp 625-631,1988

[*Kati 92*] D. Katìc, M. Vukobratovìc*Decomposed Connectionist Architecture for Fast and Robust Learning of Robot Dynamics*, Proceedings of IEEE International Conference on Robotics and Automation, pp 2064-2068,1992

[*Khem 93*] S. Khemaissia and A. S Morris*Neuro-Adaptive Control of Robotic Manipulators*,Robotica, Vol 11, pp 465-473, 1993

[*Kong 92*] Seong-Gong Kong, B.Kosko, *Adaptative Fuzzy Systems for Backing up a truck-and-trailer* , IEEE transactions on Neural Networks, vol 3, N2, mars 1992

[*Le Cun 87*] Y. Le Cun, *Modèles connexionnistes de l'aprentissage*, thèse de l'Université Paris 6, juin 1987.

[*Lee 92*] D.M.A. Lee, W.H. ElMaraghy, *A Neural Network Solution for Biped Gait Synthesis*, Proceedings of Int. Joint Conference on Neural Networks, Boston, Vol II, pp. 763-767,1992

[*Li 91*] Q. Li, A. Takanishi, I. Kato*A Biped Walking Robot Having A ZMP Measurement System Using UniversalForce-Moment Sensors*,IEEE International Workshop on Intelligent Robots and Systems, IROS '91,pp 1568-1573

[*Mart 91*] J. M Martinez, M. Houkari, Ch. Parey, C. Barret, P. Grizzo *La Rétropropagation sous l'Angle de la Théorie du Contrôle*, Proceedings of the Fourth International Conference on Neural Networks and Their Applications,Nîmes, 1991

[*Mill 93*] P. C. Y Chen, J. K. Mills, K. C. Smith*On the Dynamics of a Neural Network for Robot Trajectory Tracking*, Proceedings of IEEE International Conference on Intelligent Robots and Systems, pp 252-258,1993

[*Mill 93*] P. C. Y Chen, J. K. Mills, K. C. Smith*On the Dynamics of a Neural Network for Robot Trajectory Tracking*, Proceedings of IEEE International Conference on Intelligent Robots and Systems, pp 252-258,1993

[*Mill 93*] P. C. Y Chen, J. K. Mills, K. C. Smith*On the Dynamics of a Neural Network for Robot Trajectory Tracking*, Proceedings of IEEE International Conference on Intelligent Robots and Systems, pp 252-258,1993

[*Min 92*] B. Min and Z. Bien *Neural Computation for Adaptive Gait Control of the quadruped over Rough Terrain*, Proceedings of IEEE International Conference on Robotics and Automation, pp 2612-2617,1992

[*Nagy 92*] P. V. Nagy, W. L. Whittaker and Subhas Desa *A Walking Prescription for Statically-Stable Walkers Based on Walker/Terrain Interaction*, Proceedings of IEEE International Conference on Robotics and Automation, pp 149-1156,1992

[*Narr 91*] K. S. Narendra, K. Parthasarathy *Gradient Methods for the Optimization of Dynamical Systems Containing Neural Networks*, Proceedings of IEEE Transactions on Neural Networks,Vol. 2. No. 2, pp 252-262, 1991

[*Nerr 93*] O. Nerrand, P. Roussel-Ragot,L. Personnaz, G. Dreyfus *Neural Networks and Non-Linear Adaptive Filtering : Unifying Concepts and New Algorithms*, Neural Computation , vol. 5, 99. pp 165-197,1993

[*Nerr 94*] O. Nerrand, P. Roussel-Ragot, D. Urbani, L. Personnaz, G. Dreyfus *Training Recurrent Neural Networks: Why and How? An Illustration in Dynamical Process Modeling*, IEEE Transactions on Neural Networks, special issue on Recurrent Networks, à paraître,1994

[*Nguy 91*] D. H. Nguyen et B. Widrow *Neural Networks for Self-Learning Control Systems*, IEEE Workshop on Industrial Applications of Neural Networks, pp 18-23, Ascona Ticino,1991

[*Noss 92*] J. Nossek, G. Seiler, T. Roska, L.Chua *Cellular Neural Networks: Theory and Circuit Design*, International Journal of Circuit Theory and Aplications, Vol. 20, pp 533-553, 1992

[*Novi 91*] P. Novikoff, *De l'Intéraction Perception-Navigation en Robotique Mobile*, thèse de l'Université Paris 6, Juillet 1991.

[*Ouez 90*] Fathi B. Ouezdou, *Outils d'Aide à la Conception de Robots à Locomotion Articulée*, thèse de l'Université Paris 6,Juin 1990.

[*Ouss 94*] Y. Oussar, *Apprentissage de règles floues pour le contrôle dynamique d'un manipulateur rapide*, Rapport de stage ,Juillet 1994.

[*Pham 94*] D. T. Pham et X. Liu *A Comparaison of Three Types of Neural Networks for System Identification*,Proceedings of IMACS International Symposium on Signal Processing, Robotics and Neural Networks,pp 568-571, Lille, 1994

[*Pitt 43*] W.Mc Culloch, W. Pitts *A Logical calculusfor the ideas immament in nervous activity*, ull. Math. Biopysics, 5,pp 115-133, 1943

[*Riva 93*] I. Rivals, L. Personnaz, G. Dreyfus et D. Canas *Real-Time Control of an Autonomous Vehicule : A Neural Network Approach to the Path Following Problem*, Sixth International Conference on Neural Networks and their Applications,Nîmes, 1993

[*Rumm 86*] D.E Rummelhart, J.L. Mc Clelland, *Parallel Distributed Processing*, The MIT Press, Vol 1, pp. 318-362, 1986

[*Sala 92a*] A. W. Salatian, Y. F. Zheng *Gait Synthesis for a Biped Robot Climbing Sloping Surfaces Using Neural Networks. Part I: Static Learning*, Proceedings of IEEE International Conference on Robotics and Automation, pp 2601-2605,1992

[*Sala 92b*] A. W. Salatian, Y. F. Zheng *Gait Synthesis for a Biped Robot Climbing Sloping Surfaces Using Neural Networks. Part II: Dynamic Learning*, Proceedings of IEEE International Conference on Robotics and Automation, pp 2607-2611,1992

[*Schi 93*] W. H. Schiffmann and H. W. Geffers *Adaptive Control of Dynamic Syustems by Back Propagation Networks*, Neural Networks, Vol 6, pp 517-524, 1993

[*Schw 92*] H. Schwenk *Contrôle des Robots par réseaux de Neurones:Application à l'Equilibre Dynamique d'un bipède Plan* Rapport de stage, Université Paris 6-Université de Karlsruhe 1992

[*Soba 88*] D. J. Sobajic, Jun-Jie Lu, Yoh-Han Pao, *Intelligent control of intelledex 605T robot manipulator*, Proceedings of IEEE International Conference on Neural Networks, pp II.633-II.640, 1988

[*Sun 93*] Chuen-Tsai Sun and Jyh-Shing Jang,*Fuzzy Modelling Based on Generalized Neural Networks and fuzzy Clustering Objective Functions*, Proceedings of the 30th Conference on Decision and Control, pp 2924-2929, 1991

[*Taba 92*] G. Tabary and I. Salaün,*Control of a Redundant Articulated System by Neural Networks*,Neural Networks,Vol. 5, pp 305-311, 1992

[*Taka 85*] A. Takanishi, M. Ishidam Y. Yamazaki, I. Kato *The Realization of Dynamic Biped Walking by the Biped Walking RobotWL-10 RD.*International Joint Conference on Advanced Robotics, 1985,pp 459-466

[*Taka 88*] A. Takanishi, et al., *Realization of Dynamic Biped Walking Stabilized with Trunk Motion*, Proceedings of ROMANSY, 1988

[*Taka 89*] A. Takanishi, M. Tochizawa, H. Karaki, I. Kato, *Dynamic Biped Walking Stabilized with Optimal Trunk and Waist Motion*, Proceedings of IEEE International Workshop on Intelligent Robots and Systems, IROS '89, pp. 187-192

[*Taka 90*] A. Takanishi, et al., *A Control Method for Dynamic Biped Walking under Unknown External Force*, Proceedings of IEEE International Workshop on Intelligent Robots and Systems, IROS , pp. 795-801,1990

[*Taka 92*] Q. Li, A. Takanishi, I. Kato, *Learning Control of Compensative Trunk Motion for Biped Walking Robot based on ZMP Stability Criterion*, Proceedings of IEEE Int. Conf. on Intelligent Robots and Systems, pp. 597-603,1992

[*Taka 93*] N. Takanashi,*6 D.O.F Manipulators Absolute Positionning Accuracy Improvement Using a Neural Network*,IEEE Internationnal Workshop on Intelligent Robots and Systems, pp 935-640, 1990

[*Touz 92*] C. Touzet,O. Sarzeaud *Application d'un Algorithme d'Apprentissage par Pénalité Récompense à la Génération de Formes Locomotrices Hexapodes* AFCET-COGNITION - Journées de Rochebrune, Janvier 1992

[*Touz 93*] C. Touzet, *Apprentissage par Renforcement Neuronal: Avantages et Limitations* ,Proceedings of 2^{nd} European Congress on Systems Science, Prague, Octobre 1993

[*Tso 92*] S. K. Tso, Y. X. Ma *Discrete Learning Control for Robot Using Partial Model Knowledge*, Proceedings of IEEE International Conference on Robotics and Automation, pp 2039-2044,1992

[*Tsu 87*] K. Tsutsumi, M. Matsumoto*Neural Computation and Learning Strategy for Manipulator Position Control*,International Joint Conference on Neural Networks 1987, San Diego

[*Vuko 70*] A. Vukobratović, A. Frank, D. Juricic, *On the Stability of Biped Locomotion*, IEEE Transactions on Biomedical Engeneering, Vol. BME-17, No 1, 1970

[*Wada 93*] Y. Wada and M. Kawato, *A Neural Network Model for Arm Trajectory Formation Using Forward and Inverse Dynamics Models*, Neural Networks, Vol 6,pp 919-932,1993,

[*Widr 90*] B. Widrow and M. A. Lehr,*30 Years of Adaptive Neural Networks: Perceptron, Madaline and Backpropagation*, Proceedings of the IEEE, Special issue on Neural Networks: Theory and Modelling, pp 1415-1440, 1990

[*Widr 92*] V. V. Tolat, B. Widrow *A adaptive "Broom Balancer" with Visual Inputs*,Proceedings of IEEE International Conference on Neural Networks, pp II.641-II.647, 1988

[*Widr 93*] B. Widrow and M. A. Lehr,*30 Adaptive Neural Networks and their Aplications*, International Journal of Intelligent Systems , Vol. 8, pp 453-507, 1993

[*Yama 93*] Fu-Chuang Chen *Application of Learning Type Feedforward feedback Neural Network Controller to Dynamic Systems*, Proceedings of IEEE International Conference on Intelligent Robots and Systems, IROS'93, pp 225-231,1993

[*Yama 93*] T. Yamada and T. Yabuta, *Neural Network Controller Using Autotuning Method for Nonlinear Functions*, IEEE Transactions on Neural Networks, Vol 3,No4, 1992

[*Zhen 90b*] Y.F. Zheng, J. Shen *Gait Synthesis for the SD-2 Bipede Robot to Climb Sloping Surface*, IEEE Transactions on Robotics and Automation,Vol. 6, No 1, 1990

[*Zhen 90*] Yuan F. Zheng *A Neural Gait Synthesizer for Autonomous Biped Robots*, Proceedings of IEEE International Workshop on Robotics and Systems, pp 601-608,1990

[*Zoma 93*] A. Y. Zomaya ,M. E. Suddaby, A. S. Morris *Direct Neuro-Adaptive Control of robot Manipulators*, Proceedings of IEEE International Conference on Robotics and Automation, pp 1902-1906,1992

Annexe A

Valeurs des poids après apprentissage pour la commande du robot mobile

A.1 Par apprentissage direct (réseau 3-9-2)

```
couche 2
        neurone   1  --  entree +0.000000, sortie +0.000000
                w12(1,1)= -3.501704
                w12(2,1)= -2.227631
                w12(3,1)= -0.991390
                w12(seuil,1)= +0.080573
        neurone   2  --  entree +0.000000, sortie +0.000000
                w12(1,2)= +1.868638
                w12(2,2)= -2.079537
                w12(3,2)= -0.074858
                w12(seuil,2)= +1.205666
        neurone   3  --  entree +0.000000, sortie +0.000000
                w12(1,3)= -1.133928
                w12(2,3)= +1.331957
                w12(3,3)= -1.906851
                w12(seuil,3)= -0.081558
        neurone   4  --  entree +0.000000, sortie +0.000000
                w12(1,4)= -2.089738
                w12(2,4)= -2.603283
                w12(3,4)= +0.564539
                w12(seuil,4)= +1.279456
        neurone   5  --  entree +0.000000, sortie +0.000000
                w12(1,5)= +1.337994
                w12(2,5)= +0.689353
                w12(3,5)= -0.579654
                w12(seuil,5)= +0.570937
        neurone   6  --  entree +0.000000, sortie +0.000000
```

```
                w12(1,6)= +2.176460
                w12(2,6)= +1.363017
                w12(3,6)= -0.283766
                w12(seuil,6)= +0.356553
        neurone   7  --  entree +0.000000, sortie +0.000000
                w12(1,7)= -2.285600
                w12(2,7)= +2.509655
                w12(3,7)= +0.002574
                w12(seuil,7)= +0.906294
        neurone   8  --  entree +0.000000, sortie +0.000000
                w12(1,8)= -1.864766
                w12(2,8)= +1.736262
                w12(3,8)= +0.083510
                w12(seuil,8)= -1.279988
        neurone   9  --  entree +0.000000, sortie +0.000000
                w12(1,9)= -2.073928
                w12(2,9)= -2.205100
                w12(3,9)= -0.452388
                w12(seuil,9)= -1.220194
-------------------------------------------------------
couche 3
        neurone   1  --  entree +0.000000, sortie +0.000000
                w23(1,1)= +1.245944
                w23(2,1)= -1.318505
                w23(3,1)= +0.774724
                w23(4,1)= +2.793247
                w23(5,1)= -0.422407
                w23(6,1)= +2.432621
                w23(7,1)= -3.386203
                w23(8,1)= +2.119898
                w23(9,1)= -2.681163
                w23(seuil,1)= +0.084936
        neurone   2  --  entree +0.000000, sortie +0.000000
                w23(1,2)= -2.228786
                w23(2,2)= +2.685873
                w23(3,2)= -0.019097
                w23(4,2)= -2.029540
                w23(5,2)= -1.655292
                w23(6,2)= -0.123438
                w23(7,2)= +2.151811
                w23(8,2)= -0.984774
                w23(9,2)= +2.759127
                w23(seuil,2)= +0.413760
```

A.2 Par apprentissage indirect hors-ligne (réseau 3-9-2)

```
couche 2
        neurone   1  --  entree +0.000000
                w12(1,1)= -0.723065
                w12(2,1)= +1.154869
                w12(3,1)= -1.228074
                w12(seuil,1)= +0.050198
        neurone   2  --  entree +0.000000
                w12(1,2)= +1.191010
                w12(2,2)= -1.610935
                w12(3,2)= -0.597496
                w12(seuil,2)= -0.023970
        neurone   3  --  entree +0.000000
                w12(1,3)= +0.410392
                w12(2,3)= -0.107543
                w12(3,3)= -1.685538
                w12(seuil,3)= +0.005864
        neurone   4  --  entree +0.000000
                w12(1,4)= -0.436845
                w12(2,4)= -0.527577
                w12(3,4)= -1.539789
                w12(seuil,4)= -0.056657
        neurone   5  --  entree +0.000000
                w12(1,5)= +0.297137
                w12(2,5)= -0.664976
                w12(3,5)= -1.120319
                w12(seuil,5)= +0.012673
        neurone   6  --  entree +0.000000
                w12(1,6)= +0.406139
                w12(2,6)= -1.307570
                w12(3,6)= -1.582946
                w12(seuil,6)= +0.075353
        neurone   7  --  entree +0.000000
                w12(1,7)= +1.359773
                w12(2,7)= +1.500983
                w12(3,7)= -0.521676
                w12(seuil,7)= +0.035080
        neurone   8  --  entree +0.000000
                w12(1,8)= -0.564598
                w12(2,8)= -0.896487
                w12(3,8)= -0.160246
```

```
                w12(seuil,8)= +0.018847
        neurone   9  --  entree +0.000000
                w12(1,9)= -1.545528
                w12(2,9)= +0.839965
                w12(3,9)= -0.743622
                w12(seuil,9)= -0.021736

------------------------------------------------------------
couche 3

        neurone   1  --  entree +0.000000
                w23(1,1)= +0.871386
                w23(2,1)= +0.705845
                w23(3,1)= +0.810549
                w23(4,1)= +1.477265
                w23(5,1)= +0.374727
                w23(6,1)= -0.601673
                w23(7,1)= +0.692437
                w23(8,1)= +1.042243
                w23(9,1)= +0.564070
                w23(seuil,1)= -0.004699
       neurone   2  --  entree +0.000000
                w23(1,2)= +0.489709
                w23(2,2)= -0.294968
                w23(3,2)= -1.301873
                w23(4,2)= +0.348595
                w23(5,2)= -1.621301
                w23(6,2)= -1.105606
                w23(7,2)= -1.280132
                w23(8,2)= +0.476986
                w23(9,2)= -1.300904
                w23(seuil,2)= -0.004536
```

A.3 Par apprentissage indirect en-ligne sur simulateur

```
couche 2

        neurone   1  --  entree +0.000000
                w12(1,1)= -0.921255
                w12(2,1)= +0.994591
                w12(3,1)= -1.538290
                w12(seuil,1)= -0.086154
```

```
        neurone   2  --  entree +0.0000000
                w12(1,2)= +1.170383
                w12(2,2)= -1.621621
                w12(3,2)= -0.732577
                w12(seuil,2)= +0.058734
        neurone   3  --  entree +0.000000
                w12(1,3)= +0.570560
                w12(2,3)= +0.385995
                w12(3,3)= -2.052435
                w12(seuil,3)= +0.169975
        neurone   4  --  entree +0.000000
                w12(1,4)= -0.839715
                w12(2,4)= -0.470524
                w12(3,4)= -1.490873
                w12(seuil,4)= +0.030061
        neurone   5  --  entree +0.000000
                w12(1,5)= +0.495849
                w12(2,5)= -0.274613
                w12(3,5)= -1.441139
                w12(seuil,5)= +0.157394
        neurone   6  --  entree +0.000000
                w12(1,6)= +0.818888
                w12(2,6)= -1.056968
                w12(3,6)= -1.435846
                w12(seuil,6)= -0.306248
        neurone   7  --  entree +0.000000
                w12(1,7)= +1.286090
                w12(2,7)= +1.735243
                w12(3,7)= -1.049188
                w12(seuil,7)= +0.265697
        neurone   8  --  entree +0.000000
                w12(1,8)= -0.936837
                w12(2,8)= -0.684941
                w12(3,8)= -0.259147
                w12(seuil,8)= +0.063222
        neurone   9  --  entree +0.000000
                w12(1,9)= -1.347320
                w12(2,9)= +0.982884
                w12(3,9)= -1.368425
                w12(seuil,9)= -0.668708

-------------------------------------------------------------
couche 3
```

```
        neurone   1  --  entree +0.000000
                w23(1,1)= +1.423134
                w23(2,1)= +0.638445
                w23(3,1)= +0.980350
                w23(4,1)= +1.671976
                w23(5,1)= +0.493188
                w23(6,1)= -0.642322
                w23(7,1)= +0.798575
                w23(8,1)= +1.258252
                w23(9,1)= +1.346807
                w23(seuil,1)= +0.087882
        neurone   2  --  entree +0.000000
                w23(1,2)= +0.184954
                w23(2,2)= -0.242906
                w23(3,2)= -1.685603
                w23(4,2)= +0.359647
                w23(5,2)= -1.737275
                w23(6,2)= -0.958843
                w23(7,2)= -1.875649
                w23(8,2)= +0.683215
                w23(9,2)= -1.521405
                w23(seuil,2)= -0.137071
```

A.4 Par apprentissage indirect en-ligne sur le robot

A.4.1 Après le premier apprentissage

```
couche 2
        neurone   1  --  entree +0.000000
                w12(1,1)= -1.247891
                w12(2,1)= +1.095074
                w12(3,1)= +0.459951
                w12(seuil,1)= +0.118040
        neurone   2  --  entree +0.0000000
                w12(1,2)= +0.871893
                w12(2,2)= -0.994418
                w12(3,2)= +0.768162
                w12(seuil,2)= -0.114644
        neurone   3  --  entree +0.000000
                w12(1,3)= -1.985344
                w12(2,3)= +1.141364
                w12(3,3)= -1.191916
                w12(seuil,3)= -0.081949
        neurone   4  --  entree +0.000000
```

```
                w12(1,4)= -0.799609
                w12(2,4)= -0.913088
                w12(3,4)= +0.997954
                w12(seuil,4)= +0.008675
        neurone   5  --  entree +0.000000
                w12(1,5)= +1.522162
                w12(2,5)= +1.294514
                w12(3,5)= +0.874330
                w12(seuil,5)= +0.055672
        neurone   6  --  entree +0.000000
                w12(1,6)= -1.597675
                w12(2,6)= +1.808000
                w12(3,6)= +0.639672
                w12(seuil,6)= -0.042556
        neurone   7  --  entree +0.000000
                w12(1,7)= +1.630451
                w12(2,7)= -0.362007
                w12(3,7)= -2.359884
                w12(seuil,7)= +0.033331
        neurone   8  --  entree +0.000000
                w12(1,8)= -1.034568
                w12(2,8)= +0.333602
                w12(3,8)= -0.933808
                w12(seuil,8)= -0.053599
        neurone   9  --  entree +0.000000
                w12(1,9)= -1.371726
                w12(2,9)= +0.788733
                w12(3,9)= -0.387971
                w12(seuil,9)= +0.076949
-------------------------------------------------------
couche 3
        neurone   1  --  entree +0.000000
                w23(1,1)= -1.262644
                w23(2,1)= -1.009672
                w23(3,1)= +1.782792
                w23(4,1)= -1.612177
                w23(5,1)= -1.015736
                w23(6,1)= -0.138948
                w23(7,1)= +1.626661
                w23(8,1)= -0.016220
                w23(9,1)= +1.942994
                w23(seuil,1)= +0.052933
        neurone   2  --  entree +0.000000
                w23(1,2)= -1.913384
```

```
                w23(2,2)= -1.158559
                w23(3,2)= -0.384870
                w23(4,2)= -0.599047
                w23(5,2)= -0.956244
                w23(6,2)= +0.857367
                w23(7,2)= -2.314380
                w23(8,2)= -1.267303
                w23(9,2)= +2.002006
                w23(seuil,2)= +0.013711
```

A.4.2 Après le deuxième apprentissage

```
couche 2
        neurone   1  --  entree +0.000000
                w12(1,1)= -1.141339
                w12(2,1)= +1.145727
                w12(3,1)= +0.346131
                w12(seuil,1)= +0.085889
        neurone   2  --  entree +0.000000
                w12(1,2)= +0.939689
                w12(2,2)= -0.962537
                w12(3,2)= +0.722498
                w12(seuil,2)= -0.122138
        neurone   3  --  entree +0.000000
                w12(1,3)= -1.958023
                w12(2,3)= +1.106431
                w12(3,3)= -1.441932
                w12(seuil,3)= -0.004864
        neurone   4  --  entree +0.000000
                w12(1,4)= -0.790525
                w12(2,4)= -0.887573
                w12(3,4)= +1.105576
                w12(seuil,4)= -0.007324
        neurone   5  --  entree +0.000000
                w12(1,5)= +1.530664
                w12(2,5)= +1.287392
                w12(3,5)= +0.854750
                w12(seuil,5)= +0.126433
        neurone   6  --  entree +0.000000
                w12(1,6)= -1.645146
                w12(2,6)= +1.789452
                w12(3,6)= +0.784412
                w12(seuil,6)= +0.016628
        neurone   7  --  entree +0.000000
```

```
                w12(1,7)= +1.751492
                w12(2,7)= -0.350591
                w12(3,7)= -2.886734
                w12(seuil,7)= -0.027841
        neurone   8  --  entree +0.000000
                w12(1,8)= -0.966868
                w12(2,8)= +0.337581
                w12(3,8)= -1.119435
                w12(seuil,8)= -0.009160
        neurone   9  --  entree +0.000000
                w12(1,9)= -1.460341
                w12(2,9)= +0.754151
                w12(3,9)= -0.313292
                w12(seuil,9)= +0.039404
-------------------------------------------------------
couche 3
        neurone   1  --  entree +0.000000
                w23(1,1)= -1.307625
                w23(2,1)= -1.065540
                w23(3,1)= +1.884200
                w23(4,1)= -1.690037
                w23(5,1)= -1.108793
                w23(6,1)= -0.246600
                w23(7,1)= +1.907510
                w23(8,1)= +0.092798
                w23(9,1)= +1.947879
                w23(seuil,1)= +0.087009
        neurone   2  --  entree +0.000000
                w23(1,2)= -1.800927
                w23(2,2)= -1.101242
                w23(3,2)= -0.412815
                w23(4,2)= -0.439876
                w23(5,2)= -0.915873
                w23(6,2)= +1.000183
                w23(7,2)= -2.677028
                w23(8,2)= -1.332278
                w23(9,2)= +2.038303
                w23(seuil,2)= -0.002070
```

A.4.3 Après le troisième apprentissage

```
couche 2
        neurone   1  --  entree +0.000000
                w12(1,1)= -1.219313
```

```
        w12(2,1)= +1.148808
        w12(3,1)= +0.104437
        w12(seuil,1)= +0.147420
neurone   2  --  entree +0.000000
        w12(1,2)= +0.957653
        w12(2,2)= -0.962784
        w12(3,2)= +0.665227
        w12(seuil,2)= -0.140885
neurone   3  --  entree +0.000000
        w12(1,3)= -1.924146
        w12(2,3)= +1.114656
        w12(3,3)= -1.586952
        w12(seuil,3)= -0.226358
neurone   4  --  entree +0.000000
        w12(1,4)= -0.808583
        w12(2,4)= -0.892904
        w12(3,4)= +1.110068
        w12(seuil,4)= +0.124707
neurone   5  --  entree +0.000000
        w12(1,5)= +1.553989
        w12(2,5)= +1.286352
        w12(3,5)= +0.836488
        w12(seuil,5)= +0.126031
neurone   6  --  entree +0.000000
        w12(1,6)= -1.563706
        w12(2,6)= +1.784328
        w12(3,6)= +1.005962
        w12(seuil,6)= +0.023241
neurone   7  --  entree +0.000000
        w12(1,7)= +1.614114
        w12(2,7)= -0.330399
        w12(3,7)= -3.385240
        w12(seuil,7)= -0.274845
neurone   8  --  entree +0.000000
        w12(1,8)= -0.910648
        w12(2,8)= +0.344829
        w12(3,8)= -1.281440
        w12(seuil,8)= -0.211968
neurone   9  --  entree +0.000000
        w12(1,9)= -1.468110
        w12(2,9)= +0.753620
        w12(3,9)= -0.181680
        w12(seuil,9)= +0.048121
```

```
-----------------------------------------------------------
couche 3

        neurone   1  --  entree +0.000000
                w23(1,1)= -1.303478
                w23(2,1)= -1.104927
                w23(3,1)= +1.987708
                w23(4,1)= -1.732514
                w23(5,1)= -1.172579
                w23(6,1)= -0.265901
                w23(7,1)= +2.051366
                w23(8,1)= +0.187796
                w23(9,1)= +1.971389
                w23(seuil,1)= +0.029691
        neurone   2  --  entree +0.000000
                w23(1,2)= -1.842621
                w23(2,2)= -0.938755
                w23(3,2)= -0.642426
                w23(4,2)= -0.264940
                w23(5,2)= -0.728572
                w23(6,2)= +1.040004
                w23(7,2)= -3.123444
                w23(8,2)= -1.578552
                w23(9,2)= +1.923349
                w23(seuil,2)= -0.029967
```