UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

MÉMOIRE PRÉSENTÉ À

L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE

DE LA MAITRISE EN GÉNIE ÉLECTRIQUE

PAR

HICHAM CHAOUI

CONCEPTION ET COMPARAISON DE LOIS DE COMMANDE ADAPTATIVE À BASE DE RÉSEAUX DE NEURONES POUR UNE ARTICULATION FLEXIBLE AVEC NON-LINÉARITÉ DURE

Décembre 2002

REMERCIEMENTS

En premier lieu, j'exprime toute ma gratitude à mon directeur de recherche, le professeur Pierre Sicard, pour l'encadrement sans limite et le soutien financier qu'il a eu à m'apporter tout au long de ma formation à l'université du Québec à Trois-Rivières.

Je remercie mes collègues pour leur collaboration ainsi que les membres du laboratoire de commande.

Je remercie sincèrement tous ceux qui, de près ou de loin, ont contribué à ce travail, et qui m'ont aidé à surmonter moralement toutes les nombreuses difficultés survenues au cours de ces années.

Je profite de cette occasion, pour remercier les membres de la famille Desrosiers sans oublier ma famille, particulièrement ma sœur, pour ses précieux conseils.

Enfin, j'exprime toute ma reconnaissance à mes parents qui m'ont apporté un soutien moral ainsi que financier et qui m'ont encouragé à poursuivre mes études au Québec.

RÉSUMÉ

Pour résoudre les problèmes associés aux effets des non-linéarités des termes de friction dans une articulation flexible, le développement d'une loi de commande est requis. Le projet consiste à concevoir une loi de commande et à évaluer son comportement en considérant la flexion et les non-linéarités des phénomènes de friction dans un système de positionnement afin d'améliorer ses performances dynamiques.

L'approche de commande utilisée est basée sur la théorie de réseau de neurones artificiels. La loi de commande comporte deux réseaux de neurones: le premier a pour rôle de générer un signal d'anticipation pour le système de façon à pouvoir représenter le plus fidèlement possible une fonction du comportement inverse de l'articulation, le deuxième réseau corrige les erreurs laissées par le premier et assure une stabilité interne ainsi que celle du signal de sortie. Pour comparer les structures conçues, on établit plusieurs modes d'apprentissage pour les deux réseaux.

La validation des résultats de simulation des modèles est faite sur le logiciel Simulink™ sous Matlab®, qui nous permet de simuler le comportement du système dans le temps. Les résultats obtenus montrent que les effets de la flexion et des non-linéarités de la friction ont été compensés.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES SIGLES

DSP	Processeurs numériques de signaux (*Digital Signal Processor*).
LMS	Méthode des moindres carrés moyens (*Least Mean Squares*).
P	Proportionnel.
PD	Proportionnel Dérivatif.
PI	Proportionnel Intégral.
PID	Proportionnel Intégral Dérivatif.
RLS	Méthode des moindres carrés Récursifs (*Recursive Least Squares*).
RNA	Réseaux de Neurones Artificiels.
RNR	Réseaux de Neurones Récurrents.
VLSI	(ITGE) Intégration à Très Grande Échelle (*Very Large Scale Integration*).

LISTE DES SYMBOLES

a	Constante du calcul de la dérivée de la fonction d'activation.
b	Biais.
C	Vecteur des forces/couples centrifuges et de Coriolis.
e	Erreur d'estimation.
f	Gain pour la mise à jour des paramètres.
F	Fonction d'activation.
F	Vecteur des forces/couples de frottement.
F_c	Coefficient de frottement de Coulomb.
F_{comp}	Force de compensation de friction.
F_F	Force de frottement totale.
F_{RM}	Force/Couple de friction côté moteur.
F_{RC}	Force/Couple de friction côté charge.
F_s	Coefficient de frottement statique.
F_v	Coefficient de frottement visqueux.
g	Vecteur de gains pour la mise à jour des paramètres.
g	Vecteur des forces/couples de gravité.
I	Matrice identité.
J	Inertie.
J_c	Inertie de la charge.
J_M	Inertie du moteur.
J	Matrice Jacobienne.
k	Constante de rigidité en torsion de l'élément de transmission.
k ou n	Instant d'échantillonnage.

K	Constante pour le changement de variable sur l'inertie.
K	Nombre total d'observations.
K_p	Gain proportionnel du correcteur PD.
K_s	Matrice des coefficients de flexibilité.
K_v	Gain dérivatif du correcteur PD.
M	Dimension du vecteur de paramètres à estimer.
M	Matrice des masses.
N	Rapport de transmission.
P	Matrice de covariance.
p	Pôle en boucle fermée.
p	Variable de la rapidité de convergence.
q	Vecteur des variables articulaires.
S	Matrice racine carrée de covariance.
s	Opérateur de Laplace.
T	Période d'échantillonnage.
T	Transformation de Givens.
T_f	Matrice des perturbations et des termes dynamiques non modélisés.
t_L	Constante de temps de la charge.
t_M	Constante de temps du moteur.
u	Commande d'un procédé.
v	Vitesse linéaire.
V	Critère de la méthode des moindres carrés.
v_s	Vitesse caractéristique de frottement statique.
W	Vecteur de paramètres à identifier.
W	Poids du réseau de neurones.

X	Vecteur d'entrée du réseau de neurones.
y	Sortie mesurable d'un système.
Y	Vecteur de sortie du réseau de neurones.
Δw	Erreur de correction de la matrice des poids.
α	Accélération angulaire.
α	Coefficient d'apprentissage.
β	Rapport des variances des bruits sur la mesure et sur l'état du modèle d'estimation.
δ	Coefficient d'initialisation des matrices S ou P.
ε_ω	Limite inférieure de vitesse pour le calcul du couple de frottement.
ϕ	Vecteur de régression ou régresseur.
η	Efficacité du moteur.
η	Taux d'apprentissage.
η	Coefficient du momentum
η_α	Bruit sur l'accélération.
η_w	Bruit sur l'état du modèle d'estimation.
λ	Facteur d'apprentissage.
θ	Position angulaire.
τ	Couple de commande.
τ_a	Couple total appliqué excepté les couples de frottement.
τ_e	Couple externe.
τ_F	Couple de frottement total.
τ_T	Couple transmis.
ω	Vitesse angulaire.

ω_n	Pulsation naturelle de résonance.
ξ	Facteur d'amortissement.
$\bullet_T$	Grandeur reliée à l'élément de transmission.
$\bullet_C$	Grandeur reliée à la charge.
$\bullet_M$	Grandeur reliée au moteur.
$\bullet^T$	Transposée d'une matrice ou d'un vecteur.
$\bullet^*$	Valeur désirée.
$\approx$	Environ égal à.

Chapitre 1

INTRODUCTION

Les performances des systèmes d'entraînement à vitesse variable sont limitées par les propriétés des éléments mécaniques les constituant. La flexibilité articulaire a été identifiée comme la source principale de perte de performances pour plusieurs systèmes d'entraînement industriels [1]. La flexibilité est le résultat du choix des matériaux pour la flexion de l'arbre et du type de réducteur de vitesse employé. Par exemple, les réducteurs harmoniques sont employés pour leur faible encombrement et faible retour de dents, mais ils présentent une flexibilité notable inhérente au mode de fonctionnement du réducteur qui emploi des

matériaux flexibles. Les réducteurs planétaires permettent de réaliser des facteurs de réduction importants, mais présentent une caractéristique de flexion non-linéaire [2] causée par la déformation et le retour des dents de plusieurs engrenages en cascade.

L'effet principal de la flexion est l'apparition de modes vibratoires. Ces effets sont amplifiés pour les systèmes entraînant des charges lourdes, de grandes dimensions ou opérant à grande vitesse. En effet, un robot doit faire face à des contraintes comme le frottement de ses articulations, l'inertie de la charge transportée ainsi que sa masse en fonction de la gravité.

1.1- Problématique:

Dans l'industrie, la validation des lois de commande repose sur de très nombreuses simulations où l'on fait varier tous les paramètres incertains entrant dans la définition du système considéré. Le nombre de tests est une fonction exponentiellement croissante du nombre de paramètres incertains. Pour les systèmes de positionnement de haute performance, le nombre de paramètres incertains est tel que la validation nécessite de faire un choix parmi les configurations à tester, d'où le risque de ne pas détecter un problème de stabilité. Avec un bon sens physique, il était possible jusqu'à présent de minimiser, voir d'annuler le risque de ne pas détecter un tel cas. Les systèmes de positionnement actuels, pour des raisons de performance, sont de plus en plus flexibles. Donc, il est maintenant nécessaire de développer des lois de commande robuste en tenant compte des contraintes.

Le développement de lois de commande pour les systèmes d'entraînement avec éléments flexibles requiert le modèle dynamique. La formulation du modèle dynamique est l'élément clef contrôlant l'efficacité et la flexibilité de l'algorithme de simulation. Le modèle de la dynamique directe permet de calculer la position, la vitesse et l'accélération articulaires en

fonction du couple appliqué aux articulations. Le modèle de la dynamique inverse peut servir de base à la mise en oeuvre de divers contrôleurs.

D'autre part, certains paramètres mis en jeu dans les modèles d'articulations flexibles sont *a priori* incertains ou variables dans le temps. Ces caractéristiques nous amènent à utiliser une méthode de commande adaptative pour concevoir une loi capable de s'adapter en temps réel au comportement de l'articulation.

Certes, il y a eu plusieurs travaux de recherche sur la flexibilité [3][4][5] et la compensation de friction [2][6]. Cependant, peu de travaux traitent des deux à la fois [7] dans lequel le contrôleur comporte une anticipation basée sur les réseaux de neurones et un correcteur en logique flou en rétroaction. Cette recherche, tout comme la présente, vient contribuer au développement d'un contrôleur pour les articulations et éventuellement les robots à joints flexibles.

Plusieurs travaux ont été menés dans le domaine de la commande adaptative basée sur la théorie des réseaux de neurones et sur la commande des articulations flexibles. Par exemple, des correcteurs dédiés à la commande de robots avec joints flexibles et avec frottement de Coulomb ont été conçus dans [8]. Ce travail a présenté une loi de commande adaptative pour la compensation de friction basée sur l'identification des paramètres par un identificateur récursif aux moindres carrés et par le filtre de Kalman. L'implantation en ITGE de ces algorithmes a aussi été étudiée.

Le problème de compensation des phénomènes de flexion et des non-linéarités de la friction a été traité dans [7]. Les contrôleurs conçus sont basés sur un réseau de neurones qui représente le modèle inverse statique de l'articulation, et un correcteur avec mode de glissement comme

rétroaction. Le réseau utilisé est entraîné avec un apprentissage hors ligne. Les capacités d'apprentissage et d'adaptation des réseaux de neurones ont été utilisées pour l'approximation du modèle inverse qui était irréalisable. En plus, ils ont été utilisés pour leur parallélisme et leur facilité d'implantation.

Les correcteurs avec le mode de glissement sont avantageux par leur simplicité et une précision dans les résultats [9]. Kazuo et Toshio [10] ont obtenu de bons résultats à base de réseau de neurones artificiels, mais la friction ne peut pas être compensée avec un seul neurone de spécialisation lorsqu'on fait face à un modèle de frottement plus compliqué.

1.2- Objectifs:

Le but du projet est de concevoir une loi de commande pour un système de positionnement capable de donner un bon rendement à la charge et de s'adapter en temps réel avec le comportement de l'articulation en tenant compte de ses contraintes tel que les non-linéarités causées par le modèle de friction utilisé et la flexion due au ressort de torsion.

Les objectifs du travail portent sur l'évaluation de nouvelles structures de commande par réseaux de neurones artificiels basées sur des lois de commande pour articulations flexibles. Ces structures contiennent deux réseaux de neurones, le premier doit représenter le modèle inverse de l'articulation pour agir comme anticipation sur le système et le deuxième assure la stabilité en agissant en rétroaction.

1.3- Méthodologie:

Pour concevoir les contrôleurs, nous modélisons notre système en tenant compte de sa flexibilité et de ses non-linéarités. Nous établissons les équations du modèle inverse de

l'articulation pour concevoir le premier réseau de neurones qui agit sur le modèle comme un terme d'anticipation. Nous concevons le deuxième réseau qui représente un terme de rétroaction pour une correction d'erreur. Par la suite, on établit les algorithmes et les modes d'apprentissage hors ligne et en ligne des deux réseaux de neurones.

Les travaux vont donc porter sur la modélisation inverse statique d'une articulation flexible avec termes de frottement non-linéaire à l'aide de réseaux de neurones artificiels ainsi que l'établissement de modèles inverses de l'articulation et sur l'apprentissage hors ligne de ces modèles incluant le choix des données d'apprentissage. La structure générale est constituée d'un terme d'anticipation et d'une boucle de rétroaction de sortie. La difficulté appréhendée avec cette structure est l'assurance d'une stabilité interne. Ainsi, les travaux porteront sur la simulation des structures sous SIMULINK à partir des librairies existantes, sur le développement de stratégies d'apprentissage en ligne pour les réseaux de neurones et sur le choix de critères de mesure de performance.

Nous allons considérer un modèle s'approchant des comportements réels, c.-à-d. frottement sec inconnu ou mal connu – paramètres variables surtout pour la charge – coefficient de flexibilité, ce qui sera représentatif d'un système de positionnement de précision et de grande vitesse. La structure proposée, qui utilise les réseaux de neurones, permettra de faire les calculs à haut débit quand elle sera implantée sur une structure VLSI à cause de la structure parallèle des réseaux. Le principe de la commande présenté dans la figure 1.1 va être expliqué dans la section 4.5.2.

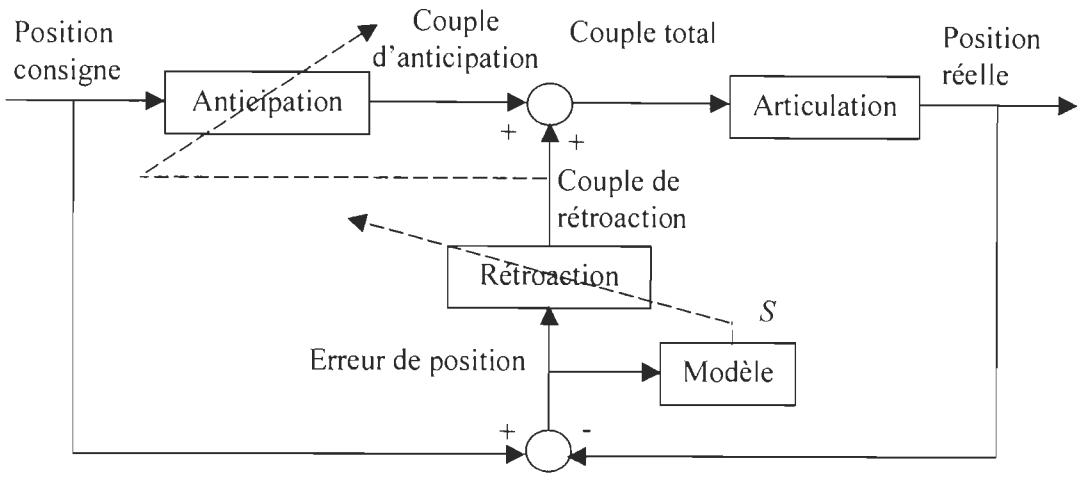

Fig. 1.1 : Principe de la commande.

1.4- Structure du mémoire:

Le deuxième chapitre présentera une synthèse des connaissances dans le domaine de la commande et de la modélisation des articulations flexibles avec des non-linéarités: les phénomènes de friction, les différentes méthodes et approches de commande utilisées pour des systèmes de positionnement à haute performance, ainsi que les méthodes d'estimation des paramètres.

Le troisième chapitre contiendra une étude sur les réseaux de neurones artificiels, leurs structures, types, fonctions d'activation, avantages, inconvénients et différentes méthodes d'apprentissage des réseaux de neurones artificiels. Une attention particulière sera apportée à l'état d'avancement des travaux dans le domaine de la recherche.

Dans le quatrième chapitre, le modèle et l'approche de commande de l'articulation flexible et ses propriétés seront décrits. Une stratégie de commande basée sur la modélisation inverse de l'articulation sous forme d'anticipation sera développée pour une estimation de couple à base de réseaux de neurones artificiels. Une commande adaptative avec rétroaction de

l'articulation, les structures du modèle et l'établissement des algorithmes d'apprentissage, hors ligne et en ligne seront traités.

Dans le cinquième chapitre, les résultats de simulation de la stratégie utilisée pour les différentes structures du modèle et les performances des lois de commande seront présentés.

Finalement, on présentera une discussion des principaux résultats et une présentation de la contribution du projet.

Chapitre 2

MODÉLISATION ET COMMANDE DES SYSTÈMES ÉLECTROMÉCANIQUES

2.1- Introduction:

Dans la plupart des problèmes de commande, l'objectif principal est d'atteindre la stabilité et de maintenir certaines performances malgré les évolutions et les incertitudes souvent pénalisantes pour le système à commander. Si, une telle commande présente de bonnes marges de stabilité ou une faible sensibilité à des incertitudes ou des variations des paramètres du système par rapport à son modèle nominal, elle peut être considérée comme robuste.

Dans la conception des systèmes de commande, la nécessité de hautes performances favorise l'utilisation de techniques sophistiquées. Une réponse rapide et stable est particulièrement critique dans les applications qui impliquent certains types de commande des entraînements.

Parmi les techniques de commande connues, on trouve la commande adaptative. Elle permet à la fois d'assurer la stabilité et une bonne réponse. Cette approche change les coefficients de l'algorithme de contrôle en temps réel pour compenser les variations dans l'environnement ou dans le système lui-même.

Par contre, ces techniques ne sont pas beaucoup utilisées à cause de la complexité de leur implantation. Mais après une évolution considérable de la micro-informatique, la situation a changé avec l'apparition des processeurs numériques de signaux (DSP). Ces derniers ont une rapidité et une capacité de calcul nettement supérieures à celles des microprocesseurs ordinaires. D'autre part, dans plusieurs applications, le DSP peut servir comme processeur unique dans le système, éliminant le besoin d'associer des circuits analogiques à des microprocesseurs d'usage général.

Un grand nombre de travaux de recherche s'orientent vers la commande adaptative [2][4], car les paramètres des modèles utilisés dans la commande des systèmes d'entraînement peuvent être inconnus ou variables dans le temps.

2.2- Modèles des systèmes électromécaniques:

Les caractéristiques ou les conditions d'utilisation d'une articulation dépendent de la structure du modèle, d'où la nécessité de développer son propre modèle qui pourrait satisfaire tous les critères pour pouvoir utiliser plusieurs des méthodes de commande déjà connues.

2.2.1- Flexibilité:

La conception d'un système de positionnement à haute performance plus rapide et plus léger demande que des compromis soient faits au niveau du choix des matériaux des actionneurs et des éléments de transmission. Par exemple, les réducteurs harmoniques sont très efficaces pour la transmission de couples élevés et présentent un faible retour de dent ainsi qu'un poids assez faible, mais ils possèdent une flexibilité inhérente. Les chercheurs ont montré l'effet déstabilisateur de la flexibilité sur les lois de commande qu'ils ont conçues pour des robots rigides [2]. En effet, les résultats de simulations, confirmés par des expériences, ont montré que plusieurs algorithmes de contrôle développés pour les robots rigides ne fonctionnent pas de manière satisfaisante en présence d'une rigidité réduite au niveau des articulations qui cause des oscillations. Ces dernières se produisent avec les lois de commande continues et discontinues conçues pour un robot rigide. Pour les cas donnant des résultats satisfaisants, les résultats sont critiques et ne sont pas robustes. Pour assurer la stabilité et de bonnes performances, la loi de commande doit prendre en compte la flexibilité. La figure 2.1 représente le modèle de l'articulation, incluant la flexibilité.

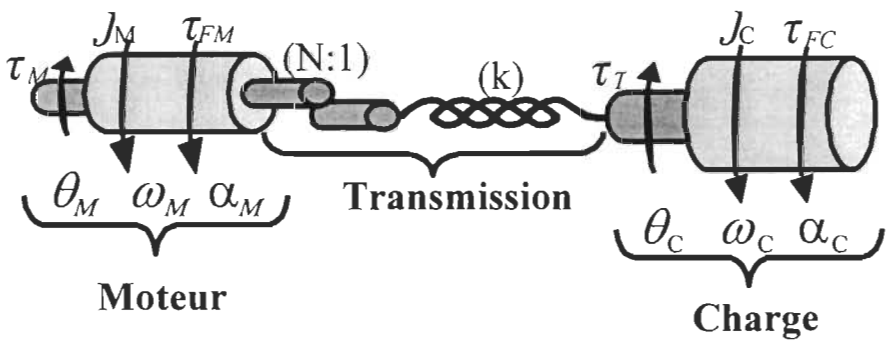

Fig. 2.1 : Modèle symbolique de l'articulation flexible.

Le modèle de l'articulation flexible proposé dans [1] est composé d'un moteur commandé en couple pilotant une charge par le biais d'une boîte de transmission équivalente à un ressort de torsion de caractéristique linéaire. Les indices M, C, et T désigneront respectivement les

valeurs reliées au moteur, à la charge et à la transmission. L'indice F représente les termes relatifs à la friction. Les variables k et N désignent la raideur du ressort de torsion et le rapport de transmission; elles seront supposées constantes et connues dans ce travail, mais ce n'est pas toujours le cas pour la constante de torsion k.

2.2.2- Friction:

Les chercheurs utilisent plusieurs modèles de frottements pour articulations. Dans chacun de ces modèles, le frottement est généralement une fonction exclusive de la vitesse. La plupart des auteurs utilisaient le modèle linéaire de frottement qu'on appelle frottement visqueux, en particulier ceux qui faisaient de la recherche exclusivement théorique dans le cadre des synthèses de lois de commande pour les articulations flexibles.

Les travaux les plus proches de la réalité physique sont ceux qui considèrent un frottement non-linéaire des articulations. Dans ce cas, les modèles utilisés sont souvent fondés sur celui de Coulomb, où le frottement est une constante en valeur absolue qui s'oppose au mouvement et qui est appelé frottement sec. Comme on constate expérimentalement qu'un couple supérieur au couple de frottement sec est nécessaire pour mettre l'articulation en mouvement, on introduit alors la notion du frottement statique. Ceci a amené des chercheurs à développer de nouveaux modèles représentant de façon plus précise les phénomènes réels. Pour une modélisation statique de l'articulation, la figure 2.2 représente un des modèles utilisés et simplifiés de la friction comprenant une bande morte autour de la vitesse nulle.

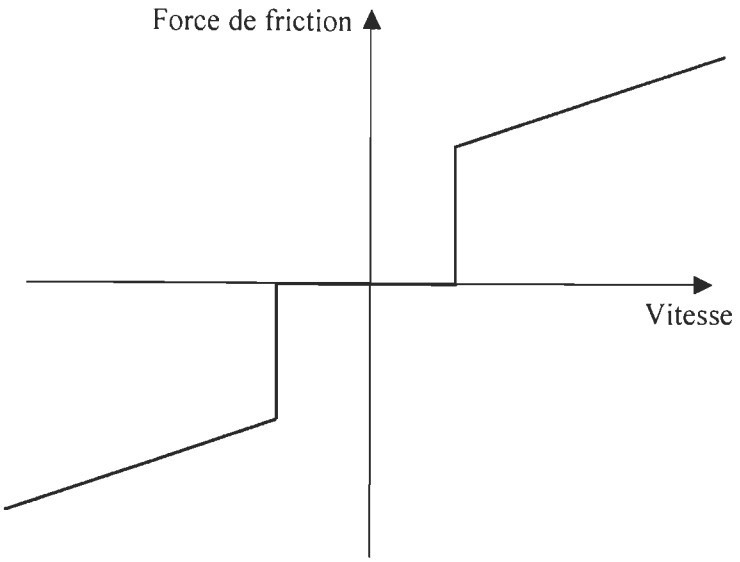

Fig. 2.2 : Modèle de friction simplifié.

Nous retrouvons plusieurs modèles pour la friction [3][6][7], tant des modèles statiques représentés par une caractéristique couple-déformation que des modèles dynamiques qui incorporent des termes supplémentaires pour caractériser les effets de mémoire dans les entraînements. Le modèle de Stribeck est largement employé et représente une forme complète de modèle statique. Le modèle de frottement proposé [8] à la figure 2.3 est décrit avec trois termes du phénomène de friction: de Coulomb, statique et visqueux. On peut écrire la force/couple de friction sous la forme suivante :

$$F_{F(t)} = F_c sign(v_{(t)}) + F_v v_{(t)} + F_s sign(v_{(t)}) e^{-(\frac{v_{(t)}}{v_s})^2} \qquad (2.3)$$

où:

F_c : Frottement de Coulomb.

F_s : Frottement statique.

F_v : Frottement visqueux.

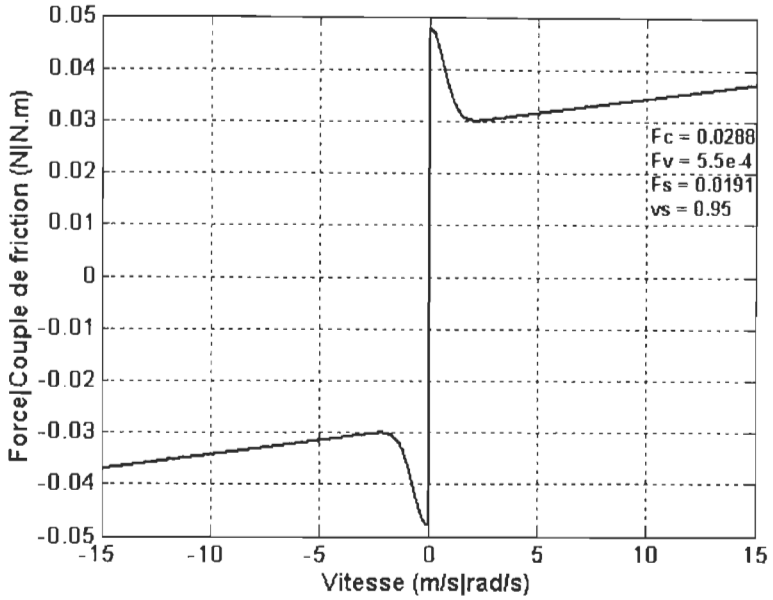

Fig. 2.3 : Modèle de Stribeck de la friction en fonction de la vitesse.

Ce modèle représente de plus près les phénomènes de friction rencontrés dans la pratique, mais ses non linéarités autour du point zéro causent parfois des oscillations importantes ce qui a poussé quelques chercheurs à utiliser des modèles simplifiés au coût d'une réduction en performance.

De plus, ce modèle ne présente pas un comportement réel à une vitesse faible parce que dans ce cas, le couple de frottement ne doit pas entraîner le moteur ou la charge. Pour résoudre ce problème et éviter les problèmes de stabilité associés, nous emploierons l'algorithme de la figure 2.4 qui fait en sorte que les phénomènes de friction restent comme contrainte et ne deviennent pas un générateur de mouvement [11].

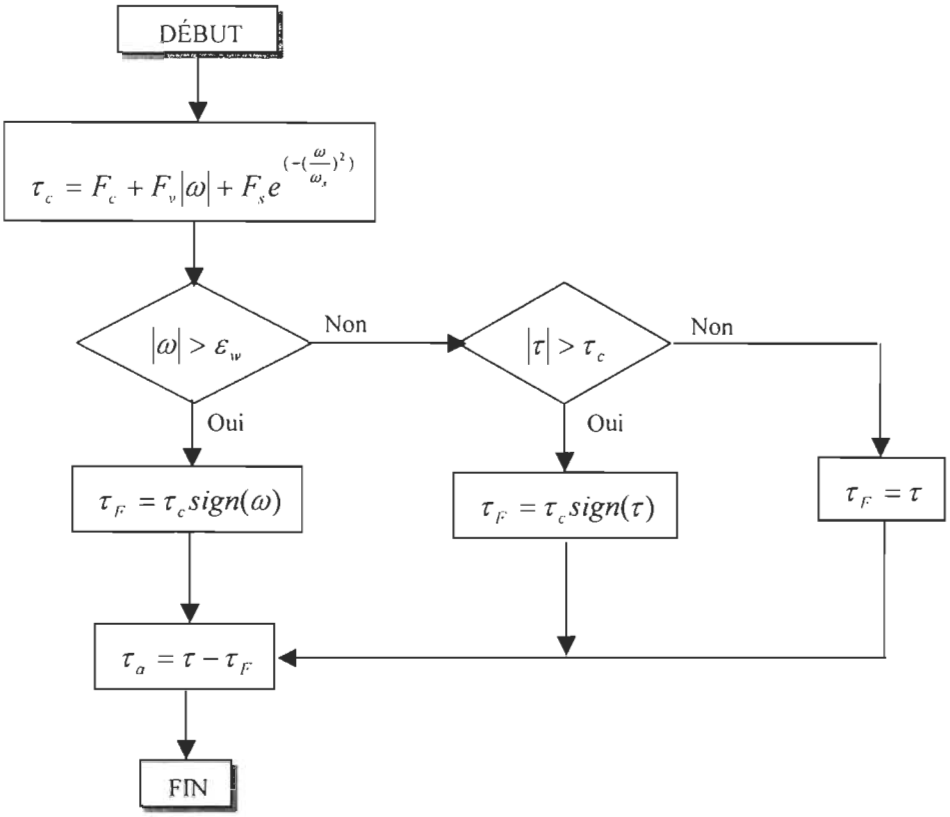

Fig. 2.4 : Algorithme de calcul des couples de frottement.

Dans la figure 2.4, τ_F, τ, ω et ε_w désignent respectivement le couple de frottement, le couple de commande, la vitesse et la limite de vitesse inférieure pour éviter les oscillations.

2.2.3- Modèle dynamique d'une articulation:

La figure 2.5 présente le modèle d'articulation proposé dans [12].

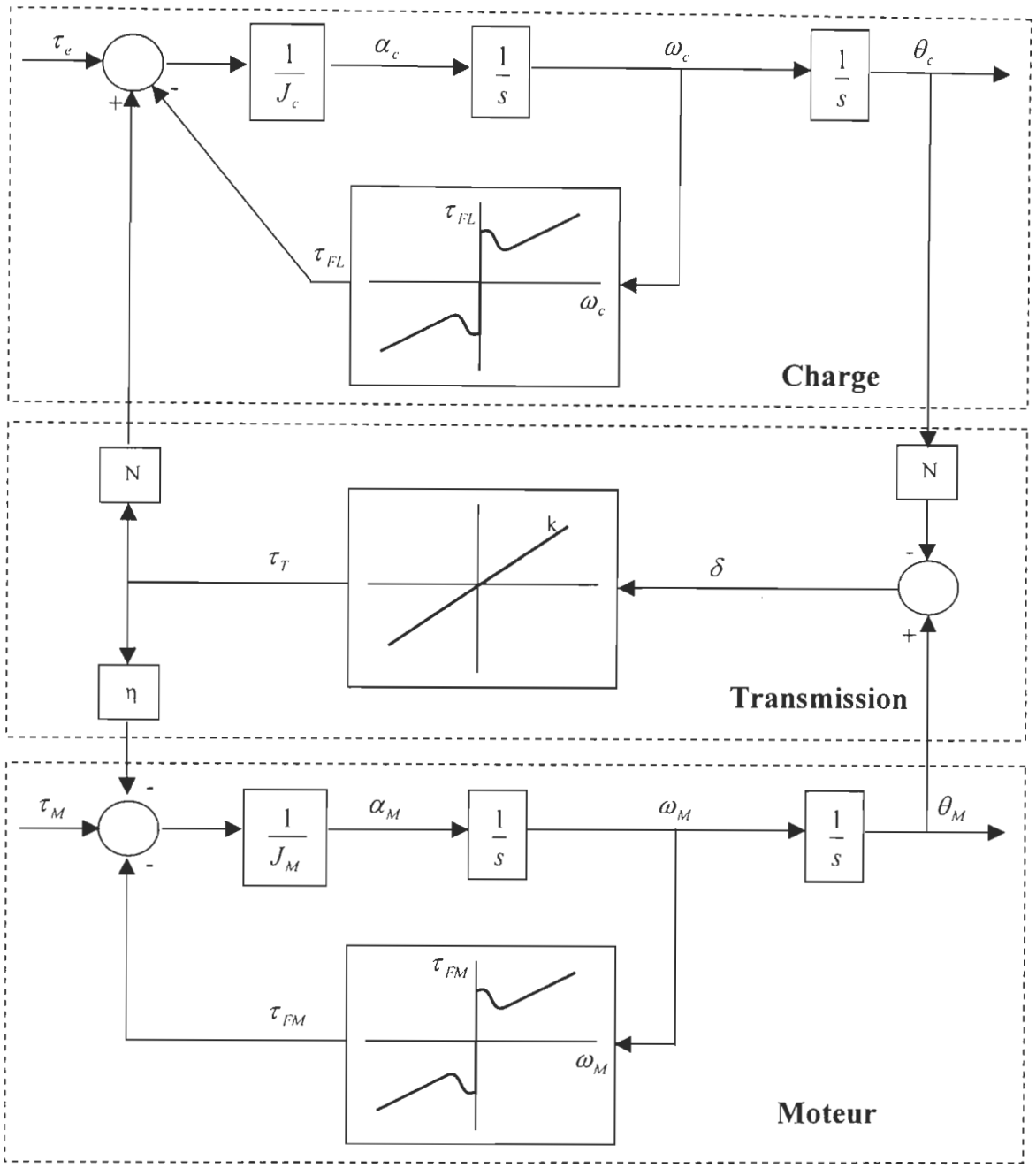

Fig. 2.5 : Modèle de l'articulation flexible.

Le modèle de l'articulation flexible contient un modèle du moteur et de la charge ainsi que le modèle de la boîte de transmission. On suppose que la caractéristique de ce dernier est linéaire exprimant le couple transmis à la charge sous la forme:

$$\tau_T = k(\theta_M - N\theta_C) \qquad (2.1)$$

Le modèle du moteur et de la charge sont affectés par des frottements de Coulomb, visqueux et statiques modélisés par les caractéristiques $\tau_{FL}(\omega_c)$ et $\tau_{FM}(\omega_M)$. On suppose aussi que le rapport de transmission N est égal à 1 et le rendement η est de 100%.

Les équations du modèle dynamique de l'articulation côté moteur et côté charge peuvent s'écrire sous la forme :

$$\tau = J_M\ddot{\theta}_M + F_{RM}(\dot{\theta}_M) + K(\theta_M - N\theta_c) \qquad (2.3)$$

$$J_c\ddot{\theta}_c + F_{RC}(\dot{\theta}_c) = NK(\theta_M - N\theta_c) \qquad (2.4)$$

où:

F_{RM} : Force/couple de friction côté moteur.

F_{RC} : Force/couple de friction côté charge.

τ : Couple appliqué à l'articulation.

K : Constante de torsion.

J_M : Inertie du moteur.

Jc : Inertie de la charge.

N : Rapport de réduction de vitesse de la boîte de transmission.

2.3- Commande des systèmes électromécaniques:

2.3.1- Effets de la flexibilité et de la friction sur la commande:

Les termes de frottement non-linéaire sont une autre source de perte de performance. Le frottement sec (de Coulomb) tend en effet à exciter les modes vibratoires et à réduire la précision du système ainsi que la qualité de la commande à faible vitesse. Les modèles de friction utilisés dans la conception des contrôleurs ont longtemps été limités à des modèles

très simples, ils représentent les effets qui sont faciles à modéliser. Des études ont démontré la présence d'autres phénomènes de friction non modélisés à basse vitesse, tel que les termes de friction de Stribeck. Une autre source de perte en performance qu'on rencontre est la flexibilité, elle peut causer une perte dans la stabilité interne et des oscillations.

2.3.2- Approches de commande:

Plusieurs lois de commande ont été proposées pour la commande des manipulateurs avec articulations flexibles [2][11]: techniques des perturbations singulières; linéarisation par rétroaction; commande robuste; commande adaptative, et techniques de passivité (incluant la commande par mode de glissement). La plupart des lois de commande qui ont été proposées requièrent la connaissance de l'état complet et même quelques fois la mesure de l'accélération. Parmi les méthodes de commande traitées dans la littérature, nous retrouvons les contrôleurs linéaires: proportionnel (P), proportionnel dérivatif (PD), proportionnels intégral (PI) et proportionnel intégral dérivatif (PID). Ces contrôleurs donnent de bons résultats pour les systèmes linéaires.

Une autre approche utilisée pour les systèmes linéaires invariants est la théorie de la commande H∞ [13]. Elle offre des outils mathématiques efficaces de synthèse des régulateurs afin de satisfaire les objectifs requis de robustesse en performance vis-à-vis de la qualité de la commande stabilisante. Les grands principes de cette approche sont introduits par Zames [14] et font appel au théorème du petit gain.

La commande H∞ nous permet d'inclure dans le modèle toutes les spécifications utiles telles que les consignes, les perturbations et les bruits de mesure ainsi que leurs points d'application, de sorte que la recherche du régulateur se présente comme la résolution d'un problème

d'optimisation bien posé qui consiste à minimiser ou à borner un transfert précis au sens de la norme H∞.

Finalement, la commande robuste H∞ permet de développer une solution optimale, mais une telle solution doit tenir compte d'une pondération appropriée. Toutefois, rien ne peut être garanti en ce qui concerne les performances temporelles du système légèrement perturbé.

En effet, la stabilité ne concerne que l'état asymptotique du système bouclé. Néanmoins, la commande du système doit aussi assurer des performances de robustesse. En d'autres termes, elle doit maintenir la stabilité en présence d'erreurs de modèle et réagir également face à des perturbations non mesurées.

Plusieurs recherches ont été menées sur la commande adaptative, mais souvent limitées dans leurs applications. Il est souvent nécessaire, dans ce type de commande, d'estimer certains paramètres du procédé ou de son modèle. Si ces paramètres sont constants, ils peuvent être identifiés hors ligne et être ensuite utilisés directement dans le calcul de la commande, et si ces paramètres sont variables, il faut les identifier en ligne.

Parmi les méthodes d'identification des paramètres d'un procédé, il y a: la méthode graphique, la méthode de décorrélation, et la méthode des moindres carrés [8].

La programmation des gains est une approche utilisée pour éliminer l'influence des variations de la dynamique du procédé lorsqu'on trouve une corrélation entre ces dernières et ses variables auxiliaires. Cette stratégie présente un avantage qui consiste dans la possibilité de

modifier rapidement les paramètres du correcteur. L'inconvénient majeur de cette stratégie est qu'elle n'est pas robuste.

Une autre stratégie permet d'adapter les paramètres du correcteur selon des spécifications définies par un modèle de référence qui décrit la réponse idéale de la sortie du procédé à un signal de commande. On retrouve aussi la structure des deux boucles, la boucle interne qui comprend le procédé et un correcteur avec rétroaction et la boucle externe qui ajuste les paramètres du correcteur.

D'excellents résultats peuvent être obtenus avec la commande adaptative avec modèle de référence. Elle permet d'utiliser un modèle simplifié lors de la conception et d'avoir une adaptation rapide pour des entrées connues. Cependant, elle présente des faiblesses comme des problèmes de stabilité, un temps de réponse lent et une difficulté pour compenser des grandes perturbations.

Il est souvent difficile de trouver une compensation efficace contre les effets de perturbations et la complexité de l'algorithme exige l'utilisation d'un processeur suffisamment rapide. Un autre inconvénient majeur de cette stratégie se situe au niveau de la conception du processus d'adaptation. Ce dernier doit détecter l'erreur à la sortie pour déterminer la façon avec laquelle doit se faire l'ajustement des coefficients du contrôleur. Il doit aussi rester stable quelles que soient les conditions. Le problème est qu'il n'y a aucune méthode théorique générale pour la conception de ce processus d'adaptation. D'ailleurs, la plupart des fonctions d'adaptation sont spécialement conçues pour des applications spécifiques.

Une autre stratégie de commande basée sur les régulateurs auto-syntonisants se distingue par les meilleures performances. Ils présentent une grande flexibilité dans le choix de l'algorithme d'identification et de celui de conception du régulateur. D'autre part, une connaissance préliminaire mineure du système à commander est requise. De plus, l'implantation pratique est relativement plus simple que dans le cas des autres techniques.

L'inconvénient de cette stratégie de commande est la convergence conditionnelle de l'algorithme d'identification des paramètres du système à commander. En effet, une condition nécessaire pour que les paramètres soient identifiables est que le signal de commande soit d'un ordre suffisant et soit actif. Pour s'assurer que cette condition soit respectée, il peut être nécessaire d'introduire un signal d'excitation ou d'utiliser un système de supervision afin de n'effectuer une mise à jour des valeurs estimées que lorsque le signal de commande est actif.

La commande par mode de glissement [15][16] est une des commandes robustes pour les systèmes non-linéaires. Pour le positionnement d'une articulation rigide, nous n'avons pas besoin de calculer la dérivée du signal de la vitesse, et la mesure de l'accélération et ses dérivées supérieures ne sont pas nécessaires. En plus, la stabilité asymptotique globale est garantie par cette méthode de commande pour la position et la vitesse du membre. La commutation du signal de commande peut exciter la fréquence naturelle de l'élément flexible du système.

Le réglage par mode de glissement présente plusieurs avantages tels que la robustesse et la grande précision. Les non-linéarités peuvent être traitées et rejetées comme des perturbations. Les systèmes de réglage par cette méthode sont applicables à de nombreux systèmes et

particulièrement appropriés pour les situations où la réponse du système original est instable ou bien si le système est soumis à de fortes perturbations externes.

Malgré que la commande par mode de glissement possède plusieurs avantages, cette méthode a des inconvénients comme la nécessité d'utilisation des dérivées de la variable à régler; le système est sensible face aux variations des paramètres et aux perturbations lors de la phase de convergence; la présence des discontinuités qui causent des oscillations à la sortie du système. De plus, l'instabilité du signal de commande peut causer une erreur stationnaire à la sortie.

Différentes solutions ont été proposées dans la littérature [15] afin de corriger ces faiblesses. Malheureusement, on ne peut compenser ces inconvénients sans avoir une perte de robustesse face aux spécifications dynamiques.

La plupart des lois de commande qui ont été proposées requièrent la connaissance de l'état complet du système de positionnement et même parfois de la mesure d'accélération, donc il est important de concevoir une loi de commande robuste qui demande seulement les informations de position et de vitesse des membres du système d'entraînement. Idéalement, la loi de commande donne une grande région d'attraction applicable et robuste dans le cas de l'incertitude des paramètres autonomes non-linéaires du système, d'une erreur du modèle, de la variation de la charge et d'une caractéristique de rigidité inconnue.

2.4- Conclusion:

Dans ce chapitre, on a présenté quelques modèles des systèmes électromécaniques (statique, dynamique), le modèle de friction utilisé s'approchant le plus de la pratique, la perte en

performance due aux effets des phénomènes de frottement et de flexibilité sur la commande, et quelques approches de commande citées dans la littérature: contrôleurs proportionnels intégraux dérivatifs (PID), commande H∞, programmation des gains, modèle de référence, régulateurs auto-syntonisants et commande par mode de glissement.

Nous avons vu que la stratégie de commande par programmation automatique des gains présente un avantage principal qui consiste dans la possibilité de modifier les paramètres du correcteur très rapidement en réponse aux fluctuations du procédé. Cependant, la conception demande beaucoup de temps à cause du grand nombre de répétitions du processus de conception et de simulation.

Par contre, la commande adaptative avec modèle de référence et les régulateurs auto-syntonisants présentent certains avantages tels que la simplicité de la conception et la rapidité d'adaptation. Les inconvénients de ces méthodes se situent dans les problèmes de stabilité.

Les régulateurs auto-syntonisants offrent un choix des algorithmes d'identification lors de conception du régulateur. D'autre part, l'implantation pratique est relativement plus simple que dans le cas des autres techniques. L'inconvénient majeur de cette stratégie c'est qu'il faut commencer la commande en boucle ouverte ou manuellement, ce qui permet d'avoir une période d'estimation préliminaire. Une fois qu'un vecteur des paramètres a été établi, la boucle est fermée et le processus d'auto-réglage peut être commencé.

Nous avons vu que malgré que le réglage par mode de glissement présente plusieurs avantages tels que la robustesse et la grande précision, cette méthode a été peu appliquée à cause de quelques désavantages tels que, durant la phase de convergence, le système devient

sensible aux variations des paramètres et aux perturbations. La commande par mode de glissement présente des discontinuités sur les hyperplans de glissement qui causent des oscillations et une activité intense du système de commande. De plus, l'instabilité du signal de commande peut causer une erreur stationnaire à la sortie.

Puisque le modèle inverse d'une articulation flexible avec un modèle de friction discontinu n'existe pas. Nous avons opté pour l'utilisation des réseaux de neurones qui sont connus pour leur capacité d'approximation des systèmes non-linéaires complexes. Dans le troisième chapitre, nous présenterons les types et structures des réseaux de neurones artificiels (RNA), ainsi que différents modes d'apprentissage qui nous aiderons par la suite à la conception de notre loi de commande.

Chapitre 3

RÉSEAUX DE NEURONES ARTIFICIELS

3.1- Introduction:

Les études des réseaux de neurones artificiels (RNA) datent depuis les années 1940. Grâce aux développements des recherches sur le cerveau et la disponibilité des outils de simulation, les chercheurs étudièrent des ensembles de neurones formels interconnectés. Ces réseaux, déjà développés à l'époque, permettaient d'effectuer quelques opérations logiques simples. Jusqu'aux années 1980, la recherche était freinée par la limitation théorique du perceptron. Peu après cette époque, Hopfield lança de nouveau en 1982 la recherche dans ce domaine après avoir montré l'analogie entre les RNA et les systèmes physiques.

Après les années 1990, quelques travaux scientifiques ont vu le jour dans le domaine de la robotique, parmi ces applications, on trouve la commande des systèmes d'entraînement et des systèmes de positionnement de haute performance.

Les réseaux de neurones sont des ensembles d'éléments de base appelés neurones. La philosophie derrière ces réseaux de neurones est d'imiter le cerveau humain, mais l'écart entre les réseaux de neurones et le cerveau est toujours grand, dû à la complexité de ce dernier. Cette complexité et une connaissance toujours améliorée du cerveau ont amené une multitude de solutions pour la conception des réseaux de neurones. L'absence de normalisation ajoute aussi à la difficulté de présenter clairement une théorie sur les réseaux de neurones. Nous allons présenter certains concepts qui sont normalement respectés. Les réseaux de neurones se modifient pour tenir compte de leur environnement. Cette capacité d'apprendre à partir d'exemples est d'un grand intérêt et elle s'apparente à celle du cerveau.

Les approches de commande utilisant les réseaux de neurones artificiels tentent d'imiter la structure connexionniste du système nerveux. Leur caractéristique fondamentale réside dans le fait que les fonctions de mémoire et de traitement y sont intimement liées, cela est à l'instar du cerveau qui autorise un certain flou et des imprécisions qui n'affectent pas la fiabilité de l'ensemble d'où une sûreté de fonctionnement ainsi qu'une grande capacité d'adaptation, d'apprentissage et de tolérance aux pannes. Il n'est donc pas surprenant que les recherches dans le domaine de la commande utilisent les réseaux de neurones artificiels avec des résultats satisfaisants. Cependant, ces tentatives ont été fortement critiquées. Une critique ayant fortement influencé les recherches dans ce domaine soutient que les réseaux sont incapables par leur nature de représenter les structures essentielles à la cognition. Ils auraient le double

effet de synthétiser les difficultés auxquelles les réseaux font face et de tirer des enseignements sur les différentes tendances actuelles.

Les RNA sont une formulation mathématique simplifiée des neurones biologiques. Ils ont la capacité de mémorisation, de généralisation et surtout d'apprentissage qui est le phénomène le plus important.

- Avantages:

Les principales qualités des réseaux de neurones sont leur capacité d'adaptabilité et d'auto-organisation et la possibilité de résoudre des problèmes non-linéaires avec une bonne approximation [17][18]. Ils ont une bonne immunité aux bruits et se prêtent bien à une implantation parallèle. La rapidité d'exécution est une qualité importante et elle justifie souvent à elle seule le choix d'implanter un réseau de neurones. Ces qualités ont permis de réaliser avec succès, plusieurs applications : classification, filtrage, compression de données, contrôleur, etc...

- Inconvénients:

La difficulté d'interpréter le comportement d'un réseau de neurones est un inconvénient pour la mise au point d'une application. Il est souvent impossible d'utiliser les résultats obtenus pour améliorer ce comportement. Il est également hasardeux de généraliser à partir d'expériences antérieures et de conclure ou de créer des règles sur le fonctionnement et le comportement des réseaux de neurones.

Plusieurs paramètres doivent être ajustés et aucune méthode ne permet de choisir des valeurs optimales. Beaucoup d'heuristiques sont utilisées, mais elles se contredisent parfois et elles ne permettent pas toujours de trouver des valeurs optimales.

3.2- Types et structures des réseaux de neurones artificiels (RNA):

3.2.1- Types de RNA:

Pour concevoir un réseau de neurones, nous devons établir des connexions entre les neurones. Nous avons quatre types principaux de connexion: directe, récurrente, latérale et à délais.
Tous les réseaux de neurones utilisent la connexion directe pour acheminer l'information de l'entrée vers la sortie. La connexion récurrente permet d'acheminer l'information de la sortie des neurones des couches supérieures vers les entrées des neurones précédents. Les réseaux de neurones qui doivent choisir un neurone gagnant utilisent la connexion latérale pour établir une relation entre les neurones de sortie et la maintenir. Finalement, les problèmes temporels sont résolus par les modèles de réseaux dynamiques avec des connexions à délais [19]. Les connexions entre les neurones peuvent être complètes ou partiellement complètes. Une connexion est complète lorsque les neurones d'une couche inférieure sont reliés à ceux de la supérieure et elle est locale lorsque les deux couches de neurones ne sont pas complètement reliées.

Une couche est définie comme un ensemble de neurones situé au niveau d'un réseau de neurones. Nous avons, par exemple, une couche de neurones de sortie avec des couches situées entre les entrées-sorties appelées couches cachées. Les réseaux de neurones possèdent une ou plusieurs couches de neurones et leur dimension dépend du nombre de couches et du nombre de neurones par couche.

- Perceptron:

Le perceptron est la forme la plus simple d'un réseau de neurones, il modélise la perception visuelle. Il comprend trois principaux éléments: la rétine, les cellules d'association et les cellules de décision. La fonction d'activation utilisée dans ce réseau est de type tout ou rien (0 ou 1). L'apprentissage du perceptron peut se faire avec plusieurs méthodes déjà utilisées, il n'y a qu'une seule couche de poids modifiables entre les cellules d'association et les cellules de décision. Le perceptron est limité dans ses applications. Premièrement, il ne peut être applicable que dans la classification dont les variables sont linéairement séparables et deuxièmement la sortie ne peut être que 0 ou 1.

- Perceptron multicouche:

Cette classe est la plus importante des réseaux de neurones car elle représente la généralisation du perceptron monocouche avec une fonction d'activation de type sigmoïde et une ou plusieurs couches cachées. Le vecteur d'entrée se propage dans le réseau de couche en couche jusqu'à la sortie, l'entraînement de celui-ci se fait avec l'algorithme par la rétro-propagation de l'erreur [19][20]. Ce réseau est caractérisé par son modèle du neurone traitant les non-linéarités. Il peut comporter une ou plusieurs couches cachées et un plus grand nombre de connexions permettant de résoudre la majorité des problèmes.

Même avec les avantages des couches cachées et la performance de l'algorithme d'apprentissage, il reste plusieurs problèmes non réglés comme le choix du nombre de couches, le nombre de neurones par couche et le problème des minimums locaux où le réseau peut converger.

- Réseau de neurones linéaire:

Cette classe de réseaux diffère du perceptron car elle possède un neurone dont la fonction d'activation est linéaire. L'une des règles d'apprentissage permet d'effectuer une descente de gradient de l'erreur sur une mesure d'erreur quadratique [18][19][21]. Les domaines d'application comprennent la commande, le contrôle, et le traitement du signal. L'avantage de ce réseau est qu'il converge sur un seul minimum si la solution existe, sinon l'ajout de couches n'a aucun effet. Parmi ses inconvénients, il est limité à une couche de sortie et ne peut résoudre que les problèmes dont la relation entrées/sorties est linéaire.

- Réseau RBF:

Les réseaux RBF ("*Radial Basis Function*") sont des réseaux à couches qui ont comme origine une technique d'interpolation nommée la méthode d'interpolation RBF. Ce réseau comporte une seule couche cachée dont la fonction d'activation est appelée fonction-noyau ou gaussienne et une couche de sortie avec une fonction d'activation linéaire. La méthode RBF est particulière par ses réponses utiles pour un domaine de valeurs restreint. La réponse de la fonction-noyau est maximale au noyau et décroît généralement de façon monotone avec la distance qui existe entre le vecteur d'entrée et le centre de la fonction-noyau. Afin d'approximer un comportement donné, les fonctions-noyau sont assemblées pour couvrir par leurs champs récepteurs l'ensemble des données d'entrée. Ces fonctions sont ensuite pondérées et la somme de leurs valeurs est calculée pour produire la valeur de sortie. Les réseaux RBF sont capables de calculs puissants. L'apprentissage est plus rapide et plus simple mais demande beaucoup de neurones par rapport aux réseaux multicouches. De plus, ils s'avèrent davantage insensibles à la destruction de leurs poids. Leur domaine d'application est vaste: le traitement d'image, la reconnaissance de forme, et surtout dans les problèmes de classification.

- Réseau Hopfield:

Ce réseau est basé sur les principes de la physique statistique et il est fondamentalement une mémoire adressable par son contenu. Les neurones, basés sur le modèle étudié en [22], sont tous interconnectés. Plusieurs domaines d'application sont possibles, en particulier: les mémoires associatives et l'économie. La principale limitation est qu'il n'y a pas de couches cachées.

3.2.2- Structures de RNA:

L'architecture multi-réseaux consiste à utiliser un réseau de neurones par objet à classer et un réseau de neurones pour superviser les décisions prises par le premier niveau de réseaux de neurones. Cette architecture a été peu utilisée et très peu d'articles ont été publiés.

En 1992, Mavrovouniotis et Chang [23] ont comparé plusieurs architectures de multi-réseaux avec un réseau de neurones conventionnel. Les réseaux de neurones utilisés sont à rétropropagation des erreurs avec une plage d'entrée de 0 à 1. Chaque réseau de neurones est entraîné trois fois pour vérifier la répétition des résultats obtenus et pour éviter le surentraînement. La somme totale des erreurs au carré de l'ensemble de test est utilisée pour évaluer l'entraînement.

Plusieurs organisations hiérarchiques des multi-réseaux sont comparées. Elles sont composées d'un sous-réseau d'entrée par classe à reconnaître, de quelques réseaux de neurones intermédiaires pour regrouper les classes initiales afin de former des super-classes et d'un réseau de supervision pour prendre la décision finale à partir des informations obtenues des réseaux de neurones intermédiaires.

Les multi-réseaux dont les classes sont définies avec le plus de précision obtiennent les taux d'erreur et les temps d'apprentissage les plus faibles. Les multi-réseaux ont également la particularité de permettre d'établir des relations entre l'ajustement des poids internes et une partie quelconque du problème résolu par ce réseau de neurones. Cette relation est impossible à établir avec un réseau de neurones conventionnel ou avec un multi-réseaux mal structuré.
Les multi-réseaux demandent une connaissance plus grande du problème à résoudre et un effort plus grand pour élaborer une stratégie d'entraînement. Comme le problème est simplifié par l'organisation hiérarchique du multi-réseaux, il lui est plus facile de trouver une solution optimale.

En 1991, Wen et Ozdamar [24] ont comparé un multi-réseaux conventionnel, dont le réseau de supervision est remplacé par un système expert et un réseau classique. Ils ont obtenu des résultats similaires à ceux de Mavrovouniotis et Chang [23]. Leurs résultats sont plus performants lorsque les caractéristiques sont plus pertinentes. En 1991, Ozdamar, Yaylali, Tayakar et Lopez [19] ont repris le multi-réseaux utilisé par Wen dans [24] pour évaluer la détection de transitoires épileptiques. Les résultats obtenus ont été comparés avec ceux de quatre experts. Même si ces résultats sont insuffisants pour être utilisés dans la pratique médicale, ils ont permis de démontrer qu'il est possible de détecter les transitoires épileptiques et que le réseau de supervision élimine certaines mauvaises décisions.

En 1995, Anand, Mehrotra, Mohan et Ranka [17] ont comparé une architecture classique avec une approche modulaire. L'approche modulaire consiste à utiliser un réseau de neurones par classe à reconnaître, mais il n'y a pas de réseau de supervision pour corriger les décisions prises par les premiers réseaux de neurones. Ils ont obtenu, avec l'approche modulaire, des temps d'apprentissage plus courts et de meilleures performances. La convergence a même été

obtenue pour des problèmes où il est impossible de l'obtenir avec une architecture classique. La fiabilité d'un réseau à rétropropagation des erreurs est plus faible quand le problème à résoudre comporte plusieurs classes. Lorsque le nombre de classes augmente, l'approche modulaire devient une solution avantageuse.

Les résultats obtenus avec ces rares expériences nous confirment l'intérêt de comparer et d'utiliser l'architecture multi-réseaux pour résoudre notre problème. Puisque notre stratégie d'entraînement ou de commande n'est traitée dans aucun de ces articles, nous devrons élaborer une stratégie de commande avec de nouvelles structures et algorithmes d'apprentissage.

- Structure d'un neurone:

Le neurone est la cellule fondamentale d'un réseau de neurones artificiels. Par analogie avec le neurone biologique, le neurone doit être apte à accomplir les tâches suivantes: collecter, traiter les données qui viennent des neurones émetteurs et transmettre les messages aux autres neurones. La relation entre l'entrée et la sortie du neurone peut être donnée par l'équation suivante:

$$S_i = F(a) \qquad (3.1)$$

$$a = \sum_{j=0}^{N} \mathbf{W}(i,j) \cdot \mathbf{x}(j) \qquad (3.2)$$

Les variables N, S_i, F, $\boldsymbol{x}$ et $\boldsymbol{W}$ désignent respectivement le nombre d'entrées du réseau de neurones, le vecteur de sortie du réseau, la fonction d'activation, le vecteur des entrées du réseau de neurones et la matrice des poids. Nous présentons dans la figure 3.1 la structure d'un neurone simple.

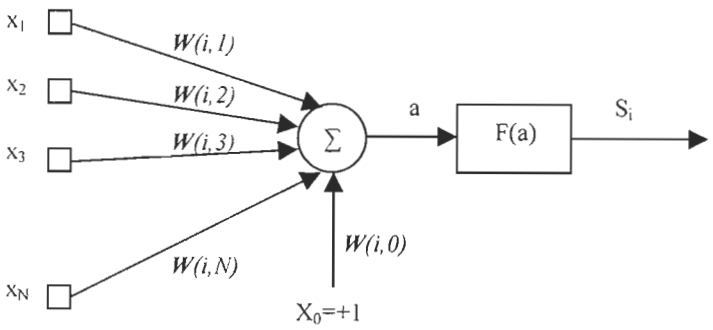

Fig. 3.1 : Structure d'un neurone simple.

- Structure d'un réseau de neurones à couches simples:

Le réseau à couches est un réseau dont les neurones sont organisés en couches, la forme la plus simple est le réseau à une seule couche. Tous les signaux d'entrées sont propagés des nœuds d'entrée vers la couche de neurones de sortie.

Le nombre de neurones d'entrée (nœuds) et de sortie est en général lié au problème à résoudre. Les entrées seront propagées à travers la matrice des poids W pour ensuite obtenir la réponse de sortie (figure 3.2). L'équation équivalente peut s'écrire sous la forme:

$$\mathbf{y}(j) = \sum_{i=0}^{N} \mathbf{W}(i,j) \otimes \mathbf{x}(i) \qquad (3.3)$$

où:

$x(i)$: Vecteur d'entrée.

$y(j)$: Vecteur de sortie.

$W(i,j)$: Poids du réseau de neurones.

$W(0,j)$ représente le biais $b(j)$.

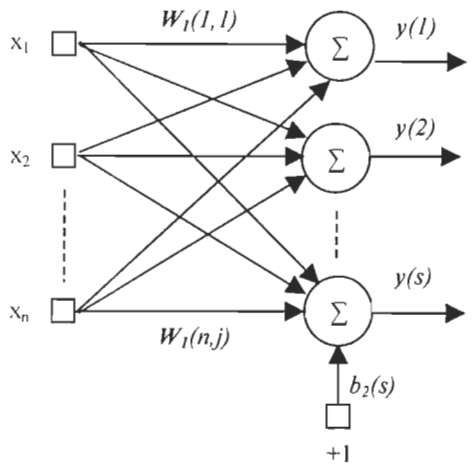

Fig. 3.2 : Réseau de neurones mono-couche.

- Structure des réseaux de neurones multicouches:

Cette structure est caractérisée par une ou plusieurs couches. À chaque couche correspond une matrice de poids W et un vecteur de seuils b, et on peut avoir aussi des fonctions d'activation différentes pour chaque couche. La fonction des couches cachées intervient entre les entrées x et la couche de sortie y. Elle permet de résoudre des problèmes plus complexes que le réseau à couche simple.

Les réseaux de neurones utilisant un apprentissage par rétropropagation [20][25] sont constitués de plusieurs couches (multicouches) qui comportent chacune plusieurs neurones. Ces derniers sont reliés entre eux par des matrices de poids qui caractérisent le réseau. Dans ce type de réseau de neurones, chaque neurone a d'abord le rôle de sommer les différentes entrées pondérées par leurs poids respectifs, puis de passer la somme obtenue dans une fonction $F(\cdot)$ linéaire ou non-linéaire et finalement, de transmettre le résultat aux neurones de la couche suivante.

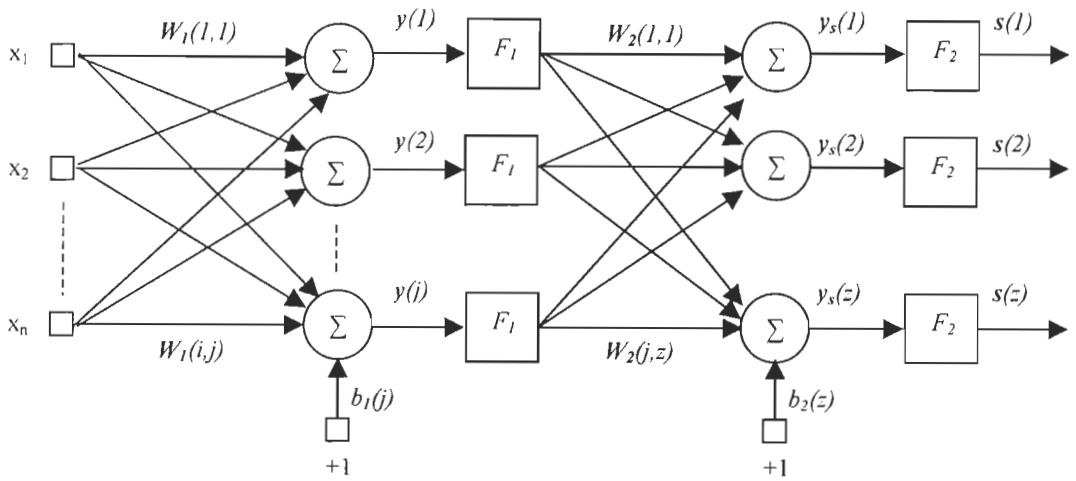

Fig. 3.3 : Réseau de neurones multicouches.

La figure 3.3 illustre un réseau de neurones multicouches avec une couche d'entrée et une de sortie. Les équations d'un tel réseau peuvent être données par la relation suivante:

$$\mathbf{s}(z) = F_2(\mathbf{y}_s(z)) \tag{3.4}$$

$$\mathbf{y}_s(z) = \sum_{i=0}^{j} \mathbf{W}_2(i,z) \cdot F_1(\mathbf{y}(i)) \tag{3.5}$$

$$\mathbf{y}(s) = \sum_{i=0}^{n} \mathbf{W}_1(i,s) \cdot \mathbf{x}(i) \tag{3.6}$$

où $\boldsymbol{s}$ est le vecteur de sortie du réseau de neurones de dimension (z x 1), $\boldsymbol{x}$ est le vecteur d'entrée de dimension (n x 1), $y(1)$ est la sortie de la 1ière couche de dimension (j x 1) et W_1 est la matrice de poids de dimension (n x j) associée à la 1ière couche. Enfin, F_1 et F_2 sont respectivement les fonctions d'activation appliquées à la 1ière et la 2ème couche.

- Structure des réseaux récurrents:

Les réseaux récurrents se distinguent des autres réseaux par la connexion des sorties de neurones avec leurs entrées. La sortie d'un neurone peut être connectée avec l'entrée du même neurone ou avec celles des autres neurones. L'importance de ces réseaux est qu'ils permettent d'apprendre la dynamique de systèmes, c'est-à-dire qu'ils peuvent imiter le comportement temporel en insérant des délais dans les boucles, reliant l'entrée à la sortie du réseau ou dans des couches internes. La figure 3.4 illustre un réseau récurrent dont les sorties sont bouclées avec les entrées de tous les neurones.

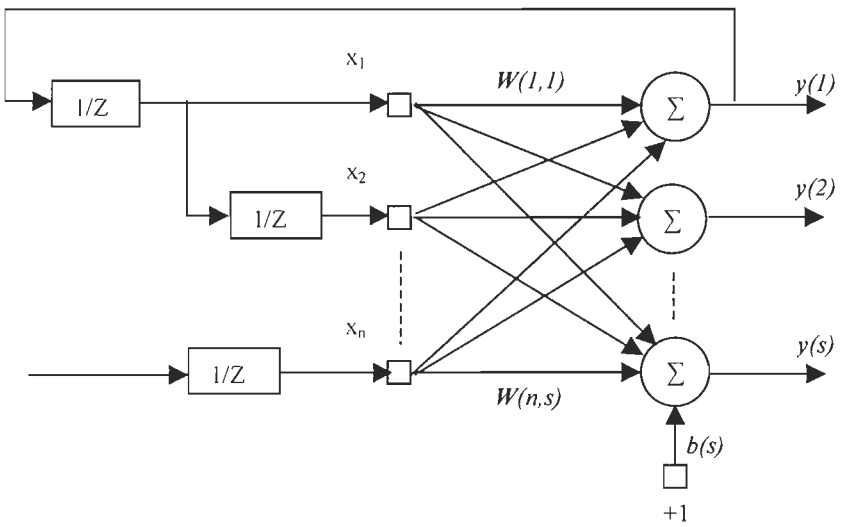

Fig. 3.4 : Réseau de neurones récurrent.

Les réseaux de neurones récurrents ont été introduits entre autres par Pineda [20] et ils s'appliquaient alors à des systèmes continus. L'équation du réseau de neurones récurrent de Pineda est donnée par:

$$\frac{d\mathbf{y}_j}{dt} = -\alpha \mathbf{y}_j + \gamma \left[\sum_{i=1}^{n} \mathbf{W}(i,j) \cdot \mathbf{y}(i) + b(j) \right] \qquad (3.7)$$

où y_j est la sortie du réseau, W est la matrice des poids, $b(j)$ est une constante de polarisation et α est une constante positive. Le but de ce type de réseau était à ce moment là, d'imposer par une procédure d'apprentissage, des points d'équilibre stables dans (3.7). Ce type de réseau fait partie de ce qu'il est commun d'appeler mémoire associative. Dans Narendra [21][22] on utilise la forme discrète et généralisée du réseau de neurones récurrent dans le cadre de l'estimation des systèmes dynamiques. L'équation du réseau de neurones récurrent généralisé peut prendre plusieurs formes selon le contexte d'utilisation. Dans [21][22] on explique en détails quatre configurations possibles de réseaux de neurones récurrents généralisés.

- Structure 'Lattice' :

C'est un réseau dont les neurones sont organisés en lignes et en colonnes, où toutes les entrées de neurones sont reliées aux nœuds d'entrées. Ces réseaux peuvent être d'une, de deux ou de plusieurs dimensions. La figure 3.5 montre un réseau de lattice à deux dimensions.

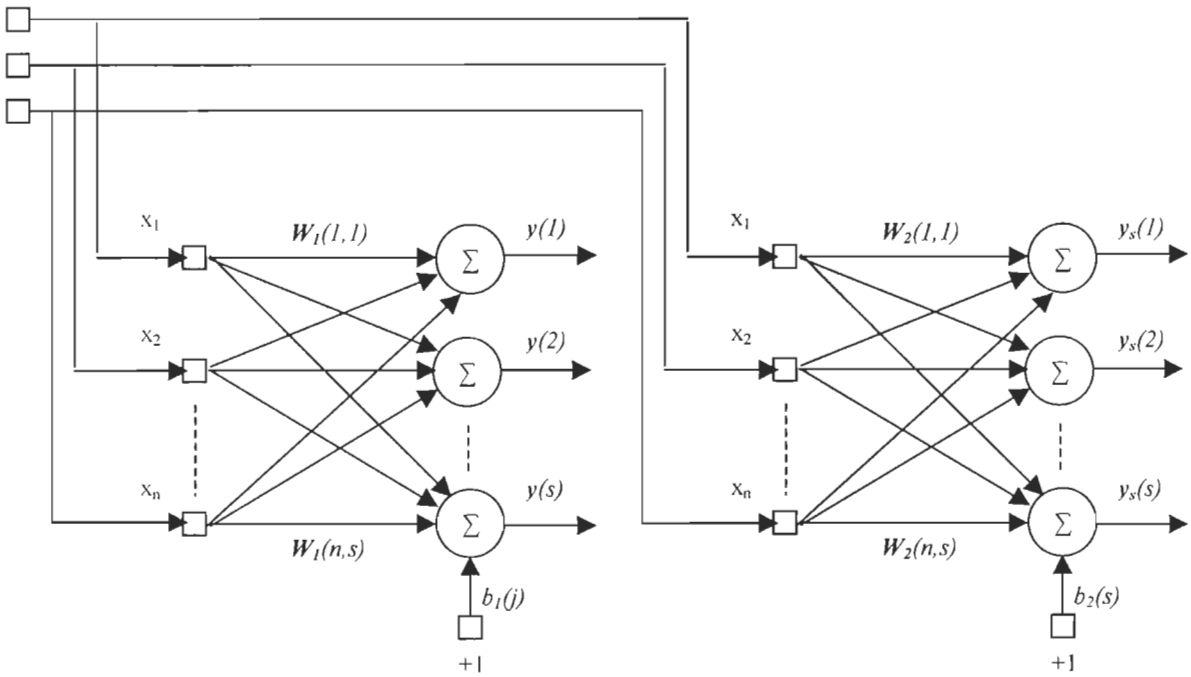

Fig. 3.5 : Structure du réseau de neurones 'Lattice'.

3.2.3- Fonctions d'activation de RNA:

Les fonctions d'activation utilisées dans les modèles connexionnistes d'aujourd'hui sont variées. On peut identifier trois principaux types de fonctions les plus connues: binaire à seuil, rampe avec saturation, et la sigmoïde.

- Fonction binaire à seuil:

La figure 3.6 présente la fonction d'activation qui a été utilisée dans [16]. Le seuil introduit une non-linéarité dans le comportement du neurone, c'est le modèle tout ou rien.

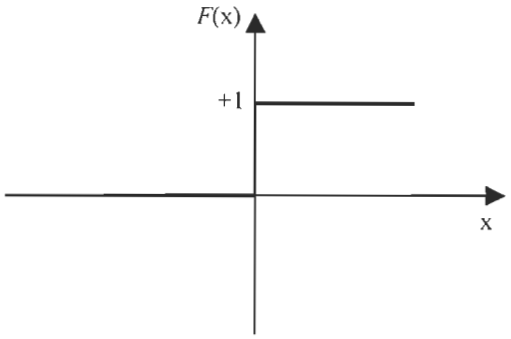

Fig. 3.6 : Fonction d'activation binaire à seuil.

- Fonction à rampe avec saturation:

Cette fonction représente un compromis entre la fonction linéaire et la fonction seuil: entre ses deux bornes, elle confère au neurone une combinaison linéaire de l'entrée. À la limite, la fonction linéaire est équivalente à la fonction seuil (Figure 3.7).

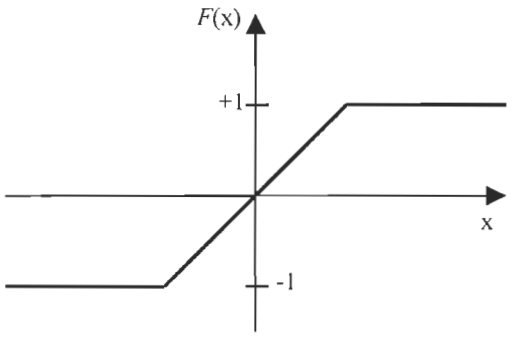

Fig. 3.7 : Fonction d'activation à rampe avec saturation.

- Fonction sigmoïde:

La fonction sigmoïde est une fonction continue qui maintient la sortie dans l'intervalle [0,1] (Figure 3.8). Son avantage principal est l'existence de sa dérivée en tout point. Elle est employée en général dans le perceptron multicouches [19].

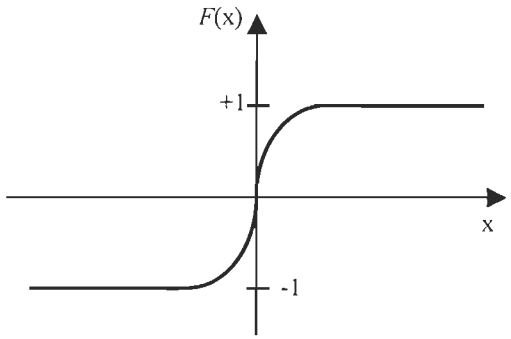

Fig. 3.8 : Fonction d'activation sigmoïde.

3.3- Modes d'apprentissage des réseaux de neurones artificiels (RNA):

On peut définir l'apprentissage comme la capacité d'emmagasiner des informations qui peuvent être rappelées par la suite. Les connaissances d'un réseau connexionniste sont mémorisées dans les poids de connexions qui seront déterminés lors de l'apprentissage. Le but de l'apprentissage pour un réseau est de trouver un ensemble de poids synoptiques qui minimisent l'erreur entre la sortie du réseau et le résultat désiré. C'est la caractéristique principale des réseaux de neurones.

Les possibilités de reconnaissance des données dans le temps sont statiques ou temporelles. Un réseau de neurones est statique si l'ordre des données dans le temps ne change pas l'interprétation faite par celui-ci et il est temporel si cela change l'interprétation. Les

possibilités de reconnaissance de données temporelles ajoutent la notion de mémoire aux réseaux de neurones. Cette mémoire est déterministe ou stochastique.

L'entraînement consiste à présenter, de façon répétitive, un ensemble de données pour permettre au réseau de neurones d'apprendre à résoudre ce problème en ajustant correctement ses poids. L'exécution est l'utilisation d'un réseau, avec tous ses paramètres et ses poids prédéterminés, pour la résolution d'un problème. Les caractéristiques qui vont constituer l'ensemble de données doivent être consistantes pour définir clairement les frontières des classes. Ces caractéristiques ont avantage à être limitées à celles qui sont vraiment nécessaires à la résolution du problème.

Les méthodes d'entraînement sont divisées en trois catégories: avec supervision, sans supervision et avec des poids prédéterminés. L'entraînement supervisé consiste à ajuster par interpolation, approximation ou optimisation les poids du réseau de neurones. Les réseaux de neurones supervisés dont l'ajustement des poids est obtenu par interpolation sont appelés à base de décision *("decision based neural network").* Avec l'entraînement non supervisé, le réseau de neurones s'ajuste uniquement en fonction de l'ensemble d'entrées. Il n'est pas nécessaire de connaître *a priori* les résultats et de constituer un ensemble de sorties désirées. Les réseaux de neurones à poids prédéterminés n'ont pas de phase d'apprentissage et la valeur des poids doit être déterminée par l'utilisateur. Une fonction d'énergie minimum est souvent utilisée pour calculer ces poids. Nous présentons par la suite différents modes d'apprentissage.

Pour l'apprentissage supervisé, on présente au réseau des entrées et on lui demande de modifier sa pondération de telle sorte que l'on retrouve la sortie correspondante. L'algorithme

consiste, dans un premier temps à propager vers l'avant les entrées jusqu'à obtenir une sortie calculée par le réseau. La seconde étape compare la sortie calculée à la sortie réelle connue. On modifie alors les poids de telle sorte qu'à la prochaine itération, l'erreur commise entre la sortie calculée et connue soit minimisée. Malgré tout, il ne faut pas oublier que l'on a des couches cachées. On rétropropage alors l'erreur commise vers l'arrière jusqu'à la couche d'entrée tout en modifiant la pondération. On répète ce processus sur tous les exemples jusqu'à ce que l'on obtienne une erreur de sortie considérée comme négligeable. Les poids optimaux seront alors appliqués sur le test pour vérifier la justesse et la précision des calculs. Les poids initiaux sont initialisés aléatoirement, puis sont déterminés par minimisation de la fonction coût du système, définie comme suit: le coût est égal à l'erreur absolue (Y_{predit}-$Y_{initial}$). Deux méthodes sont appliquées. La première pour déterminer la meilleure valeur dans la fonction sigmoïde assurant une petite valeur de l'erreur. La seconde pour déterminer la meilleure valeur du moment ou dans la boucle de minimisation du coût. Le système s'arrête soit parce qu'il a trouvé un coût inférieur à la valeur spécifiée par l'utilisateur, soit parce qu'il a balayé tout l'espace de la fonction sigmoïde. La phase de validation a pour but de tester le modèle sur le pourcentage de population non utilisé en calculant: - les prédictions correctes - les prédictions incorrectes - les mauvaises prédictions. Pour réaliser une application avec un réseau de neurones, deux étapes doivent normalement être respectées: entraînement et exécution.

Les stratégies d'entraînement sont divisées en plusieurs types: mutuelle, individuelle, discriminatoire et indépendante. Si toutes les sorties sont utilisées pour corriger les poids, nous avons une stratégie mutuelle et si une sortie est utilisée pour corriger les poids qui lui sont associés, nous avons une stratégie individuelle. Avec la stratégie mutuelle, l'expérience nous démontre que la frontière entre les classes est plus précise mais elle nécessite un temps

d'apprentissage plus grand. La stratégie discriminatoire utilise un ensemble d'entraînement comprenant des exemples couvrant le domaine complet du problème à résoudre. La stratégie mutuelle est toujours discriminatoire. La stratégie indépendante utilise un ensemble d'entraînement comprenant des exemples couvrant seulement la classe à reconnaître par une sortie. L'avantage de cette stratégie d'entraînement est de réduire le temps d'apprentissage, alors que son inconvénient est d'obtenir des frontières moins bien définies entre les classes. Il est possible d'utiliser une stratégie d'entraînement à deux niveaux pour obtenir les avantages de chacun. L'entraînement est premièrement effectué de façon individuelle, puis les résultats sont raffinés avec la stratégie mutuelle. Ces stratégies d'entraînement peuvent être utilisées pour la classification des réseaux de neurones.

- Apprentissage par rétropropagation de l'erreur:

L'algorithme qu'on désigne souvent par rétropropagation est le plus populaire parmi les techniques d'apprentissage des réseaux multicouches. L'algorithme de rétropropagation est basé sur la généralisation de la règle de Pineda [20] en utilisant une fonction d'activation sigmoïde. Le réseau utilisé est un réseau à couches où chaque neurone est connecté à l'ensemble des neurones de la couche suivante. Le principe de cet algorithme est la propagation d'un signal provenant des nœuds d'entrées vers la sortie et ensuite on propage l'erreur commise de la sortie vers les couches internes jusqu'à l'entrée [25] afin de calculer la nouvelle matrice des poids.

Dans le réseau de neurones de la figure 3.3, les matrices de poids (W_i) sont en général ajustées pour qu'un ensemble de sorties du réseau correspondent à un ensemble de sorties désirées et ce pour un ensemble d'entrées données. Dans cette optique, l'apprentissage du réseau de neurones se fait en minimisant une erreur entre l'ensemble des sorties du réseau et

l'ensemble des sorties désirées. Le plus souvent, on définit une erreur quadratique et on la minimise à l'aide d'une méthode de gradient. Ainsi, considérant le réseau de neurones représenté de (3.4) à (3.6) et une erreur quadratique définie par:

$$J(W_1,\ldots,W_m) = \frac{1}{2}\sum_{k=1}^{K}\sum_{p=1}^{n_m} e_p^2(k) \qquad (3.8)$$

où

$$e(k) = y_d(k) - y(k, \mathbf{W_1},\ldots, \mathbf{W_m}) \qquad (3.9)$$

et où $y_d(k)$ est le vecteur de sortie désirée correspondant à l'observation k et K est le nombre total d'observations considérées. Il est possible de calculer le gradient de J par rapport à l'ensemble des poids du réseau. Pour ce faire, il suffit de concaténer, dans un vecteur, l'ensemble des dérivées partielles de J par rapport à chacun des poids du réseau de neurones.

L'équation suivante présente la façon dont l'ajustement des poids du réseau se fait.

$$\mathbf{W}_{ij}(k+1) = \mathbf{W}_{ij}(k) + \Delta\mathbf{W}_{ij}(k) \qquad (3.10)$$

$$\Delta\mathbf{W}_{ij}(k) = -\eta\frac{\partial e(k)}{\partial\mathbf{W}_{ij}(k)} \qquad (3.11)$$

Les termes $\Delta\mathbf{W}_{ij}(k)$ et η désignent respectivement la correction de la matrice des poids et le taux d'apprentissage.

- Apprentissage selon une descente de gradient:

Pour de nombreuses applications, une minimisation selon une descente de gradient avec pas fixe est utilisée pour sa simplicité et ses possibilités d'intégration parallèle. Dans ce cas, la méthode peut être appliquée en utilisant la fonction de mise à jour suivante:

$$\boldsymbol{W}(k+1) = \boldsymbol{W}(k) + \Delta_{\boldsymbol{W}}(k) \qquad (3.12)$$

où

$$\Delta \boldsymbol{W}(k) = -\eta\, g(k) \qquad (3.13)$$

et où $\boldsymbol{W}(k+1)$ est le nouveau vecteur des poids du réseau de neurones, $\boldsymbol{W}(k)$ est l'ancien vecteur, $g(k)$ est le gradient de l'erreur évaluée à $\boldsymbol{W}=\boldsymbol{W}(k)$ et η est le taux d'apprentissage qui doit être strictement positif.

- Apprentissage selon la méthode Quasi-Newton:

Cet apprentissage consiste à appliquer la méthode de minimisation Quasi-Newton, sur l'erreur décrite par (3.9). Cette minimisation, effectuée par rapport au vecteur de poids $\boldsymbol{W}$, utilise le gradient $g(k)$. Dans [18], on montre que ce type d'apprentissage utilisant un estimateur d'hessien inverse converge beaucoup plus rapidement que celui utilisant la descente de gradient avec pas fixe. Watrous [18] compare les performances d'apprentissage utilisant trois méthodes de minimisation différentes. Ces méthodes sont: la descente de gradient avec pas fixe, la descente de gradient avec pas variable, la méthode Quasi-Newton. Pour deux applications distinctes, la méthode Quasi-Newton avec estimateur ressort nettement supérieure au niveau de la vitesse de convergence et de l'erreur d'apprentissage obtenue. Malgré ces avantages, la méthode Quasi-Newton comporte plusieurs inconvénients. D'abord, elle est difficilement utilisable sur des problèmes de grande taille à cause de l'espace mémoire important qu'elle requiert. Puis, contrairement à la méthode standard, la méthode

Quasi-Newton est difficilement réalisable de façon parallèle. Finalement, cette méthode ne peut être utilisée en temps réel puisqu'elle comporte une mise à jour impliquant une erreur formée par la totalité des observations de l'ensemble d'apprentissage. Pour toutes ces raisons, on suppose que dans une application considérant un apprentissage en temps différé, utilisant une machine séquentielle et impliquant des réseaux de neurones de faible dimension, il sera normalement plus avantageux d'utiliser un apprentissage selon la méthode Quasi-Newton.

Parmi plusieurs modes d'apprentissage qui existent, on trouve une partie des algorithmes qui sont dérivés de la méthode du gradient. Ces algorithmes demandent beaucoup de calcul à chaque itération et plus de mémoire que l'algorithme du gradient, malgré la rapidité de la convergence dans quelques itérations. Il est donc conseillé d'utiliser l'algorithme du rétropropagation d'erreur pour un réseau multicouche. On trouve aussi l'algorithme *'One Step Secant'* qui demande moins de mémoire et de calcul dans une itération que l'algorithme de la rétro-propagation, mais il demande plus de mémoire et de temps de calcul que la méthode du gradient. Il peut donc être considéré comme un compromis entre l'algorithme du Quasi-Newton et la méthode du gradient.

- Mémorisation, généralisation et sur-apprentissage:

Dans certaines situations, il peut arriver que l'ensemble d'apprentissage comporte un nombre très restreint d'observations. À ce moment, il est probable que le réseau de neurones soit en mesure d'apprendre l'ensemble des sorties désirées en commettant une erreur très faible. On dit alors que le réseau de neurones a mémorisé les sorties désirées. La question à se poser est alors la suivante: la mémorisation est-elle souhaitable? En général, la mémorisation n'est pas souhaitable si le nombre d'observations de l'ensemble d'apprentissage est très restreint. En fait, si on suppose que l'ensemble d'apprentissage est un échantillon d'une population

beaucoup plus vaste, il est souhaitable que le réseau de neurones puisse rendre l'erreur (3.9) faible non seulement pour l'ensemble d'apprentissage mais également pour toutes les couches du réseau en cause. Ainsi, si le réseau de neurones (après apprentissage) a la capacité de rendre (3.9) faible pour toutes les couches du réseau, on dit qu'il généralise bien. Il apparaît clair que si l'ensemble d'apprentissage est très restreint, une erreur d'apprentissage faible n'implique pas une erreur en généralisation faible. On peut même remarquer que dans ce cas, l'erreur en généralisation peut augmenter lorsque l'erreur d'apprentissage diminue de façon trop importante. On dit alors que le réseau de neurones souffre de sur-apprentissage.

- Inverse par apprentissage local:

Dans [18], Linden propose de faire un apprentissage local pour inverser le réseau de neurones représenté de (3.4) à (3.6). Le principe proposé est tout simplement d'ajuster l'entrée du réseau de neurones pour que sa sortie corresponde à une valeur donnée.

Pour cette méthode d'inversion, la procédure d'apprentissage peut être exécutée selon une des techniques classiques d'entraînement. Cependant, le gradient g doit être recalculé non pas en fonction des poids du réseau de neurones mais en fonction de son entrée $\boldsymbol{x}$(n).

- Inverse par apprentissage global:

Pour Narendra et Parthasarathy [22], l'inversion d'un réseau de neurones se fait par l'entremise d'un autre réseau. Pour obtenir ce réseau de neurones inverse, il faut premièrement générer un ensemble d'entrées-sorties $\{\boldsymbol{x},\boldsymbol{y}\}$ obtenu par la réponse du réseau N à inverser. Puis, il suffit d'exécuter l'apprentissage du réseau de neurones inverse en considérant comme entrée l'ensemble $\boldsymbol{y}$ et comme sortie désirée l'ensemble $\boldsymbol{x}$.

- Apprentissage des réseaux de neurones récurrents:

De la même façon que pour les réseaux de neurones statiques, l'apprentissage d'un réseau de neurones récurrent se fait en ajustant ses poids pour que sa sortie corresponde à une sortie désirée par rapport à une entrée donnée. Pour ce faire, il suffit donc de minimiser une fonction d'erreur par rapport à tous les poids du réseau de neurones récurrent.

Cette fonction d'erreur peut être donnée par la relation quadratique suivante:

$$J(\boldsymbol{W}) = \frac{1}{2}\sum_{k=1}^{K}\sum_{p=1}^{n} e_p^2(k) \qquad (3.14)$$

avec

$$e(k) = y_d(k) - y(k,\boldsymbol{W}) \qquad (3.15)$$

où $y(k,\boldsymbol{W})$ est le vecteur de sortie du réseau de neurones récurrent, $y_d(k)$ est le vecteur de sortie désirée correspondant à l'observation k, K est le nombre total d'observations et w est un vecteur comprenant tous les poids du réseau de neurones récurrent. On peut utiliser une méthode de gradient comme dans le cas de l'apprentissage des réseaux de neurones statiques. Cependant, pour les réseaux de neurones récurrents, il existe deux méthodes distinctes pour le calcul de l'erreur et de son gradient par rapport à $\boldsymbol{W}$. Ces méthodes sont la parallèle et la série-parallèle qui sont expliquées dans [21] et [22] pour quatre configurations différentes de réseau de neurones récurrent.

- Méthode parallèle:

La méthode parallèle est basée sur l'application directe du calcul de l'erreur (3.15) et de son gradient par rapport à $\boldsymbol{W}$. On constate dans la figure 3.9.a que la sortie y(k) dépend de la sortie précédente y(k-1). À cause de cela, le gradient de l'erreur (3.14) est lui-même dépendant du

gradient calculé précédemment. Pour illustrer le principe du calcul du gradient, considérons la fonction scalaire suivante:

$$x(k) = \Phi(x(k-1), \theta) \qquad (3.16)$$

où $x(k)$ et θ sont des scalaires.

L'équation (3.16) est une expression récurrente, on peut appliquer le même principe à l'erreur (3.15) impliquant le réseau de neurones récurrent décrit par (3.7). On peut directement calculer le gradient de J en utilisant (3.14) appliquée au réseau ou utiliser la dérivation en chaîne.

- Méthode série-parallèle:

La méthode série-parallèle est une approximation de la méthode parallèle. Elle consiste à remplacer, pour la procédure d'apprentissage du réseau de neurones récurrent, les sorties récurrentes par les sorties désirées récurrentes. Pour bien différencier les deux méthodes, la figure 3.9 illustre le principe d'apprentissage, parallèle et série-parallèle, appliqué au réseau de neurones récurrent décrit par (3.7).

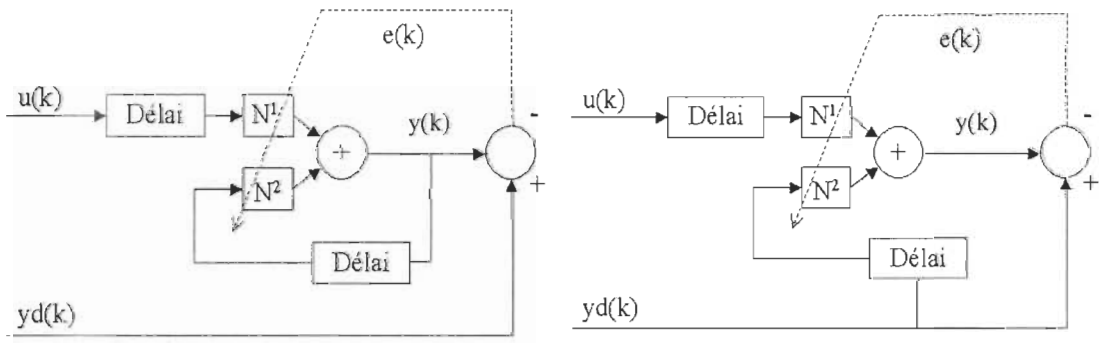

Fig. 3.9 : Apprentissage selon la méthode parallèle a) et selon la méthode série-parallèle b)

Ainsi, en considérant un apprentissage série-parallèle valable, c'est-à-dire une erreur très faible entre les sorties désirées et les sorties du réseau de neurones récurrent, on peut supposer que les méthodes parallèles et série-parallèle donneront des résultats presque équivalents. En effet si les sorties du réseau de neurones récurrent sont presque identiques aux sorties désirées, l'erreur introduite en remplaçant les sorties précédentes par les sorties désirées précédentes devrait être négligeable.

- Momentum:

La méthode de minimisation converge très lentement. Pour cette raison, il est possible d'ajouter un terme supplémentaire qui aura pour effet d'accélérer la convergence dans les régions linéaires de la fonction d'erreur (3.9). Ce terme supplémentaire appelé momentum [25] donne lieu à la fonction de mise à jour suivante:

$$W(k+1) = W(k) + \Delta W(k) \qquad (3.17)$$

où

$$\Delta W(k) = \alpha(k) + \eta\, \Delta W(k\text{-}1) \qquad (3.18)$$

et où α est une constante fixée en fonction de la rapidité de convergence souhaitée, η est le coefficient du momentum qui doit être compris dans l'intervalle [0,1].

- Mise à jour d'une observation à la fois:

L'erreur définie par (3.9) est formée de la somme des erreurs de toutes les observations de l'ensemble d'apprentissage. En pratique, la somme des erreurs sera souvent remplacée par l'erreur d'une seule observation. Ainsi, on calculera le gradient par rapport à l'erreur d'une seule observation puis la mise à jour sera exécutée et l'on passera à l'erreur de l'observation

suivante et ainsi de suite. Arrivé à la fin de l'ensemble d'apprentissage, on recommencera le processus avec la première observation. Cette méthode de mise à jour locale offre en général une convergence plus lente et plus difficile à détecter. Malgré cela, elle demeure la méthode d'apprentissage la plus utilisée pour sa simplicité et ses possibilités d'intégration parallèle. De plus, cette méthode que l'on appelle souvent apprentissage par rétro-propagation standard peut facilement s'exécuter en temps réel. En effet, parce que les observations sont considérées une par une, l'ensemble d'apprentissage peut évoluer à mesure que le temps s'écoule.

3.4- Conclusion:

Dans le présent chapitre, nous avons d'abord introduit les réseaux de neurones récurrents utilisés pour l'estimation non paramétrique des systèmes dynamiques et des systèmes dynamiques inverses. Nous avons expliqué que l'apprentissage de ces réseaux peut se faire par la méthode parallèle ou par la méthode série-parallèle. Puis, nous avons souligné que la méthode série-parallèle, quoique moins exacte que la parallèle, implique des temps d'exécution beaucoup moins importants. Nous avons expliqué les réseaux de neurones multicouches et l'apprentissage par rétro-propagation de l'erreur. Nous avons vu que l'apprentissage est en fait la minimisation de l'erreur de sortie du réseau par rapport à l'ensemble de ses pondérations. Pour l'exécution de cet apprentissage, nous avons expliqué que la méthode de minimisation par descente de gradient est la plus souvent utilisée pour sa simplicité et ses possibilités d'intégration parallèle. Nous avons également précisé que la méthode de minimisation Quasi-Newton est nettement plus performante dans le cas d'une implantation séquentielle de l'apprentissage. Par la suite, nous avons brièvement introduit les concepts de généralisation, de mémorisation et de sur-apprentissage. Finalement, deux méthodes d'inversion des réseaux de neurones ont été présentées. De ces méthodes, nous retenons que l'inversion généralisée est pratiquement inapplicable et que l'inversion par

apprentissage global semble plus intéressante que l'inversion par apprentissage local parce qu'elle est plus contraignante au niveau du nombre de solutions possibles.

On préfère concevoir des réseaux qui comportent au maximum une à deux couches cachées pour un gain de temps de traitement ainsi qu'une réduction de toute augmentation éventuelle d'erreur due aux imprécisions des machines lors de la circulation des données entre les différentes couches. De plus, théoriquement, un réseau qui possède deux couches cachées avec suffisamment de nœuds cachés peut être utilisé pour résoudre n'importe quel problème.
En ce qui concerne le choix de l'architecture, pour chacun des problèmes, quatre types sont proposés: deux avec une seule couche cachée et les deux autres avec deux couches cachées seulement, chacune avec un nombre de neurones cachés différent. Le nombre de neurones cachés (de l'ensemble des couches cachées) a toujours été déterminé tel qu'il soit supérieur au plus grand nombre de neurones constituant soit la couche d'entrée soit la couche de sortie. Finalement, les réseaux de neurones sont des programmes informatiques qui en simulant le fonctionnement des neurones du cerveau humain, permettent aux ordinateurs et aux robots d'apprendre à effectuer certaines tâches simples. Par contre, ils sont très gourmands en capacité de calcul et nécessitent des ordinateurs très puissants pour apprendre à exécuter des tâches complexes.

Malgré la diversité des algorithmes d'apprentissage il n'y a pas de méthode systématique pour la détermination d'un réseau de neurones, en particulier le choix de l'architecture du réseau, le nombre de neurones, le nombre de couches ou le choix des paramètres internes de l'algorithme comme l'erreur quadratique qu'on veut atteindre et le nombre d'itérations.

Un problème est rencontré lorsque l'apprentissage converge vers une solution sous optimale. Ce type de problème est difficile à résoudre car généralement la surface d'erreur est inconnue.

Le nombre de neurones cachés est particulièrement important parce qu'il détermine la capacité de calcul du réseau. Un nombre insuffisant de neurones cachés peut compromettre la capacité du réseau à résoudre le problème. Inversement, trop de neurones permettent au réseau d'apprendre par cœur au détriment des performances de la généralisation. Lorsque l'apprentissage d'un réseau reflète trop les particularités du problème au détriment de la tâche réelle, la matrice d'apprentissage ne reflète pas toujours adéquatement la tâche. Il en résulte que le réseau généralise mal. Même les paramètres propres à l'algorithme d'apprentissage sont difficiles à choisir comme par exemple le pas d'apprentissage.

Nous allons utiliser la structure du réseau de neurones multicouches avec six neurones dans la première couche et dix autres neurones dans la deuxième couche pour la commande du système décrit dans le deuxième chapitre. Nous utilisons au début le mode d'apprentissage hors ligne. L'inconvénient de ce mode est que le réseau n'apprend pas les différents états initiaux du système, ni le changement des paramètres de l'articulation dans le temps, d'où l'utilisation d'un apprentissage en ligne (rétro-propagation d'erreur) pour une meilleure adaptation.

Chapitre 4

MODÉLISATION ET COMMANDE ADAPTATIVE À BASE DE RÉSEAUX DE NEURONES ARTIFICIELS

4.1- Introduction:

Pour modéliser une loi de commande adaptative d'une articulation flexible en tenant compte des non-linéarités dures dues au modèle de friction utilisé, il faut choisir un modèle qui représente le système étudié et qui s'approche de la pratique sans oublier qu'il ne faut pas négliger des termes qui rendent la conception de la loi de commande inefficace.

Cette loi de commande doit être capable de s'adapter en temps réel avec le comportement de l'articulation et de satisfaire plusieurs critères, principalement: la stabilité interne et la robustesse.

4.2- Approche de commande par réseaux de neurones artificiels:

L'approche de commande proposée dans ce travail est une nouvelle structure de commande basée sur la théorie des réseaux de neurones artificiels pour des articulations flexibles. Pour commencer la conception, l'établissement des équations dynamiques du modèle inverse de l'articulation flexible est requis pour la modélisation du premier réseau de neurones qui agit comme anticipation sur le système (figure 4.1). Ce premier réseau doit représenter le plus fidèlement possible une fonction du comportement inverse de l'articulation.

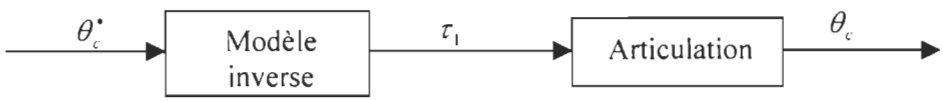

Fig. 4.1: Structure du modèle inverse.

La structure contient deux réseaux de neurones (figure 1.1), le deuxième réseau agissant en rétroaction pour corriger les erreurs laissées par le réseau d'anticipation et devant aussi assurer la stabilité interne. Pour terminer la conception et valider la fonctionnalité de cette structure de commande adaptative basée sur les réseaux de neurones, on établit les modes d'apprentissage et d'adaptation hors ligne et en ligne des deux réseaux de neurones en tenant en considération les contraintes de conception.

4.3- Modélisation inverse par réseaux de neurones artificiels:

4.3.1- Structure du modèle:

La structure du modèle inverse de l'articulation comprend un réseau de neurones artificiels qui agit comme un terme d'anticipation sur le système. Ce réseau doit représenter le comportement inverse de l'articulation. Le but du modèle est de reproduire le plus fidèlement possible l'inverse de la caractéristique de l'articulation afin d'être employé comme signal d'anticipation (figure 4.1), résultant idéalement $\theta_c \approx \theta_c^*$.

On suppose que le comportement de l'articulation est représenté par une fonction F tel que $\theta = F(\tau)$.

Une présentation simplifiée du modèle inverse est définie par une fonction G tel que $G(\theta) = G(F(\tau)) = \tau$.

4.3.2- Modèle inverse de l'articulation sous forme d'anticipation:

Nous avons déjà vu que les équations dynamiques de l'articulation peuvent s'écrire sous la forme suivante:

$$J_c\ddot{\theta}_c + F_{RC}(\dot{\theta}_c) = NK(\theta_M - N\theta_c) \tag{4.3}$$

$$J_M\ddot{\theta}_M + F_{RM}(\dot{\theta}_M) + K(\theta_M - N\theta_c) = \tau \tag{4.4}$$

D'après l'équation, on déduit la position θ_M et sa dérivée [2] [12]:

$$\theta_M = \frac{J_c\ddot{\theta}_c + F_{RC}(\dot{\theta}_c)}{NK} + N\theta_c \tag{4.5}$$

$$\dot{\theta}_M = \frac{J_c\ddot{\theta}_c + \dot{F}_{RC}(\dot{\theta}_c)}{NK} + N\dot{\theta}_c \tag{4.6}$$

Pour une modélisation inverse de l'articulation, il va falloir calculer la force de friction côté charge (figure 2.3) et sa dérivée qui interviennent dans le calcul du couple. Donc les équations peuvent s'écrire sous la forme suivante:

$$F_{RC}(\dot{\theta}_c) = F_c' sign(\dot{\theta}_c) + F_v\dot{\theta}_c + F_s' sign(\dot{\theta}_c)(e^{-(\frac{\dot{\theta}_c}{V_c})^2} - 1) \tag{4.7}$$

$$\dot{F}_{RC}(\dot{\theta}_c) = F_c'\delta(\dot{\theta}_c)\ddot{\theta}_c + F_v\ddot{\theta}_c + F_s' sign(\dot{\theta}_c)(-2\frac{\dot{\theta}_c\ddot{\theta}_c}{V_c^2}e^{-(\frac{\dot{\theta}_c}{V_c})^2}) \tag{4.8}$$

Avec F_c' le terme défini dans le modèle de Stribeck du chapitre 2.

$$F_c' = F_c + F_s' = F_c + F_s \tag{4.9}$$

D'après (4.4), (4.5), (4.6) et (4.8), le couple appliqué peut s'écrire sous la forme suivante [1]:

$$\tau = F_1\dot{\theta}_c + F_2\ddot{\theta}_c + F_3\dddot{\theta}_c + F_4\ddddot{\theta}_c + F_5 \tag{4.10}$$

où:

$$F_1 = \frac{N^2F_v + F_c'}{N} - \frac{2J_MF_s}{V_c^2}e^{-(\frac{\dot{\theta}_c}{V_c})^2} sign(\dot{\theta}_c) \tag{4.11}$$

$$F_2 = \left[F_c'\delta^2(\dot{\theta}_c)\ddot{\theta}_c + \frac{4F_s}{V_c^4}\theta_c^2 e^{-(\frac{\dot{\theta}_c}{V_c})^2}\right]J_M + \frac{1}{N}\left[\frac{F_vF_c'}{K}\delta(\dot{\theta}_c) + J_c\right] + F_v\left[F_s sign(\dot{\theta}_c)\frac{-2\dot{\theta}_c}{V_c^2}e^{-(\frac{\dot{\theta}_c}{V_c})^2} + F_v\right] \tag{3.12}$$

$$F_3 = J_M\left[F_c'\delta(\dot{\theta}_c) - \frac{2F_s}{V_c^2}sign(\dot{\theta}_c)\dot{\theta}_c e^{-(\frac{\dot{\theta}_c}{V_c})^2}\right] + \frac{F_{vm}J_c}{NK} \tag{4.13}$$

$$F_4 = \frac{J_M J_c}{NK} \tag{4.14}$$

$$F_5 = sign(\dot{\theta}_c)\left[\frac{F_c'}{N} + \frac{F_s}{N}(e^{-(\frac{\dot{\theta}_c}{V_c})^2} - 1)\right] + sign(\psi)\left[F_c' + F_s(e^{-(\frac{\psi}{V_c})^2} - 1)\right] \tag{4.15}$$

et avec:

$$\psi = \frac{J_c\dddot{\theta}_c}{NK} + N\dot{\theta}_c + \frac{1}{NK}\left[F_c'\delta(\dot{\theta}_c)\ddot{\theta}_c + F_v\ddot{\theta}_c + F_s sign(\dot{\theta}_c)\frac{-2\dot{\theta}_c\ddot{\theta}_c}{V_c^2}e^{-(\frac{\dot{\theta}_c}{V_c})^2}\right] \tag{4.16}$$

Les équations développées ne seront pas utilisées plut tard, mais elles nous ont permis de définir les termes qui interviennent dans le calcul du couple. Pour ce faire, nous avons pris comme hypothèse que le modèle du moteur est équivalent à celui de la charge. En fait, il faut appliquer une impulsion au couple pour avoir une variation instantanée de la position pour permettre une compensation des termes comme F_c.

Pour concevoir le modèle inverse de l'articulation, il faut pouvoir représenter les termes non-linéaires de l'équation du couple. Malheureusement, nous ne pouvons représenter ses non-linéarités causées par les termes de signe de vitesse $sign(\dot{\theta}_c)$ ainsi que les impulsions $\delta(\dot{\theta}_c)$, ce qui fait que le modèle inverse de l'articulation soit irréalisable, d'où l'utilisation du réseau de neurones pour apprendre le modèle inverse approximatif qui jouera aussi le rôle de l'anticipation dans notre système. On prévoit un apprentissage en ligne pour une adaptation.

4.3.2.1- Apprentissage hors ligne:

Nous avons opté pour un réseau de neurones multicouches avec deux couches cachées pour résoudre notre problème. Les neurones d'entrées correspondent aux descripteurs, les neurones de sortie correspondent aux variables à corréler, le nombre de couches intermédiaires

(couches cachées) varie en fonction de la complexité du problème (en général une ou 2 couches suffisent). Le nombre de neurones dans les couches cachées est également variable et doit être normalement égal à deux fois le nombre des entrées. Dans notre cas, vu que nous avons trois entrées ($\theta_c^*, \dot{\theta}_c^*, \ddot{\theta}_c^*$), on a choisi d'utiliser six neurones et la sigmoïde comme fonction d'activation.

Le réseau est entraîné durant la phase d'apprentissage à l'aide d'un ensemble de données qui permettent de fixer les poids de connexions. Une fois les poids calculés, le réseau est fonctionnel et il peut mener la tâche qui lui a été assignée sans modification des poids.

Pour faire un apprentissage hors ligne du premier réseau de neurones qui agit comme un terme d'anticipation, il faut tout d'abord connaître les entrées et les sorties de notre système. Dans notre cas, l'entrée est le couple appliqué à l'articulation, et les sorties sont les positions, vitesses, et les accélérations du moteur ou de la charge. Nous avons sauvegardé ces entrées et sorties qui représentent le comportement du système pour commencer la procédure de l'apprentissage.

La procédure d'apprentissage consiste à suivre les étapes de l'algorithme d'entraînement suivant:

- Chargement des données d'apprentissage.
- Normalisation des données par rapport à leurs valeurs maximales.
- Choix du nombre de neurones.
- Initialisation des poids du réseau de façon aléatoire.
- Choix des paramètres de l'algorithme d'apprentissage.
- Validation des résultats.
- Sauvegarde de la matrice des poids du réseau de neurones.

Si l'erreur quadratique est inférieure au seuil fixé, on aura l'arrêt de l'apprentissage, sinon l'apprentissage sera poursuivi avec les poids de la dernière itération. L'initialisation des poids est effectuée une seule fois au début de l'apprentissage.

4.3.2.2- Apprentissage en ligne:

L'apprentissage utilisé s'appelle apprentissage avec rétro-propagation d'erreur. Les étapes de cette procédure nécessaires pour effectuer un apprentissage en ligne sont:

- Initialisation des poids du réseau aléatoirement.
- Le choix du nombre de couches et de nœuds.
- Le choix de la fonction du taux d'apprentissage.
- Propagation.
- Calcul de l'erreur.
- Rétro-propagation de l'erreur à travers les couches du réseau de neurones.
- Calcul des nouvelles matrices de poids.

La figure 4.2 représente le schéma d'adaptation pour l'entraînement en ligne de l'anticipation.

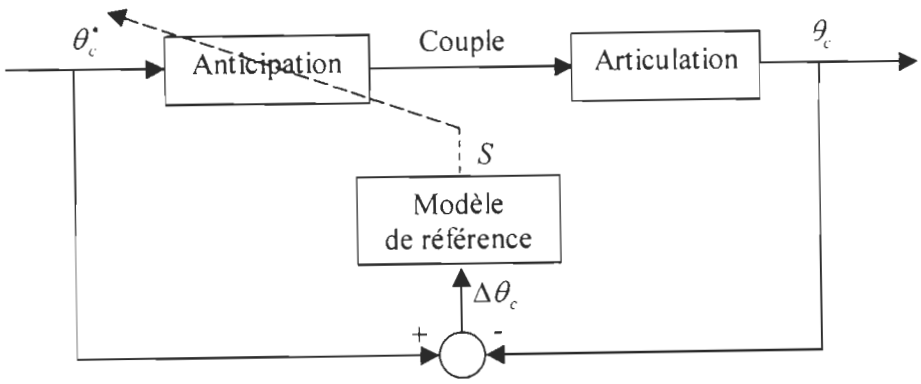

Fig. 4.2: Structure du modèle d'apprentissage en ligne de l'anticipation.

Cette structure comporte l'anticipation entraînée avec un apprentissage en ligne et adaptée avec le signal S qui est celui de la sortie du modèle de référence défini ultérieurement à la section 4.5.2. Nous utilisons un taux d'apprentissage variable décrit par:

$$\eta = \frac{\eta_{\max}}{1 + e^{-\sigma(|S|-\mu)}} \qquad (4.17)$$

$\eta_{\max}$ représente le maximum du taux d'apprentissage, σ caractérise le taux de variation du taux d'apprentissage et μ est utilisée pour fixer le minimum du taux d'apprentissage.

Nous présentons maintenant le principe de l'apprentissage par rétro-propagation de l'erreur [20]. La figure suivante présente un réseau de neurones simples à deux couches, une couche cachée et une couche de sortie:

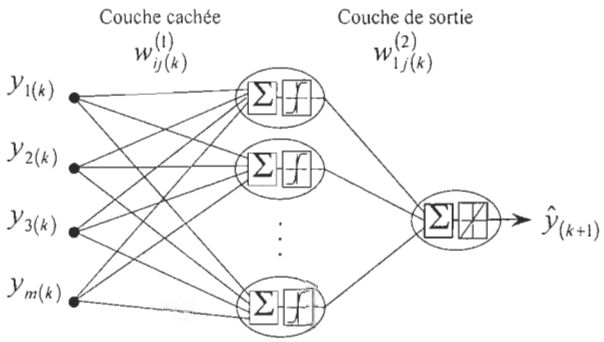

Fig. 4.3: Réseau de neurones à deux couches.

On présente la relation entre les entrées et la sortie du réseau comme suit:

$$\mathbf{v}_{\mathbf{j}}^{(\mathbf{n})}(k) = \sum_{i=0}^{m} \mathbf{W}_{\mathbf{ij}}^{(\mathbf{n})}(k)\mathbf{y}_{\mathbf{i}}^{(\mathbf{n})}(k) \qquad (4.18)$$

$$\mathbf{y}_{\mathbf{j}}^{(\mathbf{n}+1)}(k) = \varphi_{\mathbf{j}}^{(\mathbf{n})}(\mathbf{v}_{\mathbf{j}}^{(\mathbf{n})}(k)) \qquad (4.19)$$

où φ représente la fonction d'activation, m est le nombre d'entrées de la couche, n identifie la couche et $\mathbf{y}_j^{(n+1)}(k)$ est la sortie j de la couche n du réseau de neurones.

Dans notre cas, nous avons une seule sortie. Donc, l'erreur à la sortie $e(k)$ peut s'écrire sous la forme:

$$e(k) = d(k) - \hat{y}(k) \tag{4.20}$$

$$e(k) = d(k) - \varphi(\mathbf{v}(k)) \tag{4.21}$$

$$\xi(k) = \frac{e^2(k)}{2} \tag{4.22}$$

Les termes *d(k)* et $\hat{y}(k)$ désignent respectivement la sortie désirée et la sortie estimée. L'algorithme de rétro-propagation applique la correction $\Delta\mathbf{W}_{ij}^{(n)}(k)$ à chaque poids des couches du réseau de neurones $\mathbf{W}_{ij}^{(n)}(k)$. On calcule la nouvelle matrice des poids à l'aide de l'équation suivante:

$$\mathbf{W}_{ij}^{(n)}(k+1) = \mathbf{W}_{ij}^{(n)}(k) + \Delta\mathbf{W}_{ij}^{(n)}(k) \tag{4.23}$$

Avec:

$$\Delta\mathbf{W}_{ij}^{(n)}(k) = -\eta \frac{\partial\xi(k)}{\partial\mathbf{W}_{ij}^{(n)}(k)} \tag{4.24}$$

$$\frac{\partial\xi(k)}{\partial\mathbf{W}_{ij}^{(n)}(k)} = -e(k)\varphi_j'^{(n)}(\mathbf{v}_j^{(n)}(k))\mathbf{y}_i^{(n)}(k) \tag{4.25}$$

Donc, on peut écrire le terme qui correspond à la correction de chaque poids sous la forme suivante:

$$\begin{pmatrix} Correction \\ des\ poids \\ \Delta \mathbf{W}_{ij}^{(n)}(k) \end{pmatrix} = \begin{pmatrix} Taux \\ d'apprentissage \\ \eta \end{pmatrix} \cdot \begin{pmatrix} Gradient \\ local \\ \delta_j^{(n)}(k) \end{pmatrix} \cdot \begin{pmatrix} Entrée\ du \\ neurone\ j \\ \mathbf{y}_i^{(n)}(k) \end{pmatrix} \tag{4.24}$$

$$\Delta \mathbf{W}_{ij}^{(n)}(k) = \eta \delta_j^{(n)}(k) \mathbf{y}_i^{(n)}(k) \tag{4.25}$$

Avec η le taux d'apprentissage de l'algorithme de rétro-propagation et le terme $\delta_j^{(n)}(k)$ qui s'écrit sous la forme suivante pour la couche cachée:

$$\delta_j^{(n)}(k) = -\frac{\partial \xi(k)}{\partial \mathbf{v}_j^{(n)}(k)} = \varphi_j'^{(n)}(\mathbf{v}_j^{(n)}(k)) \sum_j \delta_j^{(n+1)}(k) \mathbf{W}_{ij}^{(n+1)}(k) \tag{4.26}$$

Pour la couche de sortie le terme $\delta_j(k)$ s'écrit sous la forme:

$$\delta_j(k) = -\frac{\partial \xi(k)}{\partial v(k)} = e(k) \tag{4.27}$$

Le facteur $\varphi_j'^{(n)}(\mathbf{v}_j^{(n)}(k))$ est requis pour le calcul du terme $\delta_j^{(n)}(k)$. Il présente la dérivée de la fonction d'activation associée à chaque couche cachée du réseau. Pour que cette dérivée existe, il faut que la fonction soit continue.

Dans le cas de la fonction d'activation tangente hyperbolique, sa dérivée s'écrit sous la forme suivante:

$$\varphi_j'^{(n)}(\mathbf{v}_j^{(n)}(k)) = a \mathbf{y}_j^{(n)}(k)(1 - \mathbf{y}_j^{(n)}(k)) \tag{4.28}$$

où a représente la constante du calcul de la dérivée de la fonction d'activation $0 < a < 1$. Normalement on prend $a = 1$.

4.4- Commande adaptative avec rétroaction par réseaux de neurones artificiels:

4.4.1- Problématique et stratégie de commande:

L'approche de commande consiste à concevoir et évaluer un terme de rétroaction avec un retour unique de sortie de la position et de la vitesse de la charge avec un apprentissage des paramètres incertains pour lequel il faut assurer la stabilité interne en plus de la stabilité de sortie.

Pour concevoir le système de rétroaction, on doit prendre en considération la possibilité de mesure des variables du moteur de façon à traiter plus directement le problème de stabilité interne et des variables de la charge pour assurer la stabilité de la sortie. La stratégie de commande consiste à développer un algorithme d'adaptation hors ligne pour la structure de commande et valider par simulation et comparer les résultats avec ceux du modèle inverse.

4.4.2- Structure du modèle:

La structure du modèle de la rétroaction est présentée dans le schéma de la figure 4.4:

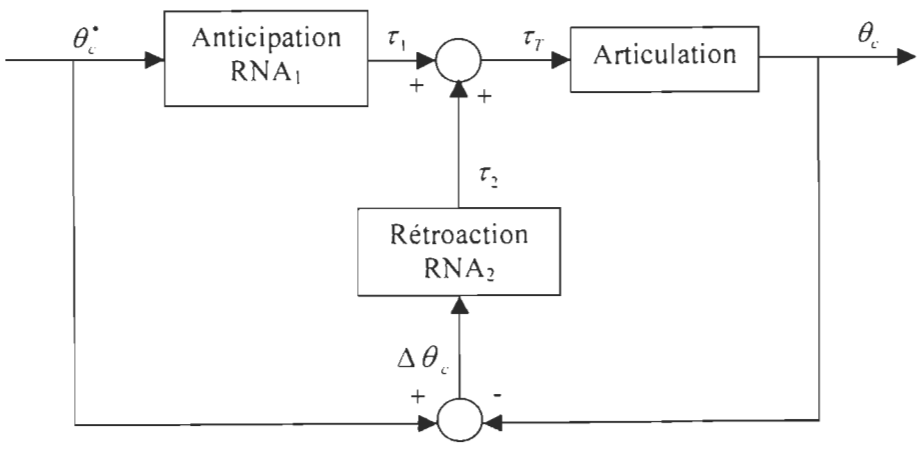

Fig. 4.4: Structure du modèle d'anticipation et rétroaction.

Cette structure comporte l'anticipation de la figure 4.1 qui représente l'approximation du modèle inverse de l'articulation, et une rétroaction qui vient pour assurer la stabilité interne et externe. Les deux réseaux sont entraînés avec un apprentissage hors ligne.

4.4.3- Apprentissage:

Nous utilisons comme anticipation le réseau de neurones RNA_1 conçu à la section 4.3.3. Pour corriger les erreurs laissées par ce dernier et assurer la stabilité interne, nous ajouterons un réseau de neurones RNA_2 pour générer un terme de rétroaction. Les données d'apprentissage sont les erreurs de position et de vitesse. On ne peut prendre en considération l'erreur de l'accélération car elle n'est généralement pas accessible dans la pratique. Ensuite, on procède au choix du nombre de neurones et on initialise les poids du réseau de façon aléatoire. Enfin, nous choisissons les paramètres de l'algorithme d'apprentissage et nous commençons l'apprentissage. Si l'erreur est inférieure au seuil fixé dans les paramètres, on forcera l'arrêt de l'apprentissage, sinon l'apprentissage se poursuivra avec les poids de la dernière itération jusqu'à ce que le seuil visé ou le nombre limite d'itérations soit atteint.

4.5- Commande adaptative avec anticipation et rétroaction par réseaux de neurones artificiels:

4.5.1- Initialisation:

Avant de commencer l'apprentissage des deux réseaux de neurones, il faut commencer par l'étape d'initialisation des paramètres qui consiste à fixer la dimension des RNA_1 et RNA_2; les poids des réseaux de façon aléatoire; remise à zéro de tous les termes de correction ainsi que la détermination des taux d'apprentissage des deux réseaux de neurones qui forment notre loi de commande.

En ce qui concerne le choix de l'architecture, le nombre de neurones cachés (de l'ensemble des couches cachées) a toujours été déterminé tel qu'il soit supérieur au plus grand nombre de neurones constituant les couches d'entrées ou de sorties. Dans notre cas, le réseau d'anticipation a pour entrées la position, la vitesse et l'accélération de la charge. Par contre, le réseau de rétroaction a pour entrées les erreurs de position et vitesse charge. Donc, on a choisi d'utiliser six neurones dans la première couche et dix autres dans la deuxième couche et la sigmoïde comme fonction d'activation vu les résultats les plus satisfaisants qu'on peut avoir.

Le choix des taux d'apprentissage η_1 et η_2 des deux RNA est fait pour que l'anticipation fournisse tout le signal de commande désiré. Dans ce cas, on peut dire que le réseau de l'anticipation a bien appris le modèle inverse de l'articulation, et les phénomènes de friction et de flexion ont été compensés par ce dernier.

4.5.2- Apprentissage en ligne:

Le réseau est en phase d'apprentissage continuel. À chaque nouvelle action le réseau apprend et s'ajuste en réalisant la tâche qui lui est assignée. La figure 4.5 représente la structure du modèle utilisé.

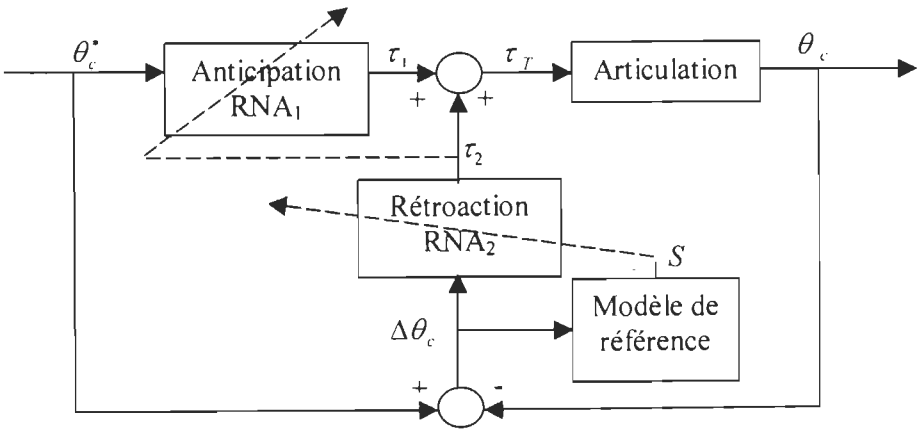

Fig. 4.5: Structure du modèle de l'apprentissage en ligne (anticipation et rétroaction).

Cette structure comporte une anticipation et une rétroaction entraînées avec un apprentissage en ligne (rétro-propagation de l'erreur), et avec un taux d'apprentissage variable. La rétroaction a pour signal d'adaptation le modèle de référence S défini ultérieurement dans cette section. Le modèle de référence est tel que S soit non nul si le comportement de la sortie du système dévie du comportement désiré, forçant alors l'ajustement des poids du réseau RNA_2. Par contre, l'anticipation est adaptée avec le signal de la sortie de la rétroaction. Les poids de RNA_1 sont donc modifiés dès que le signal de rétroaction est actif. Ainsi, avec η_2 représentant l'erreur sur l'anticipation, RNA_1 tend à apprendre le modèle inverse de l'articulation et la dynamique désirée du système (modèle de référence). Les poids initiaux des deux réseaux sont fixés de façon aléatoire.

Les taux d'apprentissage sont fixés de façon à ce que le réseau d'anticipation soit capable de fournir tout seul le signal de commande désiré, et le réseau de rétroaction vient pour ajouter un terme de correction au signal de commande à la sortie du réseau d'anticipation. Si ce dernier est capable de reproduire le signal de commande désiré, on aura une valeur nulle de la sortie du réseau de rétroaction. Si le taux d'apprentissage de l'anticipation η_1 est trop faible, la période d'apprentissage de l'anticipation sera très longue et le système opérera principalement en mode de rétroaction. Si le taux d'apprentissage de l'anticipation η_1 est trop élevé, RNA_1 agira comme une rétroaction et apprendra continuellement un nouveau modèle quand la forme de la trajectoire d'entrée changera.

Avec la complexité de notre système due au modèle de friction utilisé et à la flexion causée par le ressort de torsion, nous avons opté pour le choix d'un taux d'apprentissage variable en fonction du signal d'adaptation qui représente l'erreur.

La figure 4.6 présente l'allure du signal du taux d'apprentissage η (η_1 ou η_2) en fonction du signal d'adaptation. Dans le cas de l'anticipation ce signal représente le couple à la sortie de RNA_2, et dans le cas de la rétroaction, le signal représente la fonction S, cette structure a été présentée dans la figure 4.5.

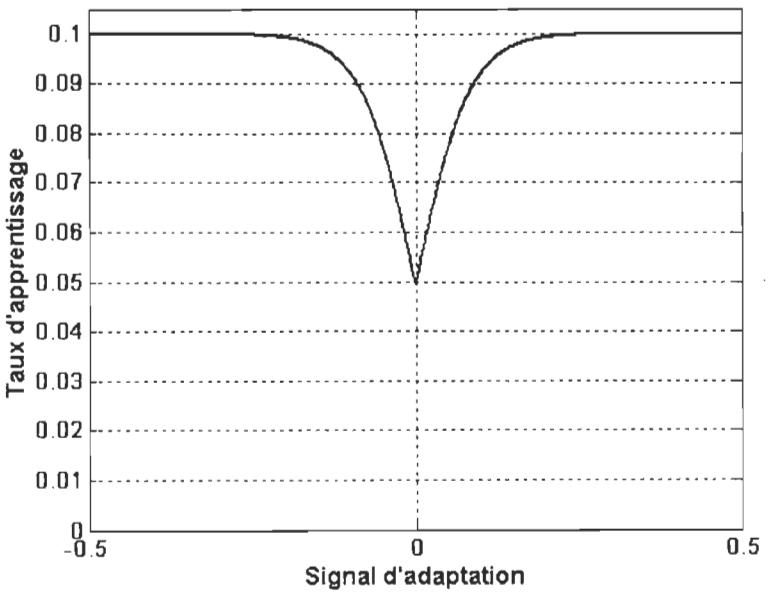

Fig. 4.6: Allure du taux d'apprentissage des réseaux de neurones.

Nous écrivons le taux d'apprentissage η sous la forme:

$$\eta = \frac{\eta_{max}}{1 + e^{-\sigma(|S|-\mu)}} \qquad (4.29)$$

Avec:

$$\eta_{max} = 1^{e-1};\ \mu = 1^{e-3};\ \sigma = 25;$$

Pour assurer la stabilité d'entrée-sortie et obtenir $\theta_c \to \theta_c^*$, nous définissons le comportement désiré de l'erreur de sortie par un modèle du premier ordre $S_0 = (\dot{\theta}_c^* - \dot{\theta}_c) + p(\theta_c^* - \theta_c)$ qui doit converger vers zéro et où $1/k$ représente la constante de temps du modèle de premier ordre. Par contre, ce modèle n'assure aucunement la stabilité interne du système. Pour cet objectif, nous pourrions définir un nouveau modèle $S_1 = (\dot{\theta}_m^* - \dot{\theta}_m) + p(\theta_m^* - \theta_m)$ dont la sortie doit converger vers zéro. Cependant, nous n'avons pas accès à la position moteur de consigne θ_m^*, ni à la vitesse moteur de consigne $\dot{\theta}_m^*$. Pour assurer la stabilité interne et d'entrée-sortie. nous écrivons la fonction S en s'inspirant de S_0 et S_1 à l'aide de S_0 dans laquelle la vitesse de la charge est remplacée par une combinaison linéaire des vitesses de la charge et du moteur:

$$S = \dot{\theta}_c^* - \gamma\dot{\theta}_c - (1-\gamma)\dot{\theta}_m + p(\theta_c^* - \theta_c) \qquad (4.30)$$

où:

p définit la rapidité de convergence souhaitée.

γ permet de faire un compromis entre la convergence de la sortie et la stabilité interne.

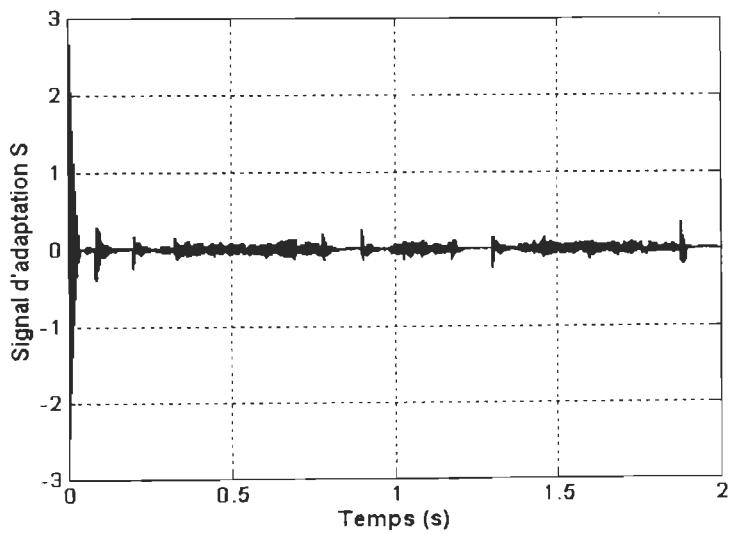

Fig. 4.7: Allure du signal d'adaptation S.

Nous avons rencontré un dilemme entre la stabilité interne et celle de la sortie lors de la simulation, mais avec le choix de l'équation de S, nous avons pu trouver un compromis. En fait, nous ne pouvons assurer la stabilité interne sans perdre dans la qualité du signal de sortie, et nous ne pouvons assurer la stabilité du signal de sortie tout en maintenant une très bonne stabilité interne. Donc nous pourrons exprimer la constante γ en pourcentage allant de 0% pour assurer une stabilité interne jusqu'à 100% pour assurer la stabilité du signal de sortie.

L'allure de la fonction d'apprentissage de la variable η_1 est inspirée par les bandes d'adoucissement employées en commande par mode de glissement. Un taux élevé est employé quand l'erreur est grande pour assurer une convergence rapide vers zéro. Le taux est réduit dans une bande près de zéro pour permettre au système de « s'accrocher » et ainsi éviter les oscillations autour du point zéro. Une réduction importante du taux d'apprentissage de RNA_1 près d'une erreur zéro amène aussi ce réseau à se corriger principalement lorsque la rétroaction est très active et à éviter qu'il essaie de compenser les faibles erreurs qui sont du ressort de la rétroaction.

Nous avons effectué des simulations avec les paramètres suivants:

$J_L = J_M$ = 5.5e-5 [N.m / (rad/s^2)]; $F_{cL} = F_{cM}$ = 0.0288 [N.m]; $F_{sL} = F_{sM}$ = 0.0191 [N.m];

$v_{sL} = v_{sM}$ = 0.95 [rad / s^2]; $F_{vL} = F_{vM}$ = 5.5e-4 [N.m / (rad/s)]; p = 130; γ = 0.85;

Le signal de consigne est présenté à la figure 4.8. Les figures 4.9 et 4.10 présentent des résultats de simulation avec un taux d'apprentissage fixe. Ces figures représentent respectivement la vitesse moteur dans le cas où le taux d'apprentissage η_1 est égal à 0,1 et 0,05. Ces valeurs designent le maximum et le minimum de la fonction d'apprentissage présentée dans la figure 4.6. Dans ces deux cas, la stabilité d'entrée-sortie a été assurée, mais

pas la stabilité interne comme l'illustrent ces figures: on remarque aussi un cycle limite dans les deux cas.

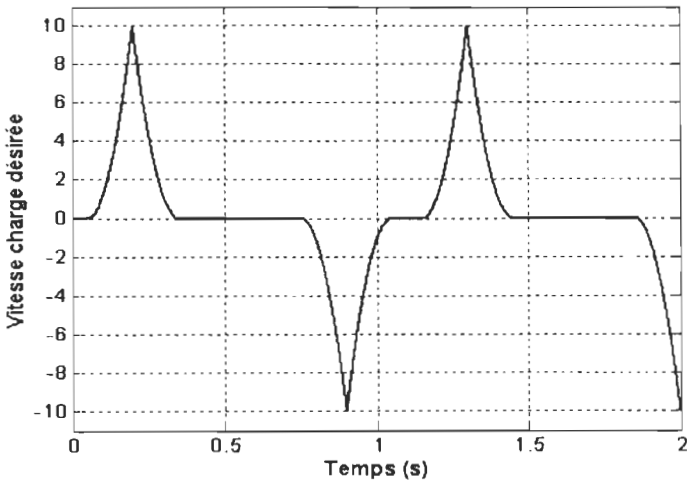

Fig. 4.8: Consigne des vitesses moteurs.

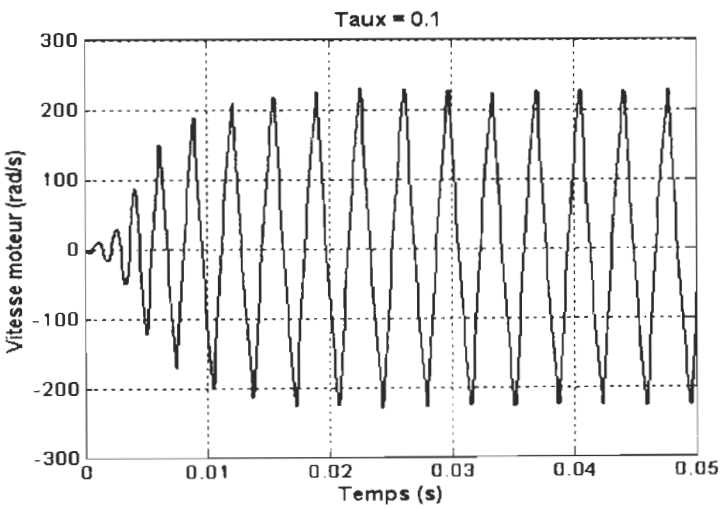

Fig. 4.9: Vitesse moteur avec un taux d'apprentissage $\eta_1 = 0,1$.

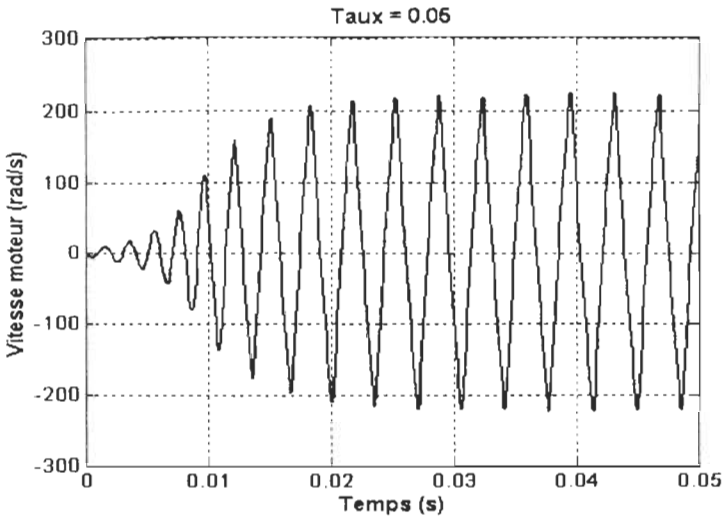

Fig. 4.10: Vitesse moteur avec un taux d'apprentissage $\eta_1 = 0{,}05$.

Nous remarquons dans les résultats de la figure 4.11 qu'avec l'utilisation du taux d'apprentissage variable, nous avons pu régler le problème de la stabilité interne.

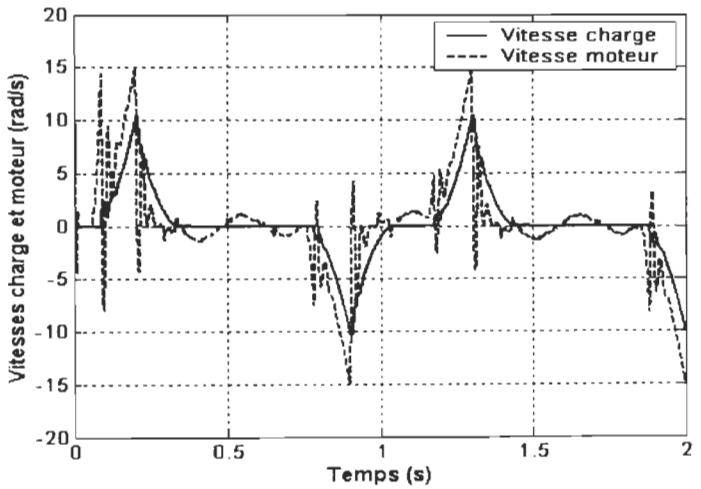

Fig. 4.11: Vitesses charge et moteur avec un taux d'apprentissage variable.

4.6- Conclusion:

Le chapitre a porté sur la modélisation et la conception d'un modèle inverse d'une articulation flexible, ainsi qu'un modèle de rétroaction, l'initialisation des paramètres du système et enfin l'établissement des modes d'apprentissage hors ligne et en ligne.

L'établissement des équations du modèle inverse de l'articulation était nécessaire pour déterminer les termes non-linéaires qu'on ne peut reproduire, d'où l'utilisation d'un réseau de neurones pour estimer ces non-linéarités. Nous avons pu constater l'utilité de la fonction du taux d'apprentissage variable lors de l'entraînement en ligne pour assurer la stabilité interne ainsi que celle d'entrée-sortie.

Nous avons préféré construire des réseaux qui comportent deux couches cachées pour un gain de temps de traitement ainsi qu'une réduction de toute augmentation éventuelle d'erreur due aux imprécisions des machines lors de la circulation des données entre les différentes couches. De plus, théoriquement, un réseau qui possède deux couches cachées avec suffisamment de nœuds cachés peut être utilisé pour résoudre n'importe quel problème.

Nous présenterons dans le prochain chapitre les principaux résultats de simulation, un tableau comparatif et une analyse de la performance de la loi de commande proposée.

Chapitre 5:

RÉSULTATS DE SIMULATION ET PERFORMANCES DES LOIS DE COMMANDE

5.1- Introduction:

Dans ce chapitre, nous présentons les résultats de simulation et comparons les quatre structures de commande. Nous démontrons une convergence de la position de sortie tout en assurant une stabilité interne, ainsi que la rapidité de convergence avec des conditions initiales non nulles.

Les simulations ont été faites avec les paramètres suivants [8]:

$J_L = J_M$ = 5.5e-5 N.m / (rad/s^2);

$v_{sL} = v_{sM}$ = 0.95 rad / s^2;

$F_{cL} = F_{cM}$ = 0.0288 N.m;

$F_{sL} = F_{sM}$ = 0.0191 N.m;

$F_{vL} = F_{vM}$ = 5.5e-4 N.m / (rad/s);

k = 0.52; p = 130; γ = 0.85; Tech =10^{-4};

5.2- Commande inverse d'une articulation flexible avec non-linéarité:

5.2.1- Apprentissage hors ligne:

La figure 5.1 représente la trajectoire utilisée lors de l'entraînement hors ligne du réseau de neurones (code en annexe B). La trajectoire d'entraînement doit être riche en information et doit contenir différents niveaux de vitesse et accélération pour assurer un bon apprentissage.

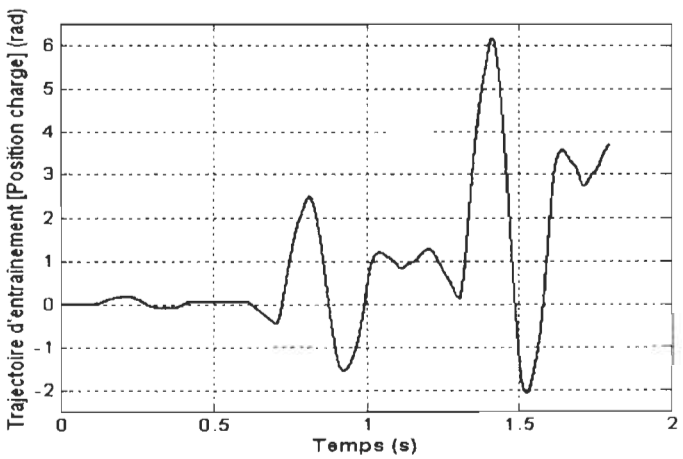

Fig. 5.1: Trajectoire d'entraînement utilisée dans l'apprentissage hors ligne de l'anticipation.

La consigne utilisée comme position désirée de la charge lors de la simulation du modèle d'anticipation avec un apprentissage hors ligne est représentée dans la figure 5.2. Le schéma

de validation est disponible dans la page 116. L'apprentissage a été fait avec les mêmes paramètres de l'articulation et en fixant l'erreur souhaitée à 10^{-1} et le nombre d'itérations à 50. Les performances de l'apprentissage sont démontrées dans la figure B.2 dans l'annexe B.

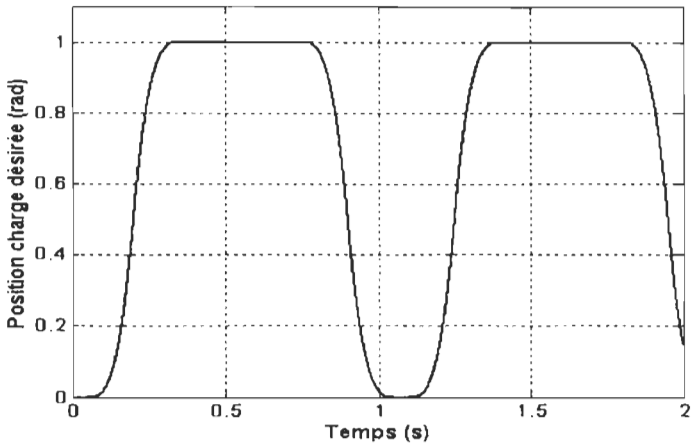

Fig. 5.2: Consigne utilisée lors des simulations.

La figure 5.3 représente l'erreur entre la position désirée de charge représentée dans la figure 5.2 et la position obtenue de la sortie avec des conditions initiales nulles.

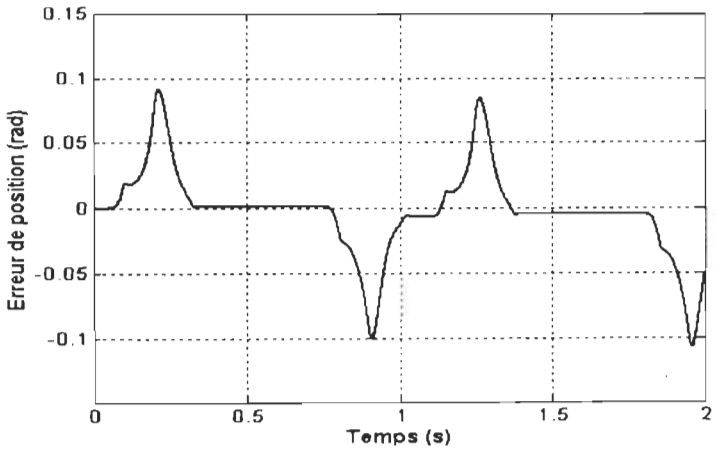

Fig. 5.3: Erreur de position charge hors ligne avec conditions initiales nulles.

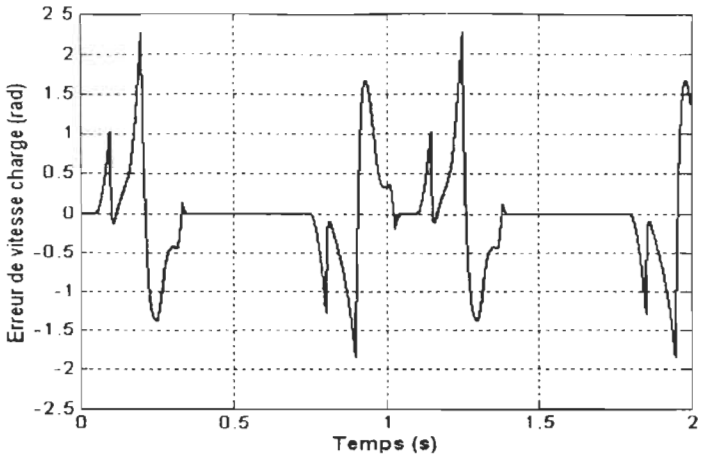

Fig. 5.4: Erreur de vitesse charge; apprentissage hors ligne avec conditions initiales nulles.

Nous remarquons que les erreurs de position et de vitesse sont très grandes avec un apprentissage hors ligne de l'anticipation. Par contre, on remarque une certaine stabilité. Dans ce cas, le rôle de la rétroaction sera de diminuer l'écart des erreurs sans perdre la stabilité.

La figure 5.5 représente la position désirée de la charge et celle obtenue avec des conditions initiales non nulles ($\theta_m = \theta_c = 0.25$; toutes les autres variables d'état sont initialement à zéro) lors d'un apprentissage hors ligne sans rétroaction (figure 4.1).

Nous remarquons qu'avec une erreur initiale non nulle, le réseau d'anticipation ne peut pas corriger cette erreur, d'où la nécessité d'introduire un terme de rétroaction ou d'utiliser un apprentissage en ligne.

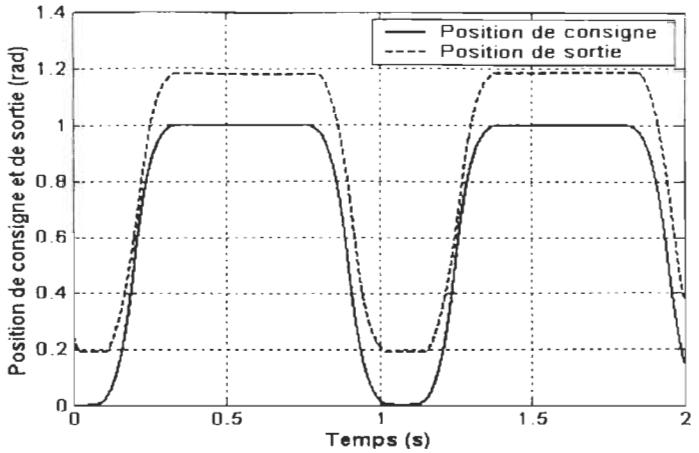

Fig. 5.5: Positions de consigne et de sortie; apprentissage hors ligne avec conditions initiales non nulles

5.2.2- Apprentissage en ligne:

La figure 5.6 illustre l'allure du taux d'apprentissage pour l'anticipation présenté dans la figure 4.2. Les paramètres de simulation sont les suivants:

$$\eta_{max} = 1^{e-1};\ \mu = 1^{e-3};\ \sigma = 25;$$

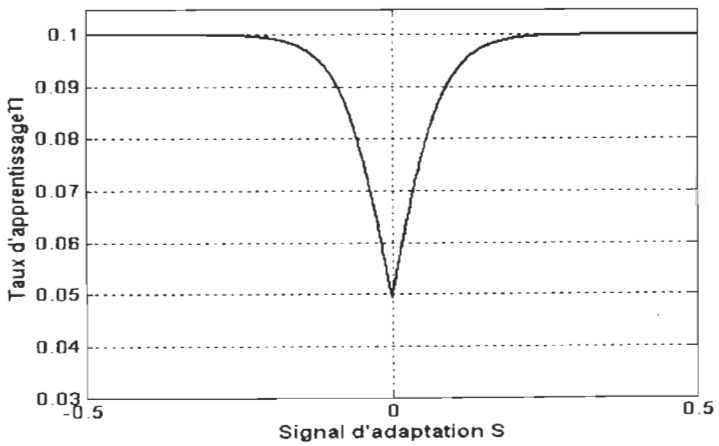

Fig. 5.6: Allure du taux d'apprentissage pour l'anticipation dans la figure 4.2.

Les figures 5.7, 5.8 et 5.9 désignent l'erreur de position entre la position de consigne et celle de la sortie, la position de la charge désirée et obtenue et l'erreur de position avec des conditions initiales non nulles lors de la simulation du modèle d'anticipation avec un apprentissage en ligne (code en annexe C).

L'apprentissage en ligne a été fait avec le terme de correction S indiqué dans (4.29) comme un signal de retour pour assurer la stabilité interne et celle de la sortie, le schéma de simulation est disponible dans l'annexe C. Nous remarquons une amélioration des résultats obtenus comparant avec ceux de l'apprentissage hors ligne.

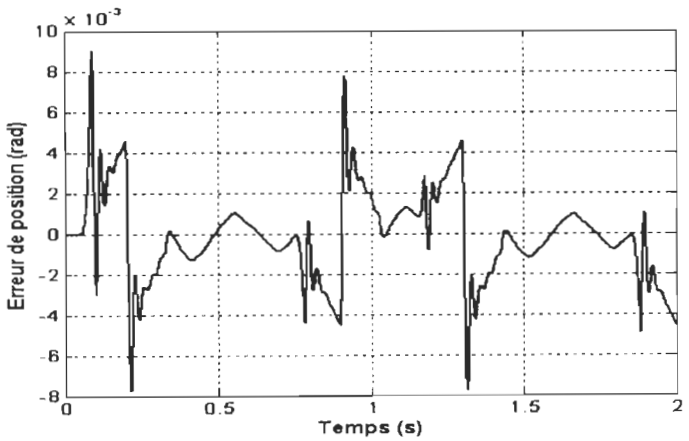

Fig. 5.7: Erreur de position charge en ligne avec conditions initiales nulles.

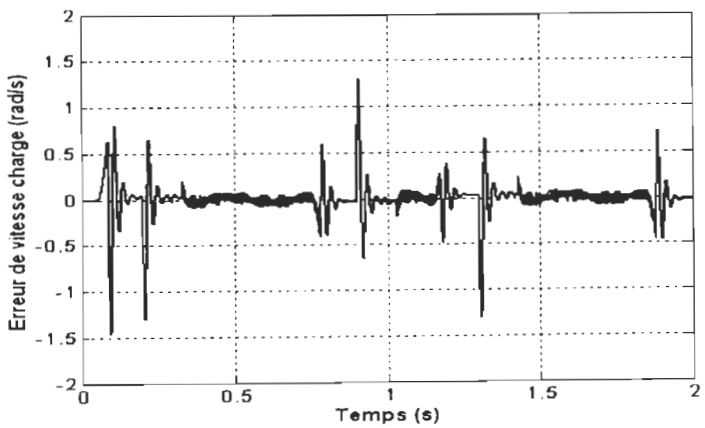

Fig. 5.8: Erreur de vitesse charge en ligne avec conditions initiales nulles.

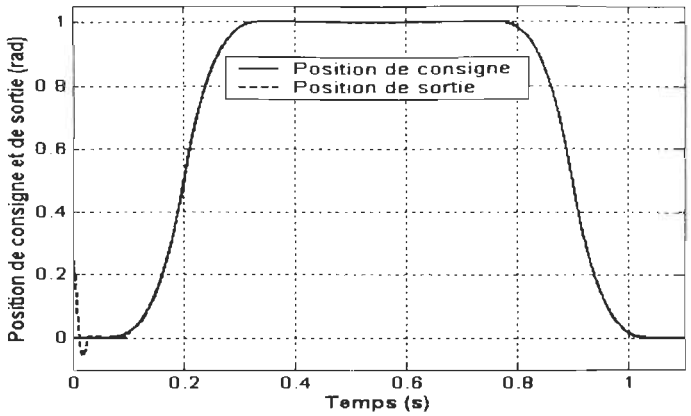

Fig. 5.9: Positions de consigne et de sortie; apprentissage en ligne avec conditions initiales non nulles.

Nous constatons une convergence très rapide face à l'erreur de position initiale, cette convergence se fait d'une façon exponentielle et représente la même allure du signal d'adaptation S présenté dans la figure 4.7. Nous remarquons que le temps de stabilisation est d'environ 0.04s. Si nous avons utilisé un échelon comme consigne, ce temps va être égale à

$$\iota = \frac{5}{p} = \frac{5}{130} = 0.038s .$$

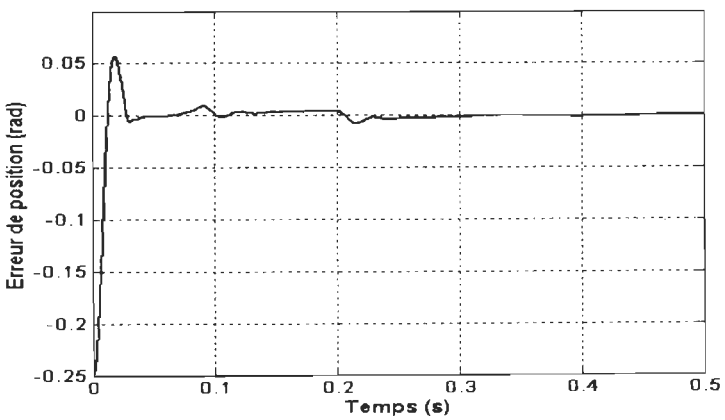

Fig. 5.10: Erreur de position charge en ligne avec conditions initiales non nulles.

L'augmentation de l'erreur à 0.1s et 0.2s est due à l'accélération brusque et aux changements du signe de la vitesse moteur causé par la flexion et la non-linéarité de la friction. La figure 5.11 démontre que même sans rétroaction et avec un apprentissage en ligne, nous pouvons assurer la stabilité interne de notre système. Le rôle de la rétroaction présenté dans la dernière structure va être de diminuer les oscillations et donner une dynamique en assurant la stabilité interne et d'entrée-sortie.

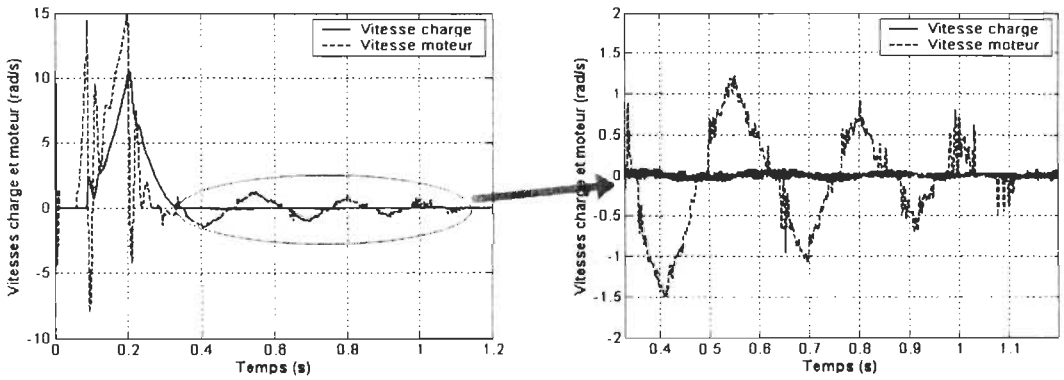

Fig. 5.11: Vitesses charge et moteur; apprentissage en ligne sans rétroaction et conditions initiales nulles.

5.3- Commande adaptative avec anticipation et rétroaction d'une articulation flexible avec non-linéarité:

5.3.1- Apprentissage hors ligne:

Les figures 5.12 et 5.13 présentent l'erreur entre la position de la charge désirée et obtenue et l'erreur entre la vitesse charge d'entrée et celle de la sortie lors de la simulation du modèle d'anticipation et de rétroaction avec un apprentissage hors ligne présenté par la figure 4.4. Le code et le schéma de simulation sont disponibles dans l'annexe B.

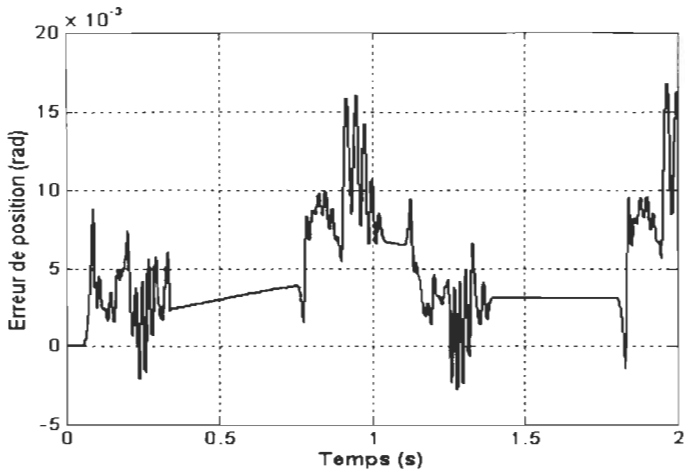

Fig. 5.12: Erreur de position charge; apprentissage hors ligne avec rétroaction et conditions initiales nulles.

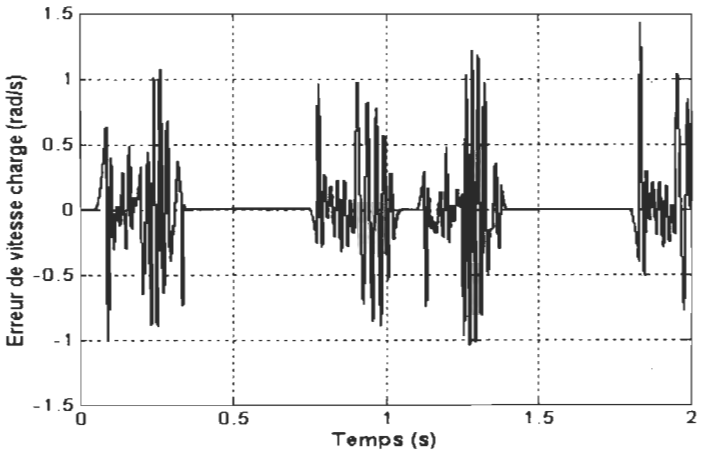

Fig. 5.13: Erreur de vitesse charge; apprentissage hors ligne avec rétroaction et conditions initiales nulles.

Nous remarquons une diminution des erreurs de position et de vitesse comparée avec celle de l'anticipation avec un apprentissage hors ligne présenté dans les figures 5.3 et 5.4. Par contre, nous constatons une perte dans la stabilité démontrée par les oscillations dans les figures 5.12 et 5.13, d'où l'utilisation de l'apprentissage en ligne des deux réseaux de neurones à la prochaine section.

La figure 5.14 présente les positions de consigne et de sortie avec la structure d'anticipation et rétroaction et des conditions initiales non nulles $\theta_m = \theta_c = 0.25$.

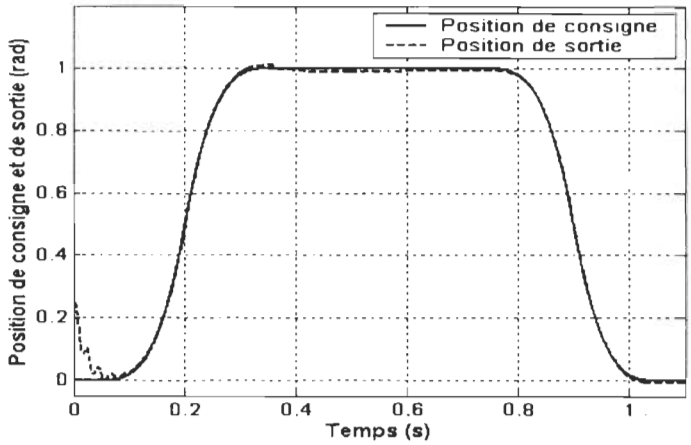

Fig. 5.14: Positions de consigne et de sortie avec rétroaction et conditions initiales non nulles.

Dans la figure 5.5, nous avons remarqué que l'anticipation n'était pas capable de faire face aux erreurs initiales parce que la structure simulée était un modèle en boucle ouverte et l'anticipation n'avait pas appris différentes conditions initiales de l'articulation lors de l'apprentissage hors ligne. Nous remarquons dans la figure 5.14 que la rétroaction a pu corriger ces erreurs initiales mais pas aussi rapidement que le résultat de l'apprentissage en ligne de l'anticipation présenté dans la figure 5.9. De plus, nous notons la présence d'une erreur stationnaire en position.

5.3.2- Apprentissage en ligne:

La figure 5.15 démontre l'allure du taux d'apprentissage utilisé pour l'anticipation présenté dans la figure 4.5. Les paramètres de simulation sont les suivants:

$$\eta_{max} = 3.7^{e+4};\ \mu = 2.5;\ \sigma = 1;$$

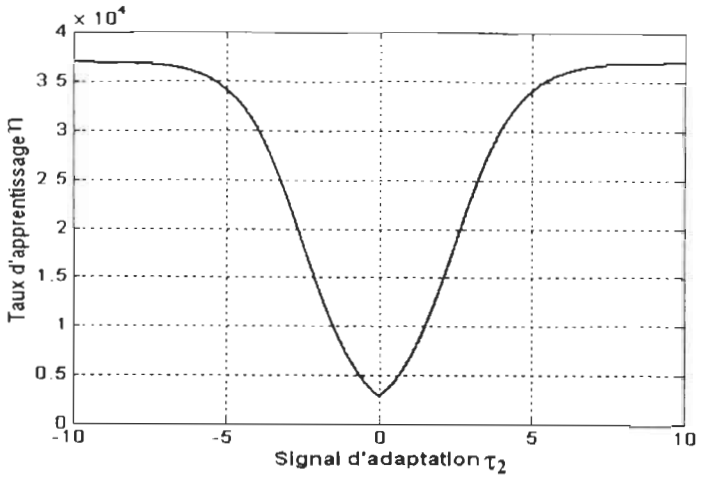

Fig. 5.15: Allure du taux d'apprentissage pour l'anticipation dans la figure 4.5.

Nous montrons dans la figure 5.16 l'allure du taux d'apprentissage pour la rétroaction présentée dans la figure 4.5. Les paramètres de simulation sont les suivants:

$$\eta_{max} = 2.7^{e-2};\ \mu = 1^{e-1};\ \sigma = 1;$$

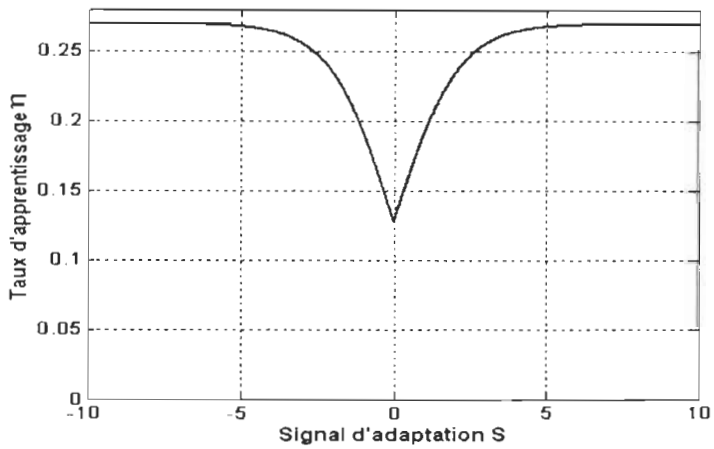

Fig. 5.16: Allure du taux d'apprentissage pour la rétroaction dans la figure 4.5.

La figure 5.17 présente l'erreur entre la position désirée de la charge et celle obtenue lors de la simulation du modèle d'anticipation et de rétroaction avec un apprentissage en ligne présenté par la figure 4.4. Le code et le schéma sont disponibles dans l'annexe C.

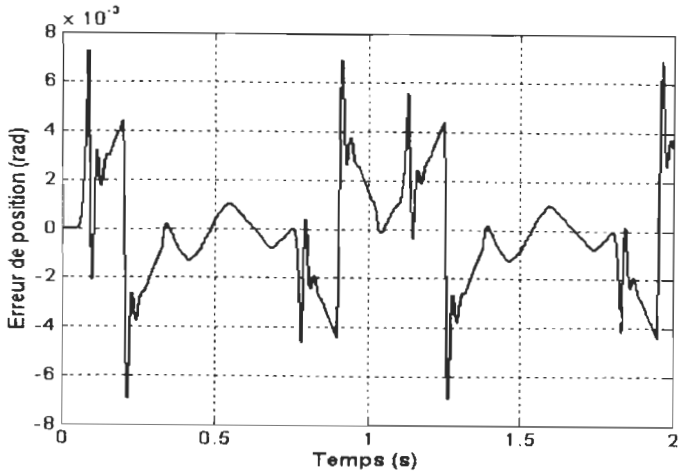

Fig. 5.17: Erreur de position charge; apprentissage en ligne avec rétroaction et conditions initiales nulles.

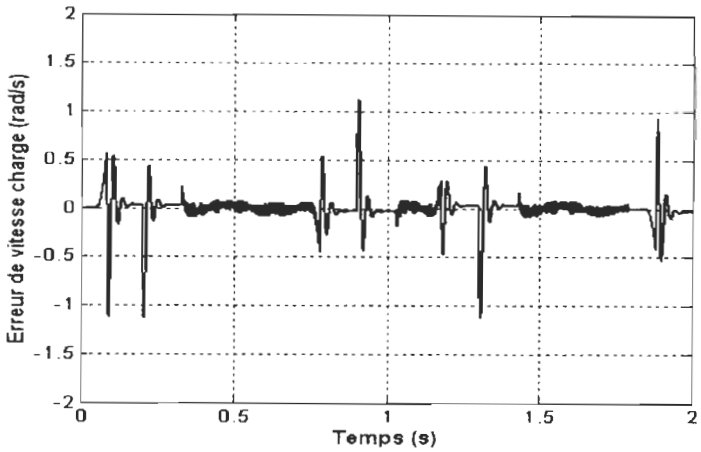

Fig. 5.18: Erreur de vitesse charge; apprentissage en ligne avec rétroaction et conditions initiales nulles.

La figure 5.18 représente l'erreur entre la vitesse désirée de la charge et celle obtenue lors de l'apprentissage en ligne de l'anticipation et la rétroaction avec une erreur initiale non nulle.

Nous constatons encore une fois une diminution des erreurs de position et de vitesse comparées avec les résultats présentés dans les figures 5.7 et 5.8. Nous remarquons une meilleure dynamique comparée avec les autres méthodes utilisées auparavant.

Nous avons fixé le taux d'apprentissage du réseau de l'anticipation et de la rétroaction tel qu'il est montré dans les figures 4.15 et 4.16 de façon à ce que l'anticipation fournie tout le signal de commande désiré, le rôle de la rétroaction sera de compenser le manque du signal d'anticipation.

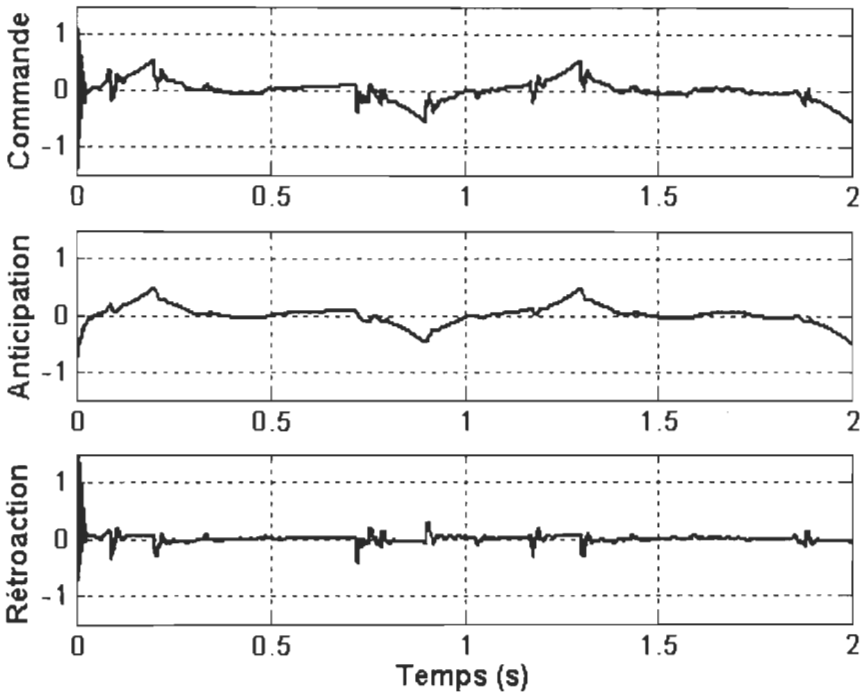

Fig. 5.19: Signal d'anticipation, de rétroaction et de commande.

La figure 5.19 montre que le réseau d'anticipation a réussi à représenter le signal de commande désiré avec une faible erreur. La rétroaction entre en jeux lorsque l'anticipation n'est pas capable de fournir tout le signal de commande à cause des accélérations brusques et des changements de signe de vitesse dus à la flexion et à la complexité du modèle de friction utilisé.

Pour s'assurer de la stabilité interne du système, nous présentons la vitesse de la charge et la vitesse du moteur dans la figure 5.20. Nous remarquons que les oscillations autour de la vitesse nulle diminuent au fur et à mesure que le temps de simulation augmente, ces dernières sont dues principalement au phénomène de la flexion.

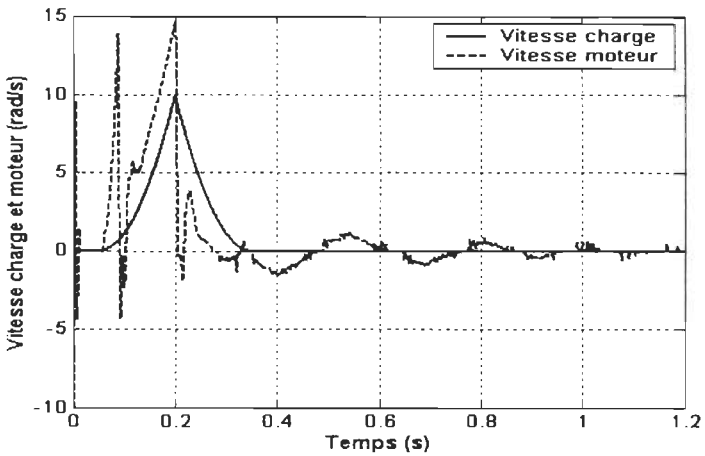

Fig. 5.20: Vitesses charge et moteur; apprentissage en ligne avec rétroaction et conditions initiales nulles.

La figure 5.21 présente la position de consigne et de la sortie pour une consigne 25 fois plus grande que précédemment afin de démontrer les propriétés d'adaptabilité du réseau. Nous constatons que la position de sortie a pu suivre celle de la consigne, nous remarquons aussi qu'il y a une amélioration par rapport au résultat présenté dans la figure 5.22 (apprentissage en ligne, sans rétroaction). La figure 5.23 présente les vitesses charge et moteur et démontre que la stabilité interne aussi a été assurée.

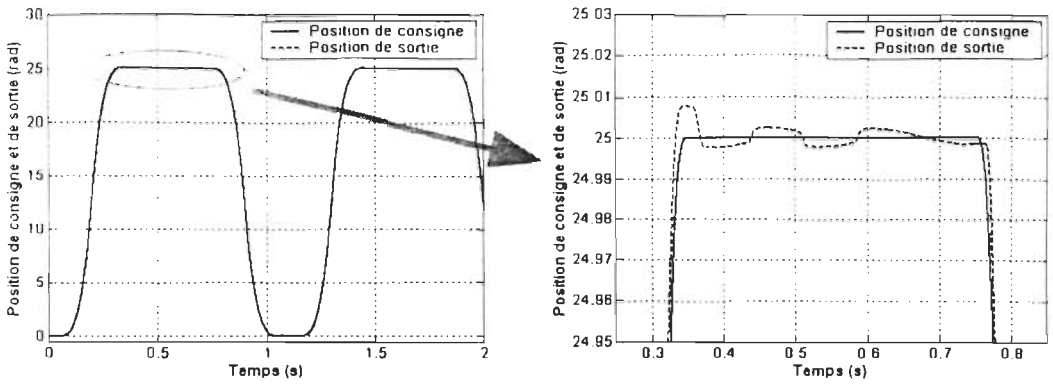

Fig. 5.21: Positions charge de consigne et de sortie; apprentissage en ligne avec une consigne de 25 rad.

La figure 5.22 présente la position de consigne et de la sortie pour une consigne de 25 rad dans le cas d'un apprentissage en ligne sans rétroaction. La figure 5.23 présente les vitesses charge et moteur et démontre que la stabilité interne a été aussi assurée.

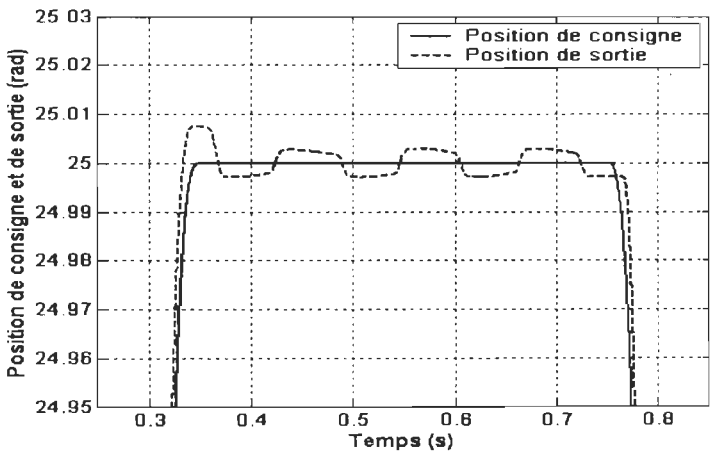

Fig. 5.22: Positions charge de consigne et de sortie en ligne sans rétroaction.

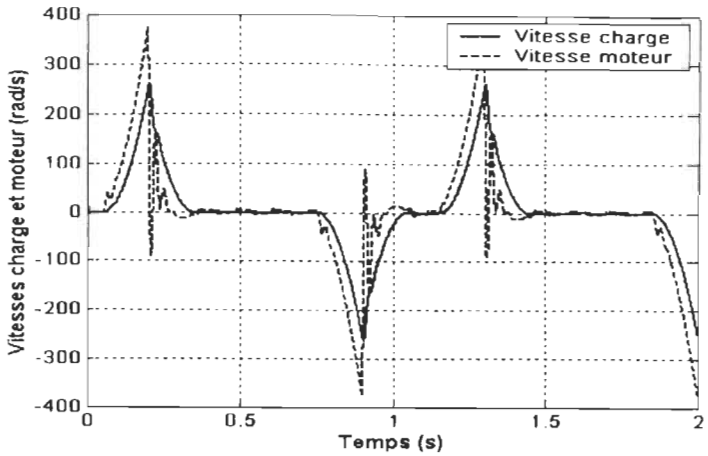

Fig. 5.23: Vitesses charge et moteur avec une consigne de 25 rad.

5.3.3 Comparaison quantitative des performances:

Nous présentons dans le Tableau 5.1 la valeur de la racine carrée de la somme des erreurs de position au carré en fonction des méthodes et des modes d'apprentissage utilisés.

Tableau 5.1: Racine carrée des erreurs au carrée pour les différents modes d'apprentissage.

Méthodes	$\sqrt{\sum e^2}$
- Apprentissage hors ligne sans rétroaction.	4.6954
- Apprentissage hors ligne avec rétroaction.	0.7750
- Apprentissage en ligne sans rétroaction.	0.3150
- Apprentissage en ligne avec rétroaction.	0.2986

Nous constatons que la précision s'est améliorée avec chaque modification apportée à la structure de commande.

5.4- Conclusion:

Dans ce chapitre, nous avons présenté des résultats de simulation qui montrent la performance de la loi de commande conçue à base de réseaux de neurones artificiels.

Nous avons conçu une anticipation avec un apprentissage hors ligne qui représente le modèle approximatif de l'inverse de l'articulation, mais ce modèle était impuissant contre les erreurs initiales. Nous avons introduit par la suite un terme de rétroaction pour corriger les erreurs de sortie. Cet ajout nous a permis de diminuer les erreurs entre l'entrée et la sortie. Avec le taux d'apprentissage variable pour l'entraînement en ligne, nous avons pu constater une convergence rapide face aux erreurs initiales tout en assurant la stabilité interne. Finalement, nous avons ajouté une rétroaction avec un apprentissage en ligne qui nous a permis d'améliorer la précision de sortie tout en maintenant une bonne stabilité interne. Enfin, nous pouvons dire que le modèle de l'anticipation a pu compenser les effets de la flexibilité et de la non-linéarité de la friction en apprenant une approximation du modèle inverse de l'articulation; la rétroaction est venue corriger les erreurs résiduelles.

Chapitre 6:

CONCLUSION

Les réseaux de neurones sont aujourd'hui de plus en plus utilisés compte tenu de leur parallélisme, de leur clarté de structure et de leur capacité à pouvoir classifier des problèmes non-linéaires. Nous avons présenté dans ce travail, une stratégie de commande avec différents algorithmes d'apprentissage pour une articulation flexible avec caractéristique de friction non-linéaire. Nous avons conçu un réseau de neurones capable de fournir un signal d'anticipation pour l'articulation en apprenant la fonction de sa caractéristique inverse. Nous avons présenté

aussi un autre réseau de rétroaction qui vient s'ajouter au premier pour une correction d'erreur et apporter une certaine dynamique au système de positionnement. Les deux réseaux ont été entraînés avec un apprentissage hors ligne. Ensuite, nous avons appliqué l'algorithme de rétro-propagation d'erreur pour concevoir nos contrôleurs permettant de faire la correction et la compensation des phénomènes de flexion et friction. Les signaux d'adaptations pour les réseaux ont été choisis de façon à assurer une stabilité interne et d'entrée-sortie avec l'utilisation de taux d'apprentissage variable.

Nos résultats ont démontré que les structures de commande ont résulté en une qualité de compensation variable des effets de la non-linéarité de la friction et de la flexion dus au ressort de torsion. Nous avons obtenu une bonne stabilité interne avec juste un réseau d'anticipation entraîné en ligne tout en assurant une stabilité d'entrée-sortie. L'apprentissage en ligne donne de meilleurs résultats que l'entraînement hors ligne.

Ce travail nous a permis d'apporter une contribution technologique pour les systèmes de positionnement de haute performance. Nous avons développé une nouvelle structure de commande à base de réseaux de neurones qui peut être utilisés dans plusieurs domaines tel que la robotique.

Nous avons supposé que le ressort de torsion a une caractéristique linéaire, ce qui n'est pas toujours le cas dans la pratique. Plusieurs travaux peuvent venir compléter le travail qui a été fait jusqu'à présent, en considérant un modèle non-linéaire du ressort, et concevoir des lois de commande robustes. D'autres travaux peuvent porter sur le développement d'autres algorithmes d'apprentissage pour un but de diminuer le temps de calcul à l'implantation.

Références:

[1] L.M Sweet et M.C. Good, *«Redefinition of the robot motion control problem: effects of plant dynamics, drive system contraints, and user requirement»*, Proceedings of the 23rd IEEE conference on Decision and Control, pp 724-31, 1984.

[2] M.W. Spong, *«Adaptative control of flexible-joint manipulators»*, Systems & Control Letters, Vol. 13, pp. 15-21, 1989.

[3] Pierre E. Dupont, *«Friction Modeling in Dynamic Robot Simulation»*, Proceedings of the 1990 IEEE conference on Robotics and Automation, Cincinati, Ohio, Mai 1990.

[4] K. Kozlowski et P. Sauer *«On adaptive control of flexible joint manipulators: theory and experiments»* Proceedings of the IEEE International Symposium on Industrial Electronics. ISIE '99. Volume: 3. Pages: 1153-1158, 1999.

[5] F. Ciuca, T. Lahdhiri et H.A ElMaraghy *«Linear robust motion control of flexible joint robots. I. Modeling»* Proceedings of the American Control Conference. Volume: 1. Pages: 699-703, 1999.

[6] B. Armstrong et C. Canudas de Wit, *«Friction modeling and compensation»*, The control handbook, chap. 77, W. S. Levine, éditeur, IEEE Press, 1996.

[7] O. Aboulshamat et P. Sicard *«Position control of a flexible joint with friction using neural network feedforward inverse models»* Canadian Conference on Electrical and Computer Engineering. Volume: 1. Pages: 283-288, 2001.

[8] S. Lesueur *«Implantation en technologie ITGE (VLSI) d'une loi de commande pour une articulation flexible»* Mémoire (M.Sc.A.) – Université du Québec à Trois-Rivières, 1999.

[9] F.J. Lin *«Robust speed-controlled induction-motor drive using EKF and RLS estimators»*, IEEE proc, Electr. Power appl. 143, (3), pp. 186-192, 1996.

[10] Kazuo Kiguchi et Toshio Fukuda *«Fuzzy Neural Friction Compensation Method of Robot Manipulation During Position / Force Control»*, Proceedings of the 1996 IEEE. International Conference on Robotics and Automation, Minneapolis, Minnesota, April 1996 .

[11] P. Sicard *«Trajectory tracking of flexible joint manipulators with passivity based Controller»*, Thèse de doctorat, Rensselaer Polytechnic Institute, États-Unis, 1993.

[12] W. Leonhard *«Control of Electrical Drives»*, Springer Verlag, Berlin, 1985.

[13] P. Dorato *«Robust control: A Historical Review»*, IEEE Control systems, pp 44-46, New York, 1987.

[14] G. Zames *«Feedback and optimal sensitivity model reference transformations, multiplicative seminorms, and approximate inverses»*, IEEE Trans Automatic Control, AC-26, pp 319-338, 1981.

[15] P. Sicard *«Commande de position d'un moteur à courant continu par correcteurs adaptatifs passifs développés à l'aide de la théorie du réglage par mode de glissement»* Mémoire (M.Sc.A.) – Université du Québec à Trois-Rivières, 1989.

[16] V.I. Utkin *«Sliding mode control design principles and applications to electric drives»*, IEEE trans. Ind. Electron. 40, (1), pp. 23-36, 1993.

[17] Anand, Mehrotra, Mohan et Ranka *«Intelligent Control Using Neural Networks»*, IEEE Control System mag; Avril, pp 11-18, 1992.

[18] R.L. Watrous *«Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization»*, IEEE First International Conference on Neural Networks, 2, pp 619-627, 1987.

[19] Ozdamar, Yaylali, Tayakar et Lopez *«Inversion of Multilayer Networks»*. Int, Joint. Conf. Neural Networks, Washington, Juin, pp 425-430, 1989.

[20] F.J. Pineda *«Generalization of Back-Propagation to Recurent Neural Networks»*, Physical Review letters, 59(19), pp 2229-2232, 1987.

[21] K.S. Narendra, et K. Parthasarathy *«Gradient Methods for the Optimization of Dynamical Systems Containing Neural Networks»*, IEEE Trans on Neural Networks, 2(2), pp 252-262, 1991.

[22] K.S. Narendra et K. Parthasarathy *«Identification and Control of Dynamical Systems Using Neural Networks»*, IEEE Trans on Neural Networks, 1(1), pp 4-27, 1990.

[23] Mavrovouniotis et Chang Numerical Recipes in C. New York : Cambridge University Press, 1990.

[24] Wen et Ozdamar *«A Comparaison of Control Strategies of Robotic Manipulators Using Neural Networks»*. International Conference on Industrial Electronics, Control, Instrumentation and Automation. Nov(2), pp 688-693, 1991.

[25] D.E. Rumelhart, G.E. Hinton et R.J. Williams *«Learning Internal Representations by error Propagation»*, Parallel Distributed Processing, 1(8), pp 319-362, 1986.

Annexes:

Annexe A: Schémas et programmes de simulation de l'articulation flexible dans Simulink™.

Annexe B: Schémas et programmes de simulation des modèles d'apprentissage hors ligne sur Simulink™ et Matlab®.

Annexe C: Schémas et programmes de simulation des modèles d'apprentissage en ligne sur Simulink™ et Matlab®.

Annexe A: Schémas et programmes de simulation de l'articulation flexible dans Simulink^{TM}.

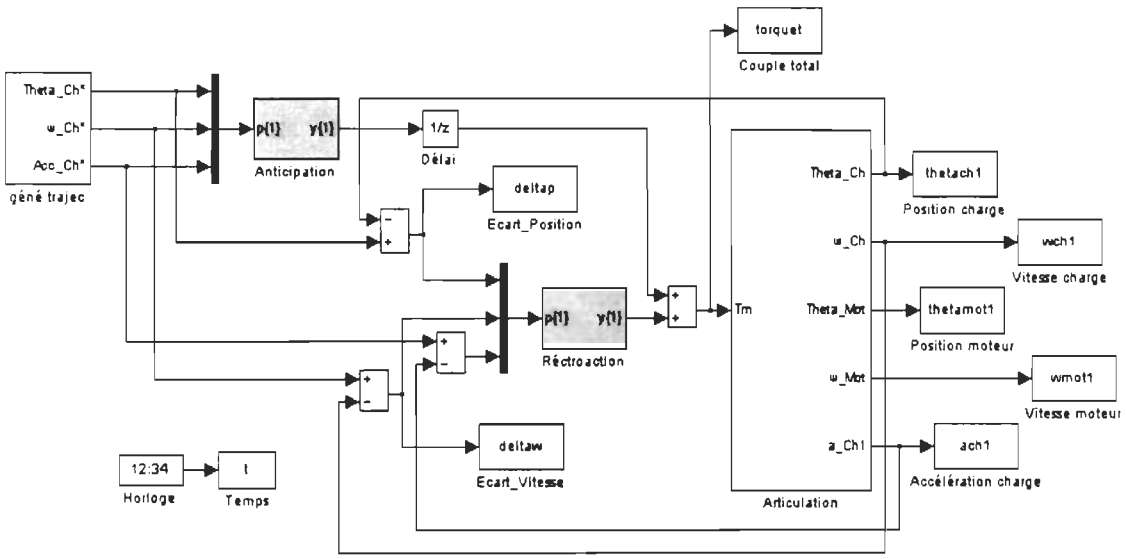

Fig. A.1: Système complet: articulation flexible et ses contrôleurs

A.1- Articulation flexible:

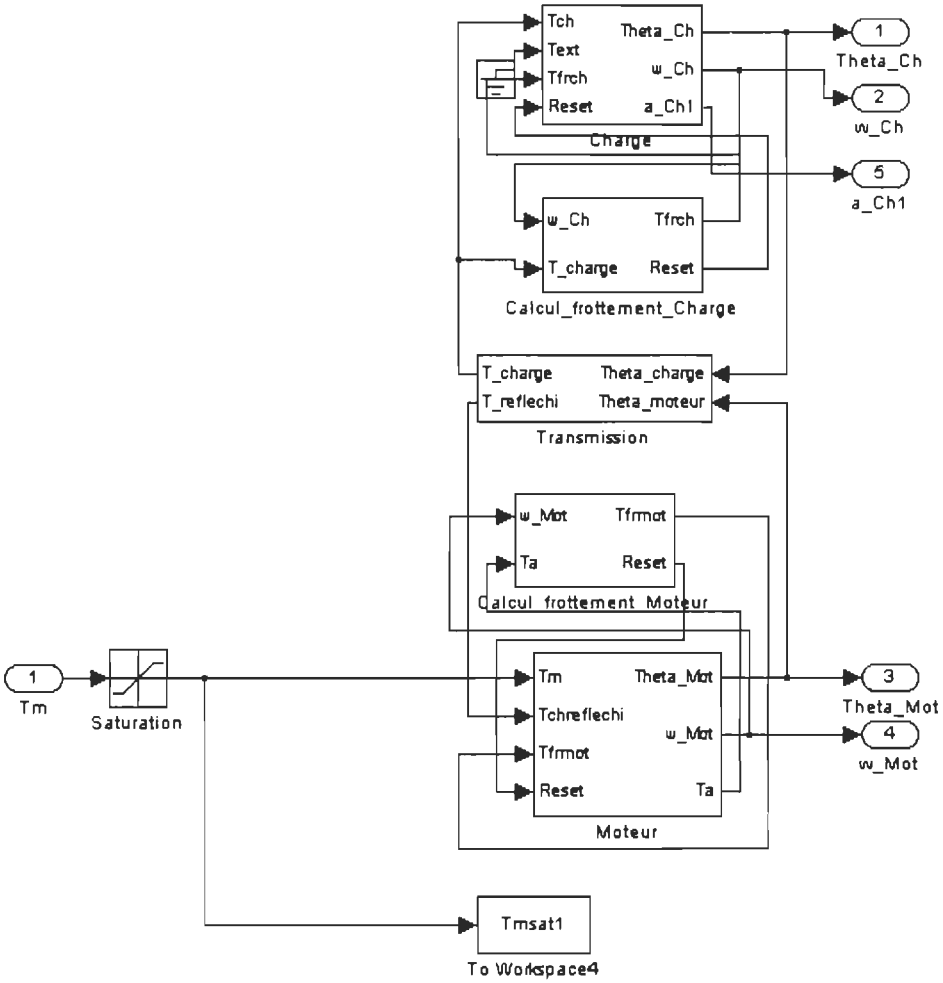

Fig. A.2: Modèle de l'articulation flexible.

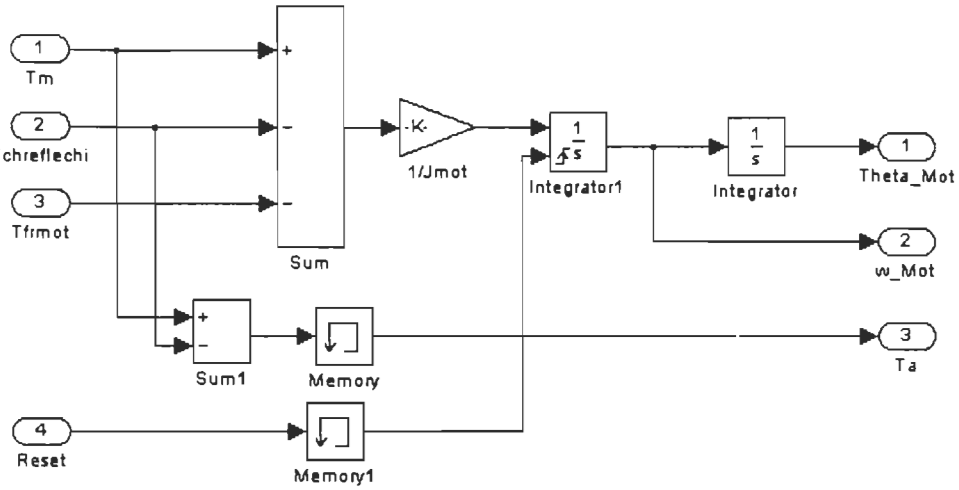

Fig. A.3: Modèle dynamique du moteur

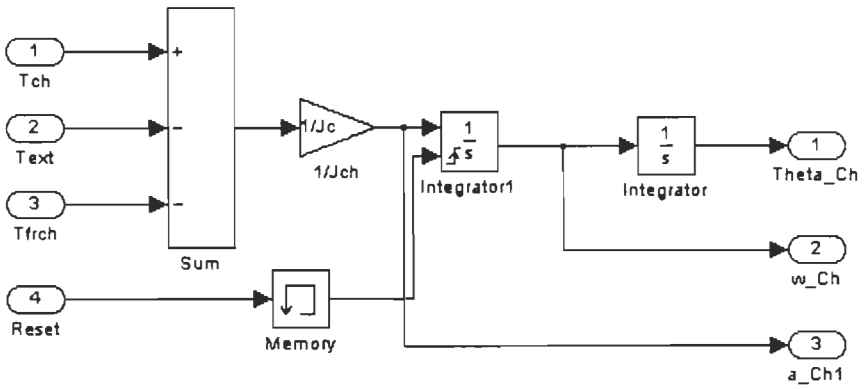

Fig. A.4: Modèle dynamique de la charge

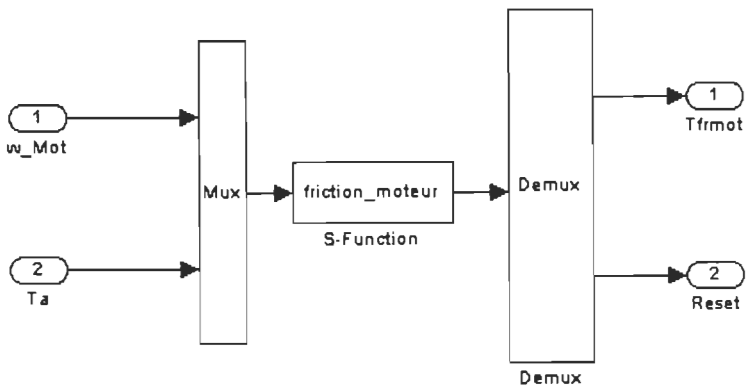

Fig. A.5: Modèle de friction du moteur (code à la page 106)

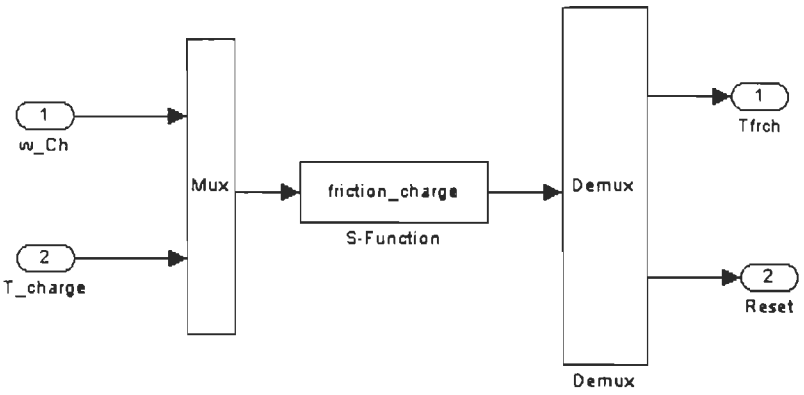

Fig. A.6: Modèle de friction de la charge (code à la page 103)

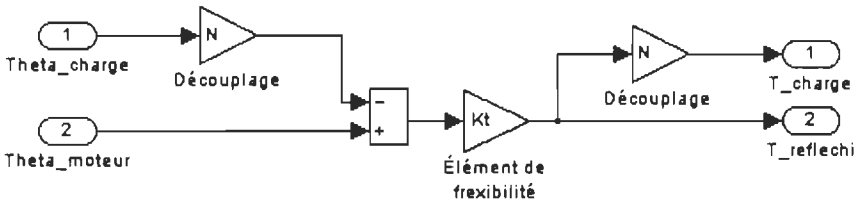

Fig. A.7: Modèle de la boîte de transmission

A.2- Générateur de trajectoire:

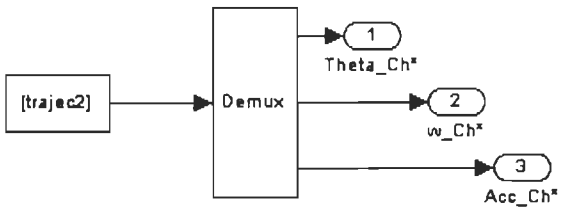

Fig. A.8: Générateur de trajectoire pour le réseau de neurones

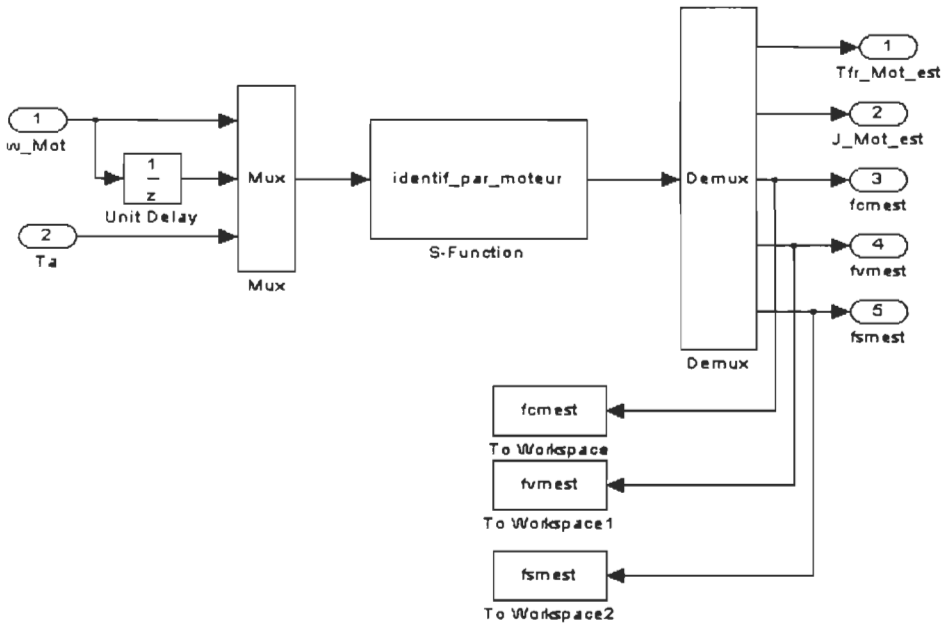

Fig. A.9: Estimateur des paramètres du moteur (code à la page 113)

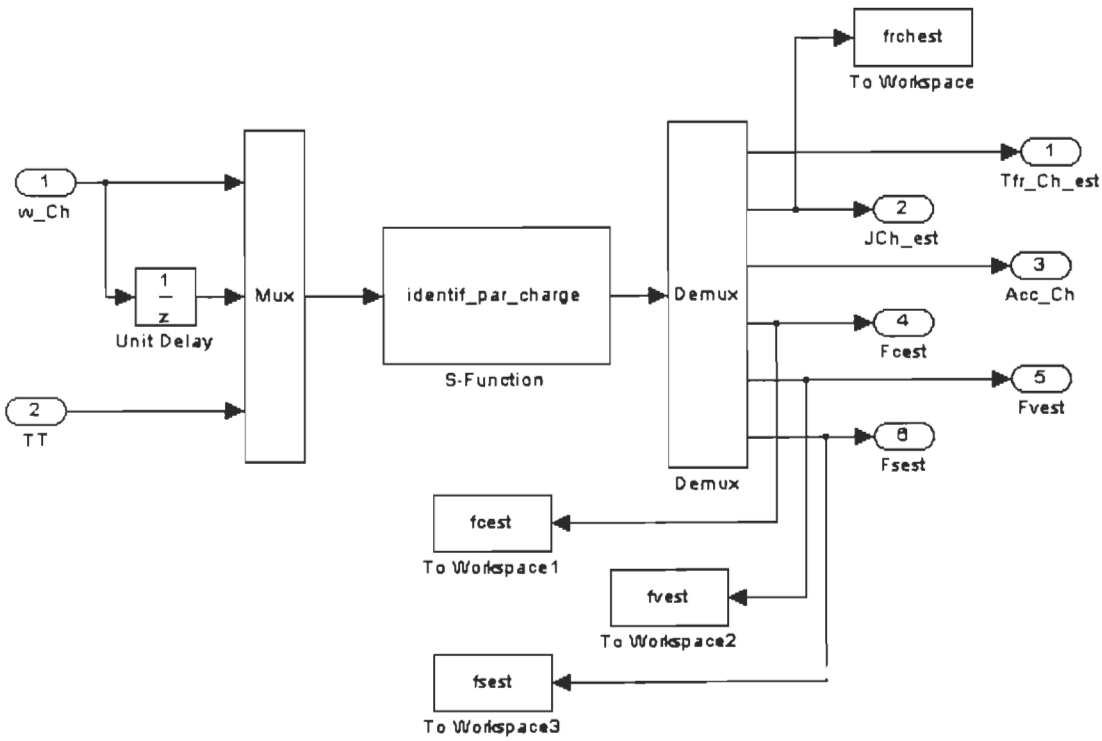

Fig. A.10: Estimateur des paramètres de la charge (code à la page 109)

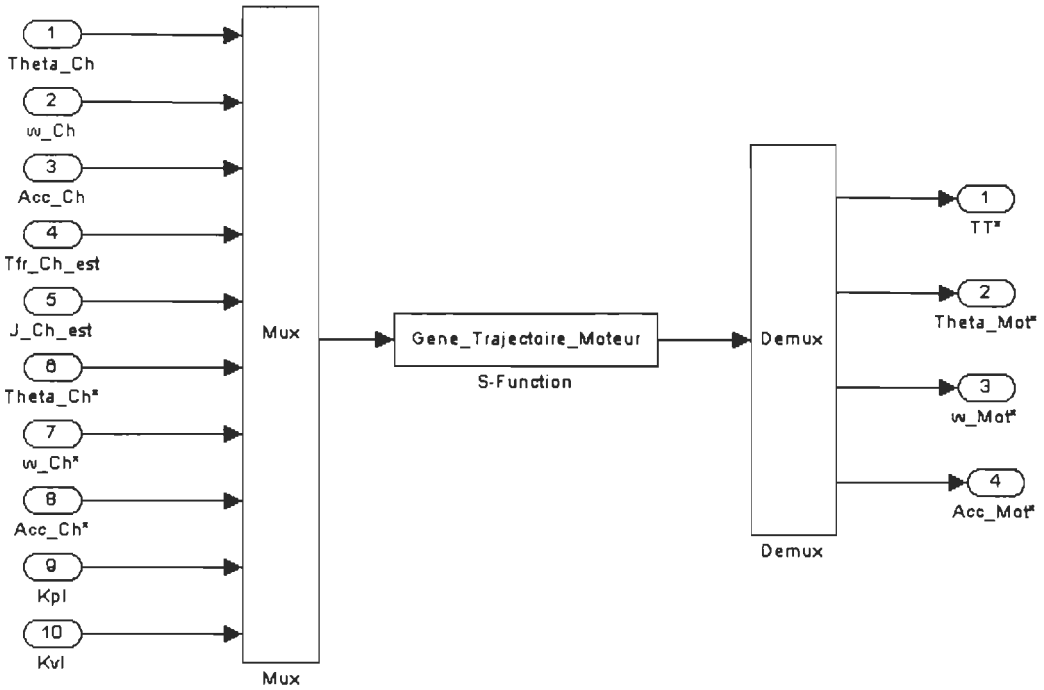

Fig. A.11: Génération de trajectoire pour le moteur (code à la page 97)

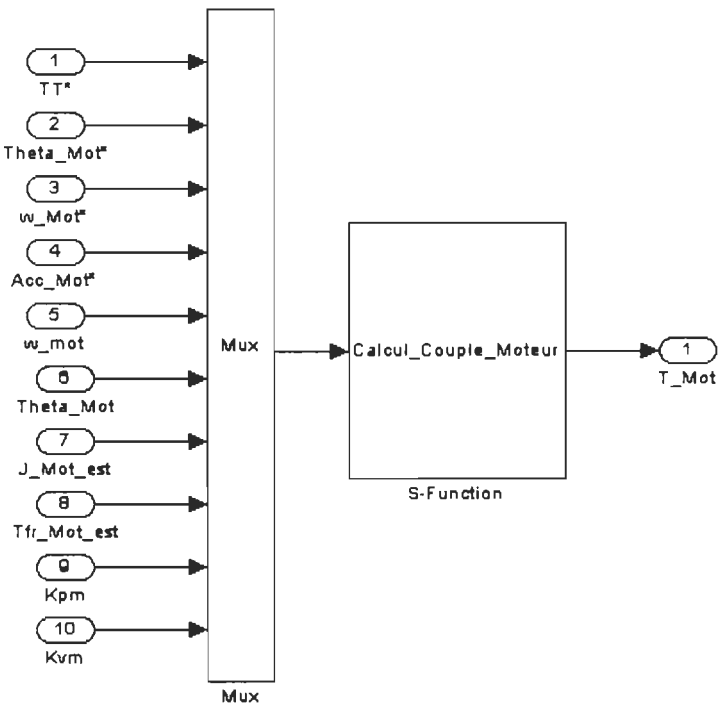

Fig. A.12: Calcul de couple de commande du moteur (code à la page 100)

```
/*
 * Calcul de la trajectoire désirée pour le moteur
 *
 * Entrees :    Position charge ThCh
 *              Vitesse charge wCh
 *              Accélération charge AccCh
 *              Couple de frottement charge estimé TfrChest
 *              Inertie charge estimée JChest
 *              Position charge désirée ThCh*
 *              Vitesse charge désirée wCh*
 *              Accélération charge désirée AccCh*
 *              Gain proportionnel sur l'erreur de position de la charge Kpl
 *              Gain proportionnel sur l'erreur de vitesse de la charge Kvl
 *
 * Sorties : Couple de charge désiré TT*
 *              Position moteur désirée ThMot*
 *              Vitesse moteur désirée wMot*
 *              Accélération moteur désirée AccMot*
 *
 * Designation : u[0]=ThCH; u[1]=wCh; u[2]=AccCh; u[3]=TfrChest;
 *              u[4]=JChest; u[5]=ThCh*; u[6]=wCh*; u[7]=AccCh* ; u[8]=kpl; u[9]=kvl;
 *              y[0]=TT*; y[1]=ThMot*; y[2]=wMot*; y[3]=Accmot*;
 * Crée par: Sébastien lesueur.
 */
#define S_FUNCTION_NAME Gene_Trajectoire_Moteur
#include "simstruc.h"
#include "math.h"
/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
   ssSetNumContStates(    S, 0);   /* number of continuous states          */
   ssSetNumDiscStates(    S, 0);   /* number of discrete states            */
```

```
    ssSetNumInputs(       S, 10);/* number of inputs        */
    ssSetNumOutputs(      S, 4);  /* number of outputs      */
    ssSetDirectFeedThrough(S, 1); /* direct feedthrough flag */
    ssSetNumSampleTimes(  S, 1);  /* number of sample times  */
    ssSetNumSFcnParams(   S, 0);  /* number of input arguments */
    ssSetNumRWork(        S, 0);  /* number of real work vector elements */
    ssSetNumIWork(        S, 0);  /* number of integer work vector elements*/
    ssSetNumPWork(        S, 0);  /* number of pointer work vector elements*/
}
/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}
/*
 * mdlOutputs - compute the outputs
 */
static void mdlOutputs(double *y, double *x, double *u, SimStruct *S, int tid)
{
  double N, k;
  N=1.0;
  k=0.50;
  y[0]=(1/N)*(u[4]*u[7]+u[3]+u[9]*(u[6]-u[1])+u[8]*(u[5]-u[0]));
  y[1]=(y[0]/k)+N*u[5];
  y[2]=N*u[6]+(1/k)*(u[9]*(u[7]-u[2])+u[8]*(u[6]-u[1]));
```

```
  y[3]=N*u[7]+(u[8]/k)*(u[7]-u[2]);
}
/*
 * mdlUpdate - perform action at major integration time step
 */
static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}
/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct *S, int tid)
{
}
/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}
#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

```
/*
 * Calcul du couple de commande du moteur
 *
 * Entrees :    Couple de charge desire TT*
 *              Position moteur désirée ThMot*
 *              Vitesse moteur désirée wMot*
 *              Accélération moteur désirée AccMot*
 *              Vitesse moteur mesurée wMot
 *              Position moteur mesurée Thmot
 *              Inertie moteur estimée JMotest
 *              Couple de frottement moteur estimé TfrMotest
 *              Gain proportionnel sur l'erreur de position moteur Kpm
 *              Gain proportionnel sur l'erreur de vitesse moteur Kvm

 * Sortie : Couple de commande du moteur Tm
 *
 * Designation : u[0]=TT*; u[1]=ThMot*; u[2]=wMot*; u[3]=AccMot*;
 *              u[4]=wMot; u[5]=ThMot; u[6]=JMotest; u[7]=TfrMotest;
 *              u[8]=Kpm; u[9]=Kvm; y[0]=Tm
 * Crée par: Sébastien lesueur.
 */
#define S_FUNCTION_NAME Calcul_Couple_Moteur
#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
   ssSetNumContStates(    S, 0);   /* number of continuous states        */
   ssSetNumDiscStates(    S, 0);   /* number of discrete states          */
   ssSetNumInputs(        S, 10);/* number of inputs         */
   ssSetNumOutputs(       S, 1);   /* number of outputs     */
```

```
  ssSetDirectFeedThrough(S, 1);  /* direct feedthrough flag              */
  ssSetNumSampleTimes(  S, 1);  /* number of sample times               */
  ssSetNumSFcnParams(   S, 0);  /* number of input arguments            */
  ssSetNumRWork(        S, 0);  /* number of real work vector elements  */
  ssSetNumIWork(        S, 0);  /* number of integer work vector elements*/
  ssSetNumPWork(        S, 0);  /* number of pointer work vector elements*/
}
/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
  ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
  ssSetOffsetTime(S, 0, 0.0);
}
/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}
/*
 * mdlOutputs - compute the outputs
 */
static void mdlOutputs(double *y, double *x, double *u, SimStruct *S, int tid)
{
  double n;
   n=1.00;
  y[0]=u[6]*u[3]+u[7]+n*u[0]+u[9]*(u[2]-u[4])+u[8]*(u[1]-u[5]);
}
/*
 * mdlUpdate - perform action at major integration time step
 */
```

```
static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}
/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct *S, int tid)
{
}
/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}
#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

```
/*
 * Calcul du terme de frottement
 * Modele = frottement visqueux + Coulomb + statique (Stribeck)  [figure 2.4]
 *
 * Entree : Vitesse et couple charge/moteur
 * Sortie : Couple de friction
 * Designation : u[0]=vitesse; u[1]=couple; y[0]=couple de friction; y[1]=Reset.
 */

#define S_FUNCTION_NAME friction_charge
#include "simstruc.h"
#include "math.h"
static real_T rest;
/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
   ssSetNumContStates(    S, 0);   /* number of continuous states         */
   ssSetNumDiscStates(    S, 0);   /* number of discrete states           */
   ssSetNumInputs(        S, 2);/* number of inputs         */
   ssSetNumOutputs(       S, 2);   /* number of outputs     */
   ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag             */
   ssSetNumSampleTimes(   S, 1);   /* number of sample times              */
   ssSetNumSFcnParams(    S, 0);   /* number of input arguments           */
   ssSetNumRWork(         S, 0);   /* number of real work vector elements   */
   ssSetNumIWork(         S, 0);   /* number of integer work vector elements*/
   ssSetNumPWork(         S, 0);   /* number of pointer work vector elements*/
}
/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
```

```
  ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
  ssSetOffsetTime(S, 0, 0.0);
}
/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}
/*
 * mdlOutputs - compute the outputs
 */
static void mdlOutputs(double *y, double *x, double *u, SimStruct *S, int tid)
{
  double Fcm, Fsm, Fvm, vsm, Fcc, Fsc, Fvc, vsc,epsilo, Tc, temp1, signev, signet;
  epsilo=1e-6;
  if (u[0]>=0.0)
    signev=1.0;
  else
    signev=-1.0;
  if (u[1]>=0.0)
    signet=1.0;
  else
    signet=-1.0;
 temp1=u[0]/vs;
 Tc=Fcc+Fvc*fabs(u[0])+Fsc*exp(-(temp1*temp1));
 if (fabs(u[0])>=epsilo)
                {
                y[0]=Tc*signev;
                y[1]=0;
                }
 else
        {
                if (fabs(u[1])>Tc)
```

```
                {
                y[0]=Tc*signet;
                y[1]=0;
                }
        else
                {
                y[0]=u[1];
                y[1]=1;
                }
        }
}
/*
 * mdlUpdate - perform action at major integration time step
 */
static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}
/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct *S, int tid)
{
}
/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}
#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

```
/*
 * Calcul du terme de frottement
 * Modele = frottement visqueux + Coulomb + statique (Stribeck)  [figure 2.4]
 *
 * Entree : Vitesse et couple charge/moteur
 * Sortie : Couple de friction
 * Designation : u[0]=vitesse; u[1]=couple; y[0]=couple de friction; y[1]=Reset.
 */

#define S_FUNCTION_NAME friction_moteur
#include "simstruc.h"
#include "math.h"
static real_T rest;
/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
   ssSetNumContStates(    S, 0);   /* number of continuous states          */
   ssSetNumDiscStates(    S, 0);   /* number of discrete states            */
   ssSetNumInputs(        S, 2);/* number of inputs         */
   ssSetNumOutputs(       S, 2);   /* number of outputs     */
   ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag              */
   ssSetNumSampleTimes(   S, 1);   /* number of sample times               */
   ssSetNumSFcnParams(    S, 0);   /* number of input arguments            */
   ssSetNumRWork(         S, 0);   /* number of real work vector elements   */
   ssSetNumIWork(         S, 0);   /* number of integer work vector elements*/
   ssSetNumPWork(         S, 0);   /* number of pointer work vector elements*/
}
/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
```

```
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}
/*
 * mdlOutputs - compute the outputs
 */
static void mdlOutputs(double *y, double *x, double *u, SimStruct *S, int tid)
{
  double Fcm, Fsm, Fvm, vsm, Fcc, Fsc, Fvc, vsc,epsilo, Tc, temp1, signev, signet;
  epsilo=1e-6;
  if (u[0]>=0.0)
    signev=1.0;
  else
    signev=-1.0;
  if (u[1]>=0.0)
    signet=1.0;
  else
    signet=-1.0;
 temp1=u[0]/vs;
 Tc=Fcm+Fvm*fabs(u[0])+Fsm*exp(-(temp1*temp1));
 if (fabs(u[0])>=epsilo)
        {
        y[0]=Tc*signev;
  y[1]=0;
        }
 else
        {
        if (fabs(u[1])>Tc)
```

```
                {
                y[0]=Tc*signet;
                y[1]=0;
        }
        else
                {
                y[0]=u[1];
                y[1]=1;
                }
        }
}
/*
 * mdlUpdate - perform action at major integration time step
 */
static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}
/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct *S, int tid)
{
}
/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}
#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

```
%% IDENTIFICATION DES PARAMETRES DE LA CHARGE

% Entrees :  Vitesse de rechargement
%            Couple applique a la charge TT

% Sorties :  Couple de frottement estime TFT
%            Inertie charge estimee JL

% Fait par: Sebastien Lessard

function [sys,x0,str,ts] = identif_par_charge(t,x,u,flag)

% Initialisation des constantes de simulation

lambda=1.00;
vs=0.95;
epsilo=1e-4;
T=1e-4;
Jchi=5.5e-5;
Fc=0.0288;
Fv=5.5e-4;
Fs=0.0191;

switch flag,

  %%%%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%%%
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs);

  %%%%%%%%%%%%%%%%%%
  % Update %
  %%%%%%%%%%%%%%%%%%
  case 2,
    sys=mdlUpdate(t,x,u,lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs);

  %%%%%%%%%%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%%%%%%%%%
  case 3,
    sys=mdlOutputs(t,x,u,lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs);

  case {1,4,9},
    sys=[];

  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

%% mdlInitializeSizes

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-
function.
```

```
function
[sys,x0,str,ts]=mdlInitializeSizes(lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs)

  % Call simsizes for a sizes structure, fill it in and convert it to a
  % sizes array.

  % Note that in this example, the values are hard coded.  This is not a
  % recommended practice as the characteristics of the block are typically
  % defined by the S-function parameters.

sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 20;
sizes.NumOutputs     = 6;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);

  % Initialize the initial conditions

x0  = [0.5*1/Jchi
       0.5*Fc/Jchi
       0.5*Fv/Jchi
       0.5*Fs/Jchi
       1200
       0
       0
       0
       0
       1200
       0
       0
       0
       0
       1200
       0
       0
       0
       0
       1200];

  % str is always an empty matrix

str = [];

  % Initialize the array of sample times

ts  = [T 0];

  % end mdlInitializeSizes
```

```
function sys=mdlUpdate(t,x,u,lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs)

acccalc=(u(1)-u(2))/T;
if (abs(acccalc)>0.0001)&(abs(u(1))>0.005)
      P=[x(5:8,1)'
         x(9:12,1)'
         x(13:16,1)'
         x(17:20,1)'];

      phi=[u(3)
           -sign(u(1))
           -u(1)
            -sign(u(1))*exp(-((u(1)/vs)^2))];

      P=(1/lambda)*P-(1/lambda)*(P*phi)/(lambda+phi'*P*phi)*phi'*P;
      acccalc=(u(1)-u(2))/T;
      alpha=acccalc-phi'*x(1:4,1);
      sys=[x(1:4,1)+P*phi/(lambda+phi'*P*phi)*alpha
           P(1,1:4)'
           P(2,1:4)'
           P(3,1:4)'
           P(4,1:4)'];
   else
      sys=x;
end

function sys=mdlOutputs(t,x,u,lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs)

if (abs(u(1))>epsilo)
   sys=[x(2:4,1)'/x(1,1)*[sign(u(1));u(1);sign(u(1))*exp(-((u(1)/vs)^2))]
        1/x(1,1)
        (u(1)-u(2))/T
        max(0,x(2,1)/x(1,1))
        max(0,x(3,1)/x(1,1))
        max(0,x(4,1)/x(1,1))];

  else
     Tc=x(2,1)/x(1,1)+x(3,1)/x(1,1)*abs(u(1))+x(4,1)/x(1,1)*exp(-
(u(1)/vs)^2);

      if (abs(u(3))>Tc)
         sys=[Tc*sign(u(3))
```

```
                1/x(1,1)
                (u(1)-u(2))/T
                max(0,x(2,1)/x(1,1))
                max(0,x(3,1)/x(1,1))
                max(0,x(4,1)/x(1,1))];
        else
           sys=[u(3)
                1/x(1,1)
                (u(1)-u(2))/T
                max(0,x(2,1)/x(1,1))
                max(0,x(3,1)/x(1,1))
                max(0,x(4,1)/x(1,1))];
        end

end

end mdlOutputs
```

```
                IDENTIFICATION DES PARAMETRES DU MOTEUR

 Entrees :  Vitesse du moteur wmot
            Couple appliqué au moteur Ta

 Sorties :  Couple de frottement estime CFV
            Inertie moteur estimee JM

                Fait par: Sébastien Lesue...

function [sys,x0,str,ts] = identif_par_moteur(t,x,u,flag)

  Initialisation des constantes de simulation

lambda=1.00;
vs=0.95;
epsilo=1e-4;
T=1e-4;
Jchi=5.5e-5;
Fc=0.0288;
Fv=5.5e-4;
Fs=0.0191;

switch flag,

  % Initialization %

  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs);

  % Update %

  case 2,
    sys=mdlUpdate(t,x,u,lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs);

  % Outputs %

  case 3,
    sys=mdlOutputs(t,x,u,lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs);

  case {1,4,9},
    sys=[];

  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

  End of dsfunc.

%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-
function.
```

```
function
[sys,x0,str,ts]=mdlInitializeSizes(lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs)


   Call simsizes for a sizes structure, fill it in and convert it to a
   sizes array.

   Note that in this example, the values are hard coded.  This is not a
   recommended practice as the characteristics of the block are typically
   defined by the S-function parameters.

sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 20;
sizes.NumOutputs     = 5;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);


   Initialize the initial conditions

x0  = [0.5/Jchi
       0.5*Fc/Jchi
       0.5*Fv/Jchi
       0.5*Fs/Jchi
       1200
       0
       0
       0
       0
       1200
       0
       0
       0
       0
       1200
       0
       0
       0
       0
       1200];


   str is always an empty matrix

str = [];


   Initialize the array of sample times

ts  = [T 0];

   end mdlInitializeSizes
```

```
function sys=mdlUpdate(t,x,u,lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs)
accest=(u(1)-u(2))/T;
if (abs(accest)>0.01)&(abs(u(1))>0.005)
   P=[x(5:8,1)'
      x(9:12,1)'
      x(13:16,1)'
      x(17:20,1)'];

   phi=[u(3)
        -sign(u(1))
        -u(1)
        -sign(u(1))*exp(-((u(1)/vs)^2))];

   P=(1/lambda)*P-(1/lambda)*(P*phi)/(lambda+phi'*P*phi)*phi'*P;
   accest=(u(1)-u(2))/T;
   alpha=accest-phi'*x(1:4,1);
   sys=[x(1:4,1)+P*phi/(lambda+phi'*P*phi)*alpha
        P(1,1:4)'
        P(2,1:4)'
        P(3,1:4)'
        P(4,1:4)'];

else
   sys=x;
end
```

```
function sys=mdlOutputs(t,x,u,lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs)

if (abs(u(1))>epsilo)
   sys=[x(2:4,1)'/x(1,1)*[sign(u(1));u(1);sign(u(1))*exp(-((u(1)/vs)^2))]
        1/x(1,1)
        max(0,x(2,1)/x(1,1))
        max(0,x(3,1)/x(1,1))
        max(0,x(4,1)/x(1,1))];

  else
     Tc=x(2,1)/x(1,1)+x(3,1)/x(1,1)*abs(u(1))+x(4,1)/x(1,1)*exp(-
(u(1)/vs)^2);

      if (abs(u(3))>Tc)
```

```
        sys=[Tc*sign(u(3))
             1/x(1,1)
             max(0,x(2,1)/x(1,1))
             max(0,x(3,1)/x(1,1))
             max(0,x(4,1)/x(1,1))];
    else
        sys=[u(3)
             1/x(1,1)
             max(0,x(2,1)/x(1,1))
             max(0,x(3,1)/x(1,1))
             max(0,x(4,1)/x(1,1))];
    end
end

end %% Outputs
```

Annexe B: Schémas et programmes de simulation des modèles d'apprentissage hors ligne sur Simulink™ et Matlab®.

```
% Le modèle inverse de l'articulation
% Programme d'apprentissage hors ligne
% Hicham CHAOUI, Novembre 2001

clear all
close all
load anti;      % anti.mat contient la trajectoire de l'apprentissage.

acht      = aTarget(1:55000);
wcht      = wTarget(1:55000);
thetacht  = Target(1:55000);
torquecht = Tm(1:55000);
pm        = [thetacht';wcht';acht'];
t         = torquecht';

net=newfftd(minmax(pm),[0 2],[6 10
1],{'satlin','tansig','purelin'},'trainlm');
net.trainParam.goal=1e-1;
net.trainParam.epochs=50;
net.trainParam.show=10;
net.trainParam.mem_reduc=3;
net.performFcn='SSE';

achv      = aTarget(55001:60001);
wchv      = wTarget(55001:60001);
thetachv  = Target(55001:60001);
torquechv = Tm(55001:60001);
VV.P      = [thetachv';wchv';achv'];
VV.T      = torquechv';
net       = train(net,pm,t,[],[],VV);

% Validation

load variable     % variable.mat contient les variables de l'articulation.
load Trajec2      % Trajec2.mat contient la trajectoire de validation.
pm1=[trajec2(:,2)';trajec2(:,3)';trajec2(:,4)'];
work  = sim(net,pm1);
for var=1:20001
    Tm1(var,1)=(var-1)*(1e-4);
    Tm1(var,2)=work(var);
end
sim('testanti');
figure;
plot(t,trajec2(:,2),t,thetach1);
```

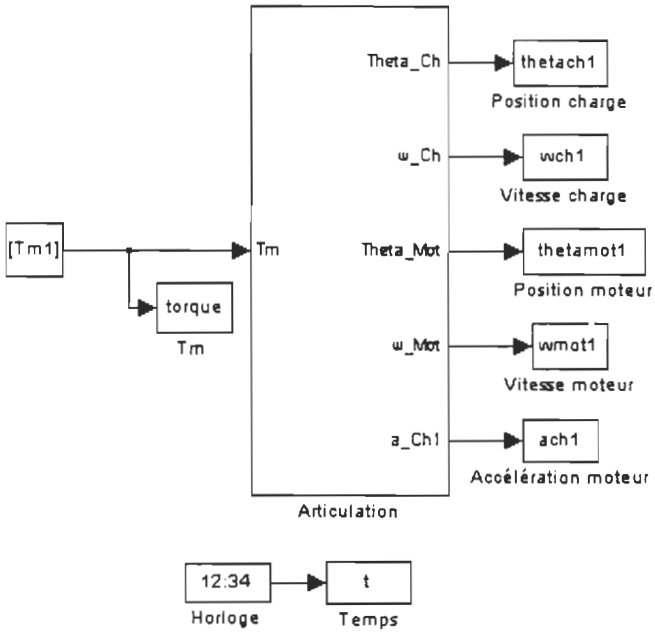

Fig. B.1: Modèle de validation de l'anticipation

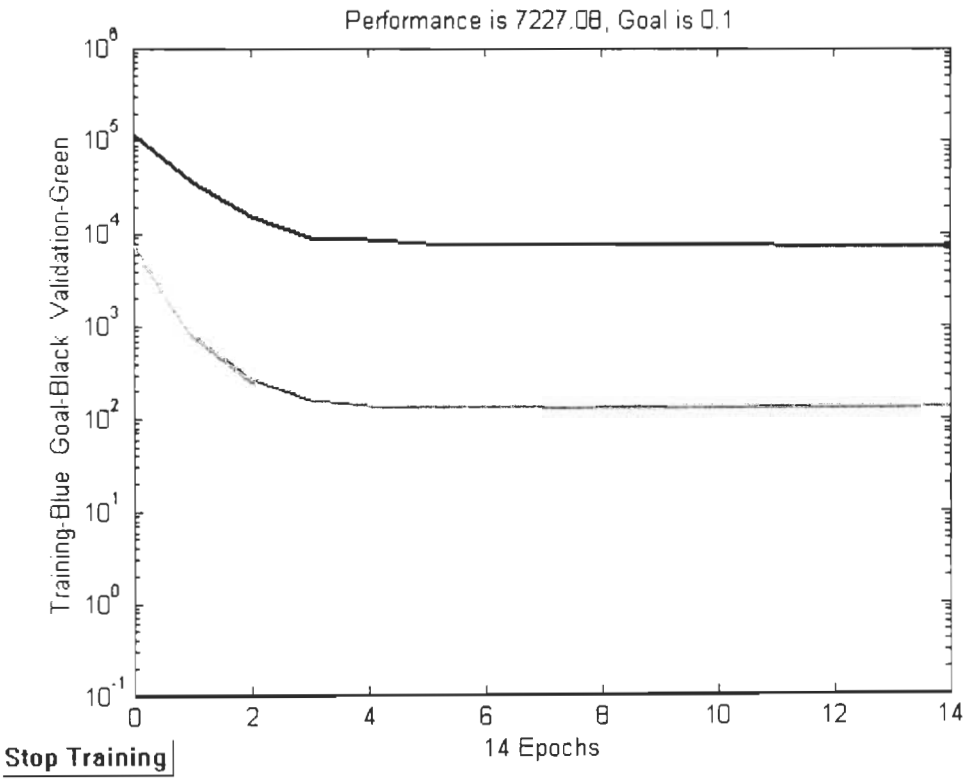

Fig. B.2: Performance d'apprentissage.

```
Le modèle de retroaction de l'articulation

Programme d'apprentissage hors ligne

Hicham CHAOUI, Novembre 2001

clear all
close all
load retro;     % retro.mat contient les variables d'apprentissage.

Tmt       = Deltat2;
deltap1   = Deltap(1:15000);
deltaw1   = Deltaw(1:15000);
Tmt1      = Tmt(1:15000);
pm        = [deltap1';deltaw1';deltaa1'];
t         = Tmt1';

net2=newfftd(minmax(pm),[0 2],[6 10
1],{'tansig','tansig','purelin'},'trainlm');
net2.trainParam.goal=1e-10;
net2.trainParam.epochs=150;
net2.trainParam.show=10;
net2.trainParam.mem_reduc=3;
net2.performFcn='SSE';

deltap2     = Deltap(15001:45001);
deltaw2     = Deltaw(15001:45001);
Tmt2        = Tmt(15001:45001);
VV.P        = [deltap2';deltaw2'];
VV.T        = Tmt2';
net2        = train(net2,pm,t,[],[],VV);

  Validation:

pm1       = [Deltap';Deltaw'];
work      = sim(net2,pm1);
torquech  = work';
figure;
plot(Deltat2);
hold on
plot(torquech,'r');
```

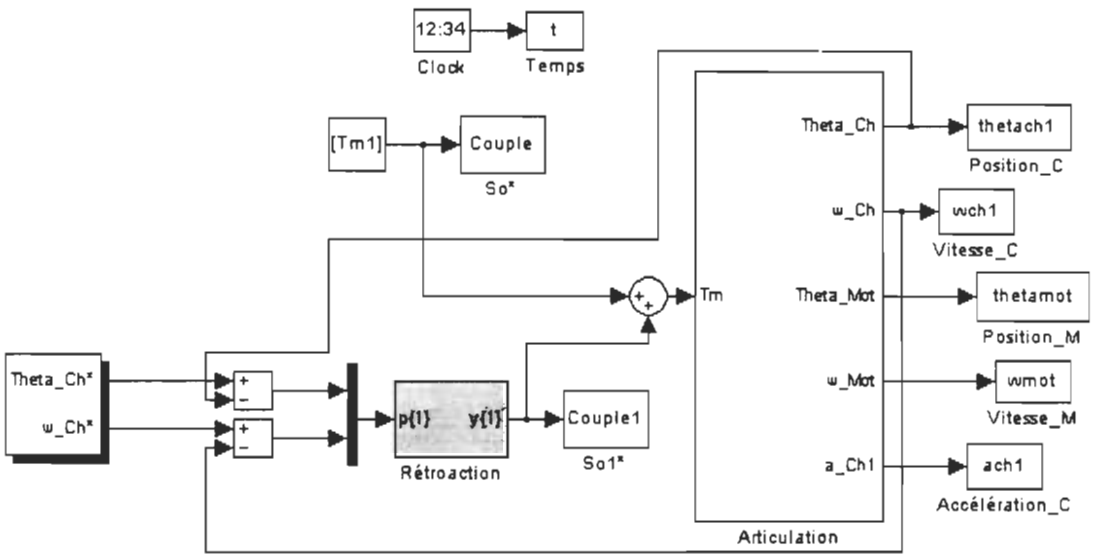

Fig. B.3: Modèle de validation de la rétroaction

Annexe C: Schémas et programmes de simulation des modèles d'apprentissage en ligne sur Simulink™ et Matlab®.

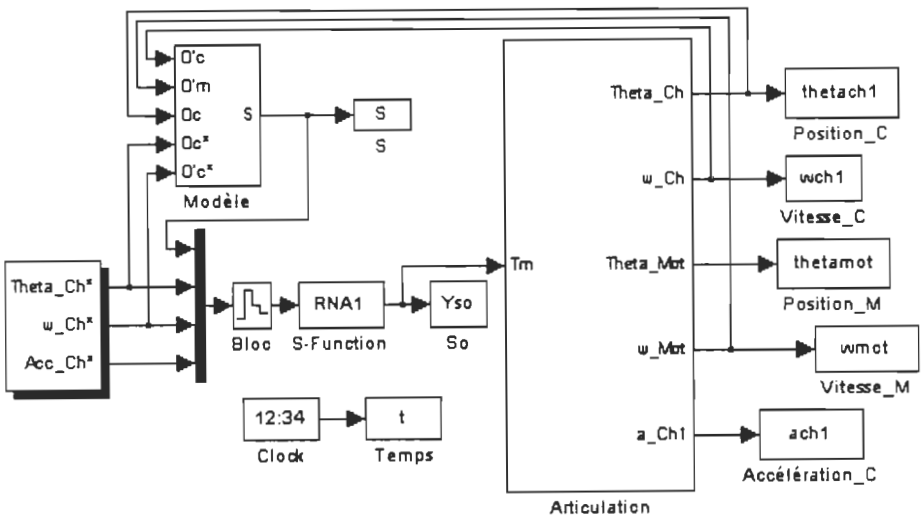

Fig. C.1: Modèle de simulation de l'anticipation avec apprentissage en ligne

```
/*
 * File : RNA1.c
 * Le modèle d'anticipation
 * Programme d'apprentissage en ligne
 * Hicham CHAOUI, Décembre 2001
*/

#define S_FUNCTION_NAME  RNA1
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"

double
iw[3][6],ix[10],iz[6][10],Delta_iw[3][6],Delta_iz[6][10],Delta
_ix[10],b1[6],b2[10],E[3];
int a,b,k,aa,bb,kk;
float A,nuu,eve,kvk,alpha,b3,os;

/*================*
 * Build checking *
 *================*/

/*                  Function:                    mdlInitializeSizes
=============================================
 * Abstract:
 *   Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by
Simulink */
    }

    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 4);
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    if (!ssSetNumOutputPorts(S,1)) return;
    ssSetOutputPortWidth(S, 0, 1);

    ssSetNumSampleTimes(S, 1);

    /* Take care when specifying exception free code - see
sfuntmpl_doc.c */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE |
                 SS_OPTION_USE_TLC_WITH_ACCELERATOR);
```

```
aa    = 2;          % Nombre d'entrée du réseau de neurones.
bb    = 5;          % Nombre de nœuds dans la première couche.
kk    = 9;          % Nombre de nœuds dans la deuxième couche.
A     = 0.7;        % Constante.
eve   = 1e-3;       % Valeur minimale du taux nuu.
kvk   = 1e-1;       % Valeur maximale du taux nuu.
alpha = 1e-4;       % Constante d'apprentissage.
os    = 25;         % Taux de variation du taux nuu.
E[0]  = 0;
E[1]  = 0;
E[2]  = 0;
}

/*              Function:              mdlInitializeSampleTimes
=========================================
 * Abstract:
 *      Specifiy that we inherit our sample time from the
driving block.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}

/*                     Function:                     mdlOutputs
=======================================================
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
InputRealPtrsType entre;
real_T            *y    = ssGetOutputPortRealSignal(S,0);
float Yso,ss,ses,sov,som,Sw,se,see,err,Sz,Sx,yso,yo;
double
Ys[10],Yv[6],thetach,wch,ach,S1,F1[6],F2[10],ro1[6],ro2[10];

entre = uPtrs;
S1    = *entre[0];
thetach = *entre[1];
wch   = *entre[2];
ach   = *entre[3];

nuu = kvk / (1 + exp(-os * (abs(S1) - eve)));

E[0] = thetach;
E[1] = wch;
E[2] = ach;
yso = 0;
for (b=0; b<=bb; b=b+1)
{
```

```
    see = 0;
    for (a=0; a<=aa; a=a+1)
     {
        se  = iw[a][b]*E[a];
        see = see+se;
     }
    Sw = see + b1[b];
    Yv[b] = 2/(1+exp(-2*Sw))-1;
}
for (k=1; k<=kk; k=k+1)
{
    ses = 0;
    for (b=0; b<=bb; b=b+1)
     {
        ss   = iz[b][k]*Yv[b];
        ses  = ses+ss;
     }
    Sz = ses + b2[k];
    Ys[k] = 2/(1+exp(-2*Sz))-1;
    yo    = ix[k]*Ys[k];
    yso   = yso+yo;
}
Sx = yso + b3;
Yso = Sx;

err = S1;
for (k=0; k<=kk; k=k+1)
   {
    F2[k]  = A * Ys[k] * (1-Ys[k]);
    ro2[k] = ix[k]*err*F2[k];
   }
for (b=0; b<=bb; b=b+1)
   {
    som   = 0;
    F1[b] = A * Yv[b] * (1-Yv[b]);
    for (k=1; k<=kk; k=k+1)
     {
        sov = ro2[k]*iz[b][k];
        som = som+sov;
     }
    ro1[b] = F1[b] * som;
}
for (b=0; b<=bb; b=b+1)
    for (a=1; a<=aa; a=a+1)
     {
        Delta_iw[a][b] = alpha * Delta_iw[a][b] + nuu * ro1[b]
* E[a];
        iw[a][b]       = iw[a][b] + Delta_iw[a][b];
     }
for (k=0; k<=kk; k=k+1)
{
```

```
    for (b=1; b<=bb; b=b+1)
     {
       Delta_iz[b][k] = alpha*Delta_iz[b][k]+nuu*ro2[k]*Yv[b];
       iz[b][k]       = iz[b][k] + Delta_iz[b][k];
     }
    Delta_ix[k] = alpha * Delta_ix[k] + nuu * err * Ys[k];
    ix[k]       = ix[k] + Delta_ix[k];
}
y[0] = Yso;
}


/*                    Function:                    mdlTerminate
=============================================================
 * Abstract:
 *     No termination needed, but we are required to have this
routine.
 */
static void mdlTerminate(SimStruct *S)
{
}


#ifdef  MATLAB_MEX_FILE     /* Is this file being compiled as a
MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"          /* Code generation registration
function */
#endif
```

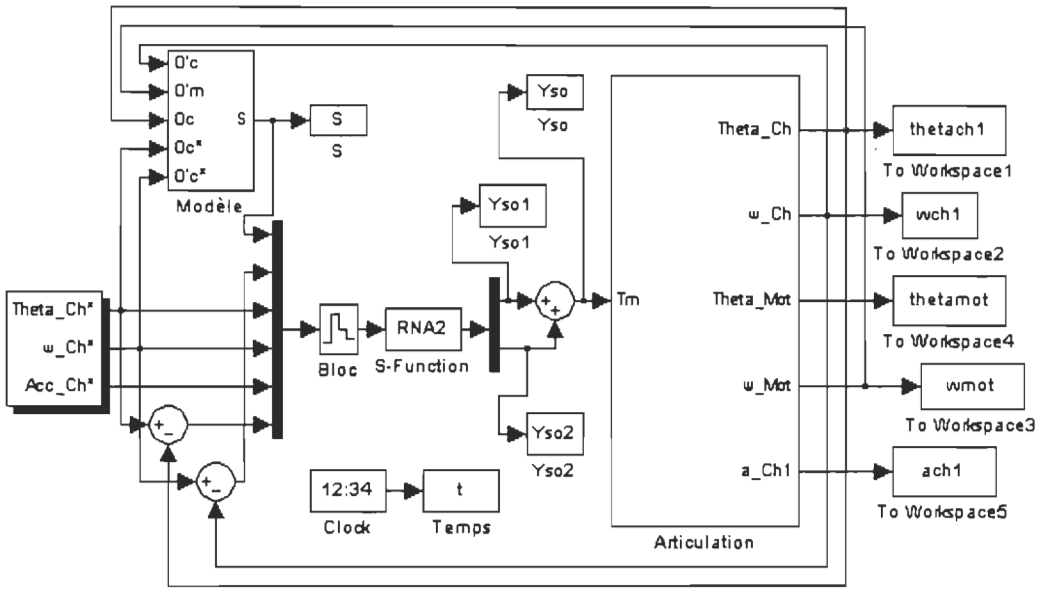

Fig. C.2: Modèle de simulation de l'anticipation et de la rétroaction avec apprentissage en ligne

```
/*
 * File : RNA2.c
 * Le modèle d'anticipation et de rétroaction
 * Programme d'apprentissage en ligne
 * Hicham CHAOUI, Decembre 2001
 */

#define S_FUNCTION_NAME  RNA2
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"

double
iw[3][6],ix[10],iz[6][10],Delta_iw[3][6],Delta_iz[6][10],Delta
_ix[10],b1[6],b2[10],E[3];
int a,b,k,aa,bb,kk,aaa,bbb,kkk;
float A,nuu,eve,kvk,alpha,b3,os,os2;
double
iw2[3][6],ix2[10],iz2[6][10],Delta_iw2[3][6],Delta_iz2[6][10],
Delta_ix2[10],b12[6],b22[10],E2[1];
float nuu2,eve2,kvk2,alpha2,b32;

/*================*
 * Build checking *
 *================*/

/*                  Function:                    mdlInitializeSizes
=================================================
 * Abstract:
 *   Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by
Simulink */
    }
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 6);
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    if (!ssSetNumOutputPorts(S,1)) return;
    ssSetOutputPortWidth(S, 0, 2);

    ssSetNumSampleTimes(S, 1);

    /* Take care when specifying exception free code - see
sfuntmpl_doc.c */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE |
```

```
                    SS_OPTION_USE_TLC_WITH_ACCELERATOR);

aa      = 2;     % Nombre d'entrée du réseau de neurones RNA1.
bb      = 5;     % Nombre de nœuds dans la première couche RNA1.
kk      = 9;     % Nombre de nœuds dans la deuxième couche RNA1.
aaa     = 1;     % Nombre d'entrée du réseau de neurones RNA2.
bbb     = 5;     % Nombre de nœuds dans la première couche RNA2.
kkk     = 9;     % Nombre de nœuds dans la deuxième couche RNA2.
A       = 0.7;   % Constante.
eve     = 1.2;   % Valeur minimale du taux nuu.
eve2    = 0.3;   % Valeur minimale du taux nuu2.
kvk     = 5e3;   % Valeur maximale du taux nuu.
kvk2    = 0.3;   % Valeur maximale du taux nuu2.
os      = 2.3;   % Taux de variation du taux nuu.
os2     = 1;     % Taux de variation du taux nuu2.
alpha   = 1e-4; % Constante d'apprentissage.
alpha2  = 1e-4; % Constante d'apprentissage.
}
/*                Function:              mdlInitializeSampleTimes
=========================================
 * Abstract:
 *       Specifiy that we inherit our sample time from the
driving block.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
/*                      Function:                       mdlOutputs
=============================================================
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
InputRealPtrsType entre;
real_T            *y    = ssGetOutputPortRealSignal(S,0);
float Yso,ss,ses,sov,som,Sw,se,see,err,Sz,Sx,yso,yo;
double
Ys[10],Yv[6],thetach,wch,ach,Dp,Dw,S1,F1[6],F2[10],ro1[6],ro2[
10];
float Yso2,err2;
double Ys2[10],Yv2[6],F12[6],F22[10],ro12[6],ro22[10];

entre=uPtrs;
S1      = *entre[0];
thetach = *entre[1];
wch     = *entre[2];
ach     = *entre[3];
Dp      = *entre[4];
Dw      = *entre[5];
```

```
nuu2 = kvk2 / (1 + exp(-os2 * (abs(S1) - eve2)));
E2[0] = Dp;
E2[1] = Dw;
yso = 0;
for (b=0; b<=bbb; b=b+1)
{
    see = 0;
    for (a=0; a<=aaa; a=a+1)
     {
        se  = iw2[a][b]*E2[a];
        see = see+se;
     }
    Sw = see + b12[b];
    Yv2[b] = 2/(1+exp(-2*Sw))-1;
}
for (k=1; k<=kkk; k=k+1)
{
    ses = 0;
    for (b=0; b<=bbb; b=b+1)
     {
        ss   = iz2[b][k]*Yv2[b];
        ses  = ses+ss;
     }
    Sz    = ses + b22[k];
    Ys2[k] = 2/(1+exp(-2*Sz))-1;
    yo    = ix2[k]*Ys2[k];
    yso   = yso+yo;
}
Sx  = yso + b32;
Yso2 = Sx;

err2 = S1;
for (k=0; k<=kkk; k=k+1)
   {
    F22[k]  = A * Ys2[k] * (1-Ys2[k]);
    ro22[k] = ix2[k]*err2*F22[k];
   }
for (b=0; b<=bbb; b=b+1)
   {
    som   = 0;
    F12[b] = A * Yv2[b] * (1-Yv2[b]);
    for (k=1; k<=kk; k=k+1)
     {
        sov = ro22[k]*iz2[b][k];
        som = som+sov;
     }
    ro12[b] = F12[b] * som;
}
for (b=0; b<=bbb; b=b+1)
    for (a=1; a<=aaa; a=a+1)
     {
```

```
        Delta_iw2[a][b] = alpha2 * Delta_iw2[a][b] + nuu2 *
ro12[b] * E2[a];
        iw2[a][b]       = iw2[a][b] + Delta_iw2[a][b];
     }
for (k=0; k<=kkk; k=k+1)
{
    for (b=1; b<=bbb; b=b+1)
     {
        Delta_iz2[b][k] = alpha2 * Delta_iz2[b][k] + nuu2 *
ro22[k] * Yv2[b];
        iz2[b][k]       = iz2[b][k] + Delta_iz2[b][k];
     }
    Delta_ix2[k] = alpha2 * Delta_ix2[k] + nuu2 * err2 *
Ys2[k];
    ix2[k]       = ix2[k] + Delta_ix2[k];
}

nuu = kvk / (1 + exp(-os * (abs(Yso2) - eve)));

E[0] = thetach;
E[1] = wch;
E[2] = ach;
yso = 0;
for (b=0; b<=bb; b=b+1)
{
    see = 0;
    for (a=0; a<=aa; a=a+1)
     {
        se  = iw[a][b]*E[a];
        see = see+se;
     }
    Sw = see + b1[b];
    Yv[b] = 2/(1+exp(-2*Sw))-1;
}
for (k=1; k<=kk; k=k+1)
{
    ses = 0;
    for (b=0; b<=bb; b=b+1)
     {
        ss   = iz[b][k]*Yv[b];
        ses  = ses+ss;
     }
    Sz = ses + b2[k];
    Ys[k] = 2/(1+exp(-2*Sz))-1;
    yo    = ix[k]*Ys[k];
    yso   = yso+yo;
}
Sx = yso + b3;
Yso = Sx;

err = Yso2;
```

```
for (k=0; k<=kk; k=k+1)
   {
     F2[k]  = A * Ys[k] * (1-Ys[k]);
     ro2[k] = ix[k]*err*F2[k];
   }
for (b=0; b<=bb; b=b+1)
   {
     som   = 0;
     F1[b] = A * Yv[b] * (1-Yv[b]);
     for (k=1; k<=kk; k=k+1)
      {
         sov = ro2[k]*iz[b][k];
         som = som+sov;
      }
     ro1[b] = F1[b] * som;
}
for (b=0; b<=bb; b=b+1)
     for (a=1; a<=aa; a=a+1)
      {
         Delta_iw[a][b] = alpha * Delta_iw[a][b] + nuu * ro1[b]
* E[a];
         iw[a][b]       = iw[a][b] + Delta_iw[a][b];
      }
for (k=0; k<=kk; k=k+1)
{
     for (b=1; b<=bb; b=b+1)
      {
         Delta_iz[b][k] = alpha * Delta_iz[b][k] + nuu * ro2[k]
* Yv[b];
         iz[b][k]       = iz[b][k] + Delta_iz[b][k];
      }
     Delta_ix[k] = alpha * Delta_ix[k] + nuu * err * Ys[k];
     ix[k]       = ix[k] + Delta_ix[k];
}
y[0] = Yso;
y[1] = Yso2;
}

/*                    Function:                    mdlTerminate
=============================================================
 * Abstract:
 *    No termination needed, but we are required to have this
routine.
 */
static void mdlTerminate(SimStruct *S)
{
}


#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a
MEX-file? */
```

```
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"           /* Code generation registration
function */
#endif
```