Ministère de l'enseignement Supérieure et de la Recherche Scientifique

Ecole Nationale Polytechnique
Laboratoire de Commande des Processus

THESE

En vue de l'obtention du titre de
Docteur en Sciences

en Automatique

Présenté par :

Hamid BOUBERTAKH
Maître Assistant Classe A à l'université de Jijel
Magister en Contrôle et Commande de l'EMP

Titre :

Contribution à l'Optimisation par Algorithmes Evolutionnaires des Contrôleurs Flous

Membres du jury :

M. S. BOUCHERIT	Professeur à l'ENP	Président
M. TADJINE	Professeur à l'ENP	Directeur de thèse
S. LABIOD	M. C. à l'université de Jijel	Co-directeur de thèse
K. BENMANSOUR	M. C. à l'université de Média	Examinateur
P.Y. GLORENNEC	Professeur à l'INSA de Rennes	Examinateur
K. HARICHE	Professeur à l'université de Boumerdes	Examinateur
B. HEMICI	M. C. à l'ENP	Examinateur
H. REZINE	Docteur-Ingénieur, MDN	Invité

Année 2009

Avant-propos

Les travaux de cette thèse ont été effectués conjointement dans le laboratoire de Commande des Processus (LCP) de l'Ecole Nationale Supérieure Polytechnique d'Alger (ENSP, ex. ENP), Le laboratoire LAMEL de l'université de Jijel, et le Laboratoire d'Informatique de l'INSA/IRISA de Rennes.

Je remercie mon directeur de thèse Monsieur Mohamed Tadjine et mon co-directeur de thèse Monsieur Salim Labiod pour leurs conseils et leurs soutiens tout au long de la réalisation de ce travail.

Je remercie les membres du jury, Messieurs Mohamed Seghir Boucherit, Pierre-Yves Glorennec, Kamel Harriche et Boualem Hemici , Khelifa Benmansour et Hacene Rezine pour avoir accepté d'évaluer mon travail.

Un grand merci au ministère de l'enseignement supérieure algérien pour m'avoir accordé une bouse de formation à l'étranger (PNE) pour finaliser les travaux de cette thèse.

Un grand merci au Professeur Pierre-Yves Glorennec pour son accueil au Laboratoire d'Informatique à l'INSA/IRISA de rennes ainsi que pour ses conseils et ses orientations durant mon séjour à Rennes. Je remercie aussi à travers lui tous les membres du département d'informatique de l'INSA de Rennes.

Je remercie tous mes collègues du département Automatique de l'université de Jijel ainsi que tous mes amis pour leur aide et leur soutien.

Enfin, une reconnaissance et des remerciements particuliers à toute ma famille pour leur soutien permanant le long de mes études.

Table des matières

Chapitre 2
Apprentissage par renforcement

Chapitre 3
Optimisation par colonies de fourmis

Chapitre 4
Contrôleurs PID flous typiques

Chapitre 5
Optimisation des contrôleurs flous en utilisant l'apprentissage par renforcement

Chapitre 6
Optimisation des contrôleurs flous par colonies de fourmis

Liste des figures

Liste des tableaux

Liste des algorithmes

Introduction générale

Les contrôleurs Proportionnel-Intégral-Dérivé (PID) sont largement utilisés dans la commande des processus industriels où plus de 90% des contrôleurs utilisés sont des PID [AST95]. Cette popularité est due à leur structure simple très familière aux ingénieurs et aux opérateurs. Ils ne sont pas couteux et sont raisonnablement suffisants pour plusieurs systèmes de contrôle industriels. Cependant, les contrôleurs PID classiques ne peuvent pas fournir de bonnes performances dans les cas suivants :

(i) système à commander trop complexe avec des dynamiques et des non linéarités complexes.

(ii) mauvais réglage des paramètres du contrôleur.

Dans ces dernières années, la commande floue [JAN07, YAG94, BUH94, PED93] est présentée comme une alternative potentielle pour combler aux insuffisances des commandes classiques. Le nombre de travaux de recherche sur le développement des contrôleurs flous ainsi que le nombre d'applications industrielles de la commande floue a augmenté exponentiellement dans les trois dernières décennies. En 1974, Mamdani [MAM74] est le premier à étudier la possibilité de commander un système dynamique par des règles d'inférence floue dont les principes de base ont été initialement introduits par Zadeh [ZAD65] en 1965. Puis un an après, Mamdani et Assilian [MAM75, MAM77] développaient le premier contrôleur par logique floue, et l'implémentaient pour la commande d'une turbine à vapeur. Les premières applications industrielles et commerciales ont vu le jour au Japon presque dix ans après [SUG84, KIS85, TOG86,

TAM86]. Depuis, plusieurs entreprises ont fabriqué des produits destinés aux consommateurs utilisant la technologie de la commande floue.

Le premier contrôleur flou développé dans [MAM75] est équivalent à un contrôleur PI flou dont les entrées sont l'erreur entre la consigne et la sortie du système à commander, et la variation de cette erreur. Les premiers travaux de Mamdani introduisaient la méthode de raisonnement flou, nommée le raisonnement min-max de Zadeh-Mamdani. En 1985, Takagi et Sugeno [SUG85] introduisaient une description linguistique différente des ensembles flous de la sortie. Ces travaux ont donné naissance à deux grandes classes de contrôleurs flous ; les contrôleurs flous de Mamdani, et les contrôleurs flous de Takagi-Sugeno (TS). Ils diffèrent essentiellement dans les conclusions des règles floues : le premier utilise des sous ensembles flous et le second utilise des fonctions scalaires. Pour des raisons historiques, le contrôleur flou de Mamdani a reçu plus d'intérêt et l'étude de sa structure analytique a révélé une procédure complexe par rapport aux contrôleurs classiques. La majorité des contrôleurs flous de Mamdani sont non linéaires. En particulier, les contrôleurs flous de type PID sont des contrôleurs PID conventionnels avec des gains variables [MOH02, MOH04, ZHA06].

De plus en plus, ces dernières années, les contrôleurs flous de TS sont utilisés par les chercheurs comme l'atteste le grand nombre de publications sur le sujet. En particulier, l'étude des structures analytiques de ce type de contrôleurs a été une problématique de plusieurs publications [DIN03, JAN07, MOH02, MOO95, YIN98]. Les structures analytiques des contrôleurs flous de TS de type PID et leurs comparaisons avec leurs homologues classiques ont été étudiées par plusieurs chercheurs. Ces études ont révélé que selon le choix de la méthode de conception, les PID flous de type TS correspondent respectivement soit à des contrôleurs PID non linéaires à gains variables [YIN98], soit à des contrôleurs PID linéaires [JAN07].

L'inconvénient majeur des contrôleurs flous est le nombre important de paramètres à régler. En revanche, les contrôleurs classiques ont au maximum trois paramètres qui peuvent être réglés par tâtonnement ou par des méthodes de réglage classiques telles que les méthodes de Ziegler-Nichols [AST95]. On trouve dans la littérature plusieurs méthodes de réglage des contrôleurs flous, à savoir : Ziegler-Nichols [JAN07], la supervision floue [ZHA06], méthodes adaptatives [BHA04], algorithmes génétiques [ADA95, KO06, TAN01a, TAN01b, WU07]. Ces méthodes sont capables de générer une solution optimale ou quasi-optimale.

Les travaux présentés dans cette thèse se situent dans le cadre des objectifs suivants :

- Etude et conception de contrôleurs PID flous typiques qui peuvent combiner les avantages des contrôleurs PID classiques et ceux des contrôleurs flous à savoir : la simplicité de leurs structures analytiques, l'interprétabilité de leurs paramètres, la faculté à introduire des connaissances expertes, et la possibilité de transposer leur loi de commande au cas non linéaire. Ceci permet d'aider au réglage de ces contrôleurs.
- La proposition de méthodes d'optimisation des contrôleurs flous en utilisant des approches stochastiques; les algorithmes d'apprentissage par renforcement [WAT89, SUT98, KAE96], et l'optimisation par colonies de fourmis [DOR06]. Cette partie du travail est motivée par le grand succès connu les dernières années par ces algorithmes pour divers problèmes d'optimisation [CLE02, DOR97, MEN02, SIM03].

Cette thèse est organisée de la manière suivante :

Dans les trois premiers chapitres, nous présentons les trois formalismes utilisés dans notre travail de recherche; les concepts de la commande floue, de l'apprentissage par renforcement et de l'optimisation par colonies de fourmis.

Dans le premier chapitre, consacré à la commande floue, nous commençons par énoncer les fondements théoriques des sous-ensembles flous et de la logique floue. Ensuite, nous donnons la structure générale d'un contrôleur flou et nous décrivons en détail touts ces composants. Et on termine le chapitre par une brève description des contrôleurs flous de type PID et les paramètres de réglage des contrôleurs flous.

Dans le deuxième chapitre, nous présentons le principe d'apprentissage par renforcement. Nous commençons par la présentation des fondements mathématiques de l'apprentissage par renforcement. Ensuite, nous décrivons les principaux algorithmes.

Dans le troisième chapitre, nous présentons le concept de l'optimisation par colonies de fourmis. Au début, nous décrivons le modèle biologique de la méthode et nous présentons quelques expériences qui ont été menées sur des fourmis réelles pour comprendre leur comportement à la recherche de la nourriture. Ensuite, nous présentons le modèle mathématique des fourmis artificielles. Enfin nous décrivons les principaux algorithmes standards existant dans la littérature.

Les trois derniers chapitres présentent les contributions de notre travail.

Le quatrième chapitre est consacré à l'étude analytique des lois de commande d'une classe typique de contrôleurs PID flous. Tout d'abord, un état de l'art est fait sur les différentes méthodes de synthèse d'un contrôleur PID flou. Ensuite, la description d'une méthode particulière de synthèse des contrôleurs PID flous est discutée. Puis, nous faisons

le calcul analytique des lois de commande de ces contrôleurs flous typiques et nous comparons leurs structures analytiques avec lois de commandes PID classiques. Enfin nous adaptons la méthode de réglage de la réponse en fréquence de Ziegler-Nichols à ces contrôleurs flous typiques.

Dans le cinquième chapitre, nous décrivons une méthode permettant de régler les paramètres des contrôleurs flous grâce à l'apprentissage par renforcement. Cette méthode est utilisée principalement pour le réglage des paramètres des contrôleurs PID flous linéaires ou non linéaires dont la méthode de conception est présentée au quatrième chapitre. Mais elle peut être appliquée à tout type de contrôleurs flous.

Dans le dernier chapitre, nous développons une méthode d'optimisation par colonies de fourmis qui permet le réglage des paramètres des contrôleurs flous, en particulier, les contrôleurs flous de type PID.

Le long des deux derniers chapitres, des exemples de simulations sont fournis afin de montrer les performances des méthodes proposées.

Chapitre 1

Théorie de la commande floue

1.1 Introduction

Le nombre de travaux de recherche sur le développement des contrôleurs par logique floue ainsi que le nombre d'applications industrielles de la commande floue a augmenté exponentiellement dans les trois dernières décennies.

Les bases théoriques de la logique floue ont été réellement introduites en 1965 par le professeur Lotfi A. Zadeh de l'université de Berkeley en Californie [ZAD65]. Il a introduit le concept des sous-ensembles flous pour approcher le raisonnement humain à l'aide d'une représentation adéquate des connaissances imprécises, incertaines ou vagues.

En 1974, Mamdani [MAM74] était le premier à étudier la possibilité de commander un système dynamique par des règles d'inférence floue. Puis un an après, Mamdani et Assilian [MAM75] développaient le premier contrôleur par logique floue, et l'implémentaient pour la commande d'une turbine à vapeur. Les premières applications industrielles et commerciales ont vu le jour au Japon presque dix ans après [SUG84, KIS85, TOG86, TAM86]. Depuis, plusieurs entreprises ont fabriqué des produits destinés au grand public utilisant la technologie de la commande floue.

Le premier contrôleur flou développé dans [MAM75] est équivalent à un contrôleur PI flou dont les entrées sont l'erreur entre la consigne et la sortie du système à commander, et

la variation de cette erreur. Les premiers travaux de Mamdani introduisaient la méthode de raisonnement flou, nommée le raisonnement min-max de Zadeh-Mamdani. En 1985, Takagi et Sugeno [SUG85] introduisaient une description différente des sous-ensembles flous de la sortie. Ces travaux ont donné naissance à deux grandes classes de contrôleurs flous : les contrôleurs flous de Mamdani et les contrôleurs flous de Takagi-Sugeno (TS). Ils diffèrent essentiellement dans les conclusions des règles floues : le premier utilise des sous-ensembles flous et le second utilise des fonctions scalaires.

Dans ce chapitre, nous exposons en bref les fondements mathématiques de la théorie des sous-ensembles flous et de la logique floue, puis nous présentons la structure d'un contrôleur flou.

1.2 Sous-ensembles flous et logique floue

La logique floue peut être vue comme une extension de la logique booléenne ; de plus, elle permet de traiter des variables linguistiques dont les valeurs sont des mots ou expressions du langage naturel. Cette partie fournit une introduction générale et un résumé des concepts de base essentiels pour l'étude de la logique floue [BUH94, PED93, YAG94,].

1.2.1 Concept de base d'ensemble flou

Un ensemble classique (figure 1.1 (a)) A de U est défini par une fonction caractéristique notée $\mu_A(x)$ telle que :

$$\mu_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases} \quad (1.1)$$

Le concept de base de la théorie d'ensembles flous est la notion de sous-ensemble flou. Cette notion provient du constat que « très souvent, les classes d'objets rencontrés dans le monde physique ne possèdent pas de critères d'appartenance bien définis ».

Soit U une collection d'objets (par exemple $U = R^N$) appelée univers de discours. Un ensemble flou A dans U est caractérisé par une fonction d'appartenance, notée :

$\mu_A(x)$ ($\mu_A: U \rightarrow [0,1]$), qui appliquée à un élément x de U, retourne un degré d'appartenance $\mu_A(x)$ de x à A (figure 1.1 (b)). Un ensemble flou peut être considéré comme une généralisation de l'ensemble classique. La fonction d'appartenance d'un ensemble classique peut prendre seulement deux valeurs $\{0,1\}$. Un ensemble flou peut être représenté comme un ensemble de paires ordonnées :

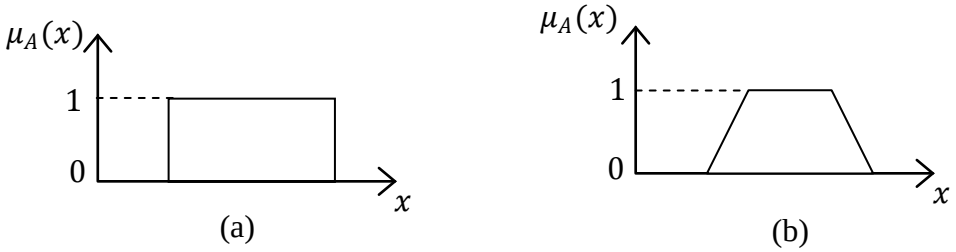


Figure 1.1 : Exemples des fonctions d'appartenance.
(a) Logique classique. (b) Logique floue

$$A = \{(x, \mu_A(x))/x \in U\} \quad (1.2)$$

Si U est discret, A est représenté par :

$$A = \sum_{x_i} \mu_A(x_i)/x_i \quad (1.3)$$

Si U est continu, A est représenté par :

$$A = \int_x \mu_A(x_i)/x_i \quad (1.4)$$

Les ensembles flous ont le grand avantage de constituer une représentation mathématique des termes linguistiques largement utilisés dans l'expression de connaissances expertes, qualitatives et qui sont manipulées par la logique floue.

1.2.2 Variable linguistique

C'est une variable dont les valeurs ne sont pas des nombres, mais des mots ou phrases exprimés en langage naturel. La raison pour laquelle on utilise cette représentation, est que le caractère linguistique est moins spécifique que le caractère numérique.

Une variable linguistique est généralement représentée par un triplet $(x, T(x), U)$dans lequel :

x est le nom de la variable linguistique (vitesse, erreur, position, chaleur, …..) ;

$T(x)$ est l'ensemble des valeurs linguistiques qui sont utilisées pour caractériser *x* ;

U est l'univers de discours de la variable linguistique *x* ;

Par exemple, si l'erreur est considérée comme une variable linguistique définie sur l'univers de discours $U = [-1, +1]$, ses valeurs linguistiques peuvent être définies comme suit :

T(Erreur)= {Négatif Grand (NG), Négatif Petit (NP), Environ Zéro (EZ), Positif Petit (PP), Positif Grand (PG)}.

Ces symboles linguistiques peuvent être considérés comme des ensembles flous dont les fonctions d'appartenance sont représentées sur la figure 1.2.

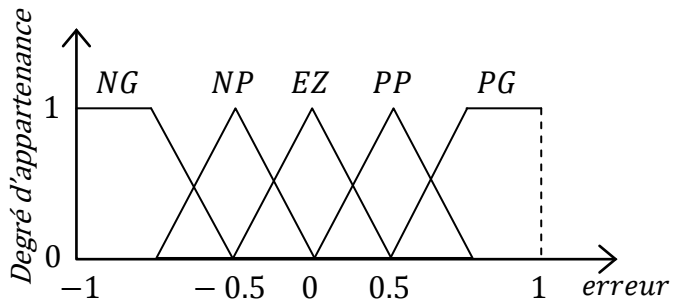


Figure 1.2 : Représentation de la variable linguistique (erreur)

1.2.3 Fonctions d'appartenance

Afin de permettre un traitement numérique des variables linguistiques dans la prise de décision dans le calculateur, une définition des variables linguistiques à l'aide des fonctions d'appartenance s'impose. Dans ce contexte, on associe à chaque valeur de la variable linguistique une fonction d'appartenance désignée par $\mu_A(x)$ où x est la variable linguistique, tandis que A indique l'ensemble concerné. Une valeur précise de $\mu_A(x)$ sera désignée par le degré ou le facteur d'appartenance. Le plus souvent, on utilise pour les fonctions d'appartenance les fonctions suivantes (figure 1.3) **:**

1.2.3.1 Fonction triangulaire

Elle est définie par trois paramètres $\{a, b, c\}$ qui déterminent les coordonnées des trois sommets (figure 1.3 (a)).

$$\mu_A(x) = max\left\{min\left\{\frac{x-a}{b-a}, \frac{c-x}{c-b}\right\}, 0\right\} \tag{1.5}$$

1.2.3.2 Fonction trapézoïdale

Elle est définie par quatre paramètres {a, b, c, d} (figure 1.3 (b)):

$$\mu_A(x) = max\left\{min\left\{\frac{x-a}{b-a}, 1, \frac{d-x}{d-b}\right\}, 0\right\} \tag{1.6}$$

1.2.3.3 Fonction gaussienne

Elle est définie par deux paramètres {σ, m} (figure 1.3(c)):

$$\mu_A(x) = exp\left\{-\frac{(x-m)^2}{2\sigma^2}\right\} \tag{1.7}$$

1.2.3.4 Fonction sigmoïde

Une fonction sigmoïde est définie par deux paramètres {a, c} (figure 1.3 (d)):

$$\mu_A(x) = \frac{1}{1+exp\{-a(x-c)\}} \tag{1.8}$$

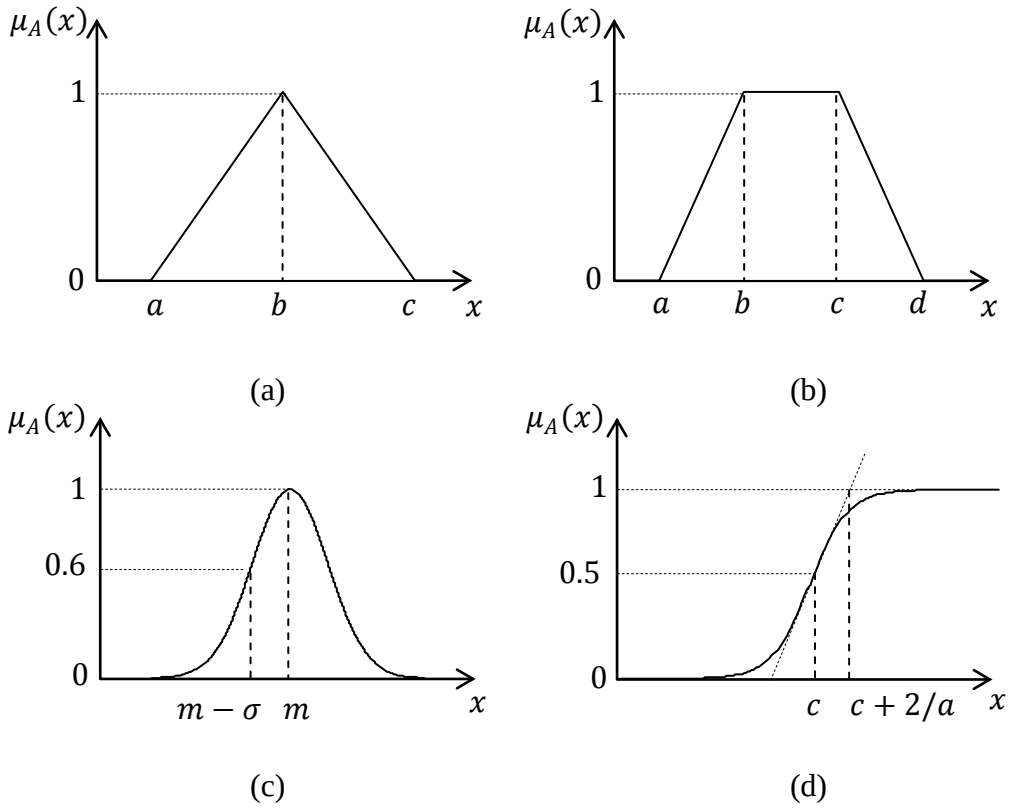


Figure 1.3 : Formes usuelles des fonctions d'appartenance

1.2.4 Caractéristiques d'un ensemble flou

Les caractéristiques d'un ensemble flou A de U les plus utiles pour le décrire sont celles qui montrent à quel point il diffère d'un ensemble classique de U :

1.2.4.1 Support

Le support A noté $supp(A)$, est la partie de U sur laquelle la fonction d'appartenance de A n'est pas nulle :

$$supp(A) = \{x \in U / \mu_A(x) \neq 0\} \tag{1.9}$$

1.2.4.2 Hauteur

La hauteur, notée $h(A)$, d'un ensemble flou A de U est la plus grande valeur prise par sa fonction d'appartenance :

$$h(A) = \max_{x \in U} \mu_A(x) \tag{1.10}$$

1.2.4.3 Noyau

Le noyau de A, noté $noy(A)$, est l'ensemble des éléments de U pour lesquels la fonction d'appartenance de A vaut 1 :

$$noy(A) = \{x \in U / \mu_A(x) = 1\} \tag{1.11}$$

1.2.4.4 Cardinalité

La cardinalité d'un ensemble flou A de U est définie par :

$$|A| = \sum_{x \in U} \mu_A(x) \tag{1.12}$$

1.2.5 Les opérateurs de la logique floue

Les principaux opérateurs dans la théorie d'ensembles classiques sont : le complément, l'intersection et l'union.

Pour deux ensembles classiques A et B d'un univers U, nous avons :

Le complément de A, noté $\overline{A}$:

$$\overline{A} = \{x / x \notin A\} \tag{1.13}$$

L'intersection de A et B, noté $A \cap B$:

$$A \cap B = \{x / x \in A \wedge x \in B\} \tag{1.14}$$

L'union de A et B, noté $A \cup B$:

$$A \cup B = \{x / x \in A \vee x \in B\} \tag{1.15}$$

Les opérateurs de la logique floue sont définis comme suit :

1.2.5.1 Complément flou

Soit l'ensemble flou A d'un univers de discours U, de fonction d'appartenance μ_A. Le complément de A est un ensemble flou dans U, noté $\overline{A}$, donné par :

$$\overline{A} = \left\{\left(x, \mu_{\overline{A}}\right) / x \in U\right\} \tag{1.16}$$

où la fonction d'appartenance de $\overline{A}$ est déterminée pour tout $x \in U$ par :

$$\mu_{\overline{A}} = \approx \mu_A$$

Le symbole ≈ désigne la négation de la fonction d'appartenance. Les opérateurs de complémentation appartiennent à une classe nommée *C-normes*. Le plus simple opérateur est définit comme suit:

$$\forall\, x \in U, u_{\overline{A}}(x) = 1 - u_A(x) \tag{1.17}$$

On trouve dans la littérature l'opérateur $\lambda - complément$ d'un ensemble flou A définit par:

$$u_{\overline{A}}(x) = \frac{1-u_A(x)}{1-\lambda u_A(x)}\ , -1 < \lambda < \infty \tag{1.18}$$

1.2.5.2 Inclusion floue

On considère A et B deux ensembles flous de U, on dit que A est inclu dans B si :

$$\forall\ x \in U, u_A(x) \leq u_B(x) \tag{1.19}$$

Et on note $A \subseteq B$.

1.2.5.3 Intersection floue (conjonction)

Soit deux ensembles flous A et B d'un univers de discours U, de fonctions d'appartenance u_A et u_B respectivement.

L'intersection de A et B, $A \cap B$, est un ensemble flou dans U, sa fonction d'appartenance définie pour tout $x \in U$ par :

$$A \cap B = \{(x, u_{A\cap B}(x))/x \in U\} \tag{1.20}$$

où la fonction d'appartenance $u_{A\cap B}$ est donnée par :

$$u_{A\cap B} = u_A(x)\ \tilde{\wedge}\ u_B(x) \tag{1.21}$$

Le symbole $\tilde{\wedge}$ désigne l'opérateur « ET flou » dans l'ensemble des fonctions d'appartenance.

Pour l'opération d'intersection, Zadeh, en 1965, a proposé l'opérateur min et *le produit algébrique*. Après l'idée de Zadeh, plusieurs chercheurs avaient proposé plusieurs variantes pour cette opération :

Min

$$u_{A\cap B} = min\{u_A(x), u_B(x)\} \tag{1.22}$$

Produit algébrique

$$u_{A\cap B} = u_A(x)u_B(x) \tag{1.23}$$

Différence bornée

$$u_{A\cap B} = max\{0, u_A(x) + u_B(x) - 1\} \tag{1.24}$$

Produit d'Einstein

$$u_{A\cap B} = \frac{u_A(x)u_B(x)}{2-u_A(x)+u_B(x)-u_A(x)u_B(x)} \tag{1.25}$$

Produit d'Hamasher

$$u_{A\cap B} = \frac{u_A(x)u_B(x)}{u_A(x)+u_B(x)-u_A(x)u_B(x)} \quad (1.26)$$

Tous ces opérateurs sont appelés les normes triangulaires ou les T-normes, ces fonctions définissent une classe générale des opérateurs d'intersection d'ensembles flous. Les opérateurs les plus souvent utilisés sont : le min et le $produit\ algébrique$.

1.2.5.4 Union floue (disjonction)

Pour l'union de deux ensembles flous, il existe une classe d'opérateur appelée $T-conormes$ ou $S-normes$. Pour deux ensembles flous A et B dans l'univers de discours U, l'union de A et B, $A \cup B$ est un ensemble flou dans U, défini par :

$$A \cup B = \{(x, u_{A\cup B}(x))/x \in U\} \quad (1.27)$$

où la fonction d'appartenance $u_{A\cap B}$ est donnée par :

$$u_{A\cup B} = u_A(x)\ \tilde{\vee}\ u_B(x) \quad (1.28)$$

Le symbole $\tilde{\vee}$ désigne l'opérateur « OU flou » dans les fonctions d'appartenance.

Les opérateurs d'union cités dans la littérature sont :

Max

$$u_{A\cap B} = max\{u_A(x), u_B(x)\} \quad (1.29)$$

Somme algébrique

$$u_{A\cup B} = u_A(x) + u_B(x) - u_A(x)u_B(x) \quad (1.30)$$

Somme bornée

$$u_{A\cup B} = min\{1, u_A(x) + u_B(x)\} \quad (1.31)$$

Somme d'Einstein

$$u_{A\cup B} = \frac{u_A(x)+u_B(x)}{1+u_A(x)u_B(x)} \quad (1.32)$$

Somme d'Hamasher

$$u_{A\cup B} = \frac{u_A(x)+u_B(x)-2u_A(x)u_B(x)}{1-u_A(x)u_B(x)} \quad (1.33)$$

L'opérateur le plus souvent utilisé est l'opérateur max.

1.2.6 Produit cartésien

Soient $A_1, A_2, \dots, A_n$ des ensembles flous, respectivement dans $U_1, U_2, \dots, U_n$. Le produit cartésien de $A_1, A_2, \dots, A_n$ est un ensemble flou défini sur $U_1 \times U_2 \times \dots \times U_n$ de fonction d'appartenance :

$$\mu_{A_1 \times A_2 \times \ldots \times A_n}(x_1, x_2, \ldots, x_n) = \mu_{A_1}(x_2) \,\tilde{\wedge}\, \mu_{A_2}(x_2) \,\tilde{\wedge}\, \ldots \tilde{\wedge}\, \mu_{A_n}(x_n) \quad (1.34)$$

1.2.7 Relation floue

Une relation floue représente le degré de présence ou d'absence d'une association entre les éléments de deux ou de plusieurs ensembles flous. Une relation floue d'ordre n est un ensemble flou défini sur $U_1 \times U_2 \times \ldots \times U_n$ par l'expression suivante:

$$R_{U_1 \times U_2 \times \ldots \times U_n} = \{((x_1, x_2, \ldots, x_n), U_R(x_1, x_2, \ldots, x_n))/(x_1, x_2, \ldots, x_n) \in U_1 \times U_2 \times \ldots \times U_n\} \quad (1.35)$$

1.2.8 Compositions des relations floues

Soit R_1 et R_2 deux relations floues définies respectivement dans $U \times V$ et $V \times W$. La composition de R_1 et R_2, est un ensemble flou symbolisé par $R_1 \circ R_2$, de fonction d'appartenance :

$$\mu_{R_1 \circ R_2}(x, z) = \left\{(x, z), Sup_{y \in V}\{\mu_{R_1}(x, y) \,\tilde{\wedge}\, \mu_{R_2}(y, z)\}\right\} \quad (1.36)$$

Cette composition est appelée $min - max$.

1.2.9 Implication floue

L'implication floue est un opérateur qui permet d'évaluer le degré de vérité d'une règle R de la forme "$Si\ x\ est\ A\ Alors\ y\ est\ B$" à partir des valeurs de la prémisse d'une part, et de celle de la conclusion d'autre part.

$$\mu_R(x, y) = imp(\mu_A(x), \mu_B(y)) \quad (1.37)$$

Les opérateurs les plus utilisés en commande floue sont les implications de Mamdani et de Larsen :

Implication de Mamdani

$$\mu_R(x, y) = min(\mu_A(x), \mu_B(y)) \quad (1.38)$$

Implication de Larsen

$$\mu_R(x, y) = \mu_A(x)\mu_B(y) \quad (1.39)$$

1.2.10 Raisonnement flou

En logique classique, la méthode d'inférence la plus utilisé est le modus ponens. Elle permet à partir de la règle « $Si\ x\ est\ A\ Alors\ y\ est\ B$ » et du fait « $x\ est\ A$ » de conclure 1e fait "$y\ est\ B$", qui sera ajouté à la base des faits. Zadeh a étendu cette mérthode au cas flou, sous le nom *modus ponens généralisé* pour permettre de raisonner lorsque les règles

ou les faits sont connus de façon imparfaite. Le *modus ponens* et le *modus ponens généralisé* se résument comme suit :

	Modus Ponens	*Modus Ponens généralisé*
Prémisse (fait) :	« x *est* A »	« x *est* $\acute{A}$ »
Implication (règle) :	« *Si* x *est* A *Alors* y *est* B »	« *Si* x *est* A *Alors* y *est* B »
Conséquence (déduction) :	« y *est* B »	«y *est* $\acute{B}$»

A partir de la règle « *Si* A *Alors* B » et du fait $\acute{A}$, on déduit un nouveau fait $\acute{B}$ qui est caractérisé par un sous-ensemble flou dont la fonction d'appartenance est donnée par :

$$\mu_{\acute{B}}(y) = Sup_x\left(\mu_{\acute{A}}(x)\,\tilde{\wedge}\,\mu_R(x,y)\right) \tag{1.40}$$

Les fonctions d'appartenance $\mu_{\acute{A}}$ et μ_R caractérisent respectivement le fait $\acute{A}$ et la règle.

1.3 Commande floue

Un contrôleur flou peut être vu comme un système expert fonctionnant à partir d'une représentation des connaissances basées sur la théorie des ensembles flous [BUH94, PED93, YAG94]. La figure 1.4 montre le schéma synoptique d'un tel contrôleur.

Nous allons rappeler dans ce qui suit une description sommaire de chaque module composant ce contrôleur :

- La base des règles floues, ou base de connaissance du processus : elle est composée de l'ensemble des renseignements que nous possédons sur le processus. Elle permet de définir les fonctions d'appartenance et les règles floues qui décrivent le comportement du système; elle est le cœur du système entier dans le sens où tous les autres composants sont utilisés pour interpréter et combiner ces règles pour former le système final.
- Interface « Numérique→Symbolique » ou fuzzification : ce module traduit les données numériques caractérisant l'état du système. Il fournit une caractérisation floue des variables du système flou sous forme symbolique.
- La logique de prise de décision, ou moteur d'inférence floue: une action, sous forme symbolique, est décidée à l'aide des techniques de raisonnement flou en fonction des variables floues précédemment calculées.
- Interface « Symbolique→Numérique » ou défuzzification : ce module traduit l'action floue issue de l'inférence en une grandeur physique directement applicable au processus à commander. Il s'agit de l'étape de défuzzification.

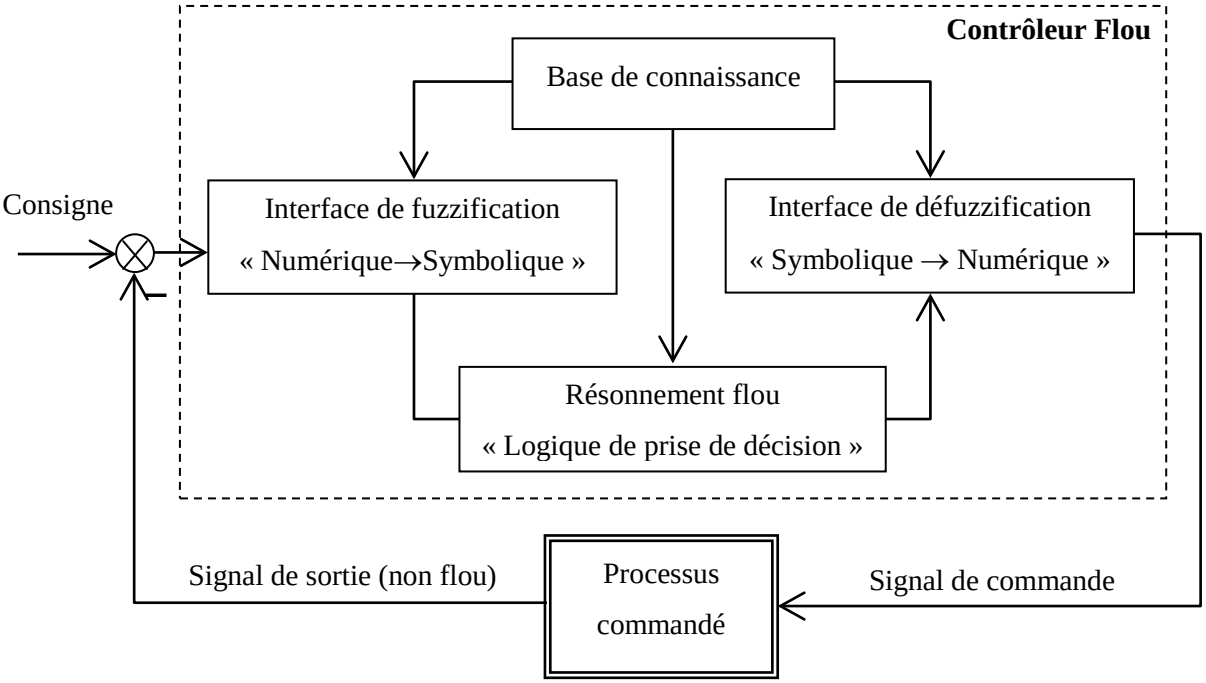


Figure 1.4 : Structure générale d'un contrôleur flou

1.3.1 Fuzzification

La fuzzification consiste à relier le point numérique $x_0 = [x_{01}, x_{02}, \dots, x_{0n}]^T$ de U à l'ensemble flou $A_x = [A_{x_1}, A_{x_2}, \dots, A_{x_n}]^T$ dans $U = U_1 \times U_2 \times \dots \times U_n$ où A_{x_i} est un ensemble flou dans U_i.

Il existe deux méthodes de fuzzification suivant la définition de A_x.

- A_x est un singleton flou défini par :

$$\mu_{A_x}(x) = \begin{cases} 1 & si\ x = x_0 \\ 0 & si\ x \neq x_0 \end{cases} \quad (1.41)$$

Dans ce cas, on considère que la valeur de *x* est précise et certaine (figure 1.5 (a)).

- A_x est un ensemble flou de fonction d'appartenance : $\mu_{A_x}(x_0) = 1$ et $\mu_{A_x}(x)$ décroît lorsque x s'éloigne de x_0.

Dans ce cas, est pris en compte le comportement de la variable autour de la valeur x_0. Par exemple, l'erreur de mesure est modélisée par une fonction d'appartenance triangulaire (figure 1.5 (b)).

Figure 1.5 : Méthodes de fuzzification ; (a) singleton, (b) ensemble flou

1.3.2 Base des règles floues

Une base de règles floues R est une collection de règles floues de la forme «*SI-ALORS*», $R = [R_1\ R_2\ \dots R_M]$. Une règle floue R_i est donnée sous le modèle de « Mamdani » comme suit:

$$R_i: SI\ x_1\ est\ A_{i1} et\ x_2 est\ A_{i2}\ et \dots et\ x_n\ est\ A_{in}\ ALORS\ y\ est\ B_i \tag{1.42}$$

ou sous le modèle de «Takagi-Sugeno (TS)» sous la forme:

$$R_i: SI\ x_1\ est\ A_{i1} et\ x_2 est\ A_{i2}\ et \dots et\ x_n\ est\ A_{in}\ ALORS\ y\ est\ f_i(x) \tag{1.43}$$

avec $f_i(x)$ généralement un polynôme. Si le polynôme est d'ordre zéro on dit que le modèle est de TS d'ordre zéro, et si le polynôme est du premier ordre, on dit que le modèle est de TS d'ordre un.

1.3.3 Moteur d'inférence floue

Le moteur d'inférence floue utilise la base des règles floues pour effecteur une transformation à partir des ensembles flous dans l'espace d'entrée vers les ensembles flous dans l'espace de sortie en se basant sur les opérations de la logique floue. L'antécédent de la règle *R*, définit un produit cartésien de $A_{i1}, A_{i2}, \dots, A_{in}$, et la règle elle-même R_i , est vue comme une implication. Soit $A_{x,}$ un ensemble flou dans U, alors chaque règle R_i détermine un ensemble flou $B_i = A_x \circ R_i$, dans *V*. La fonction d'appartenance de $\acute{B}_i$ est donnée par la règle compositionnelle (1.45) :

$$\mu_{\acute{B}}(y) = sup_{x \in A_x} \left(\mu_A(x)\ \tilde{\wedge}\ \mu_{R_i}(x,y) \right) \tag{1.44}$$

L'ensemble des M règles constituant la base des règles floues sont liées par l'opérateur de disjonction "*OU*". Ainsi, l'ensemble flou final $\acute{B}_i = A_x \circ R_i$, est donné par la relation:

$$\begin{cases} \acute{B} = \acute{B}_1\ \tilde{\vee}\ \acute{B}_2\ \tilde{\vee} \dots \tilde{\vee}\ \acute{B}_M \\ \mu_{\acute{B}}(y) = \mu_{A_x \circ R_1}(y)\ \tilde{\vee}\ \mu_{A_x \circ R_2}\ \tilde{\vee} \dots \tilde{\vee}\ \mu_{A_x \circ R_M} \end{cases} \tag{1.45}$$

Dans le jeu de règles du système flou interviennent les opérateurs flous "ET(AND)" et "OU(OR)". L'opérateur "ET" s'applique aux variables à l'intérieur d'une règle, tandis que l'opérateur "OU" lie les différentes règles. Plusieurs types de raisonnement flou ont été proposés dans la littérature suivant la réalisation des opérateurs flous "ET" et "OU" et le type des règles floues utilisées. Les trois moteurs d'inférence floue les plus utilisés sont : le moteur de Mamdani, celui de Sugeno et celui de Tsukumoto :

1.3.3.1 Méthode de Mamdani

Dans le modèle de Mamdani, les prémisses et les conclusions des règles sont symboliques ou linguistiques. Cette méthode se base sur l'utilisation de l'opérateur min pour l'implication floue et l'opérateur max pour l'agrégation des règles. La sortie nécessite l'utilisation d'une méthode de défuzzification qui est généralement le barycentre. Une autre variante du modèle de Mamdani consiste à remplacer l'opérateur min de l'implication floue par le p$roduit\ alg\acute{e}brique$.

1.3.3.2 Méthode de Sugeno

Elle s'appelle aussi méthode de Takagi-Sugeno. Dans ce cas, des règles floues de type Sugeno sont utilisées. En effet, les conclusions des règles floues sont des polynômes ou plus généralement des fonctions des variables d'entrée. L'implication floue est réalisée par l'opérateur min ou par le $produit\ alg\acute{e}brique$. La sortie finale est égale à la moyenne pondérée des conclusions des règles.

1.3.3.3 Méthode de Tsukumoto

Dans ce cas, des fonctions monotoniques sont associées aux variables de sortie. La sortie totale est une moyenne pondérée des degrés de confiance des règles floues et des valeurs des fonctions des variables de sortie.

La figure 1.6 illustre les types du raisonnement flou pour un système flou à deux entrées et une base de connaissances de deux règles floues. On constate que les différences viennent de la spécification de la partie conclusion d'une part, et de la méthode de défuzzification d'autre part.

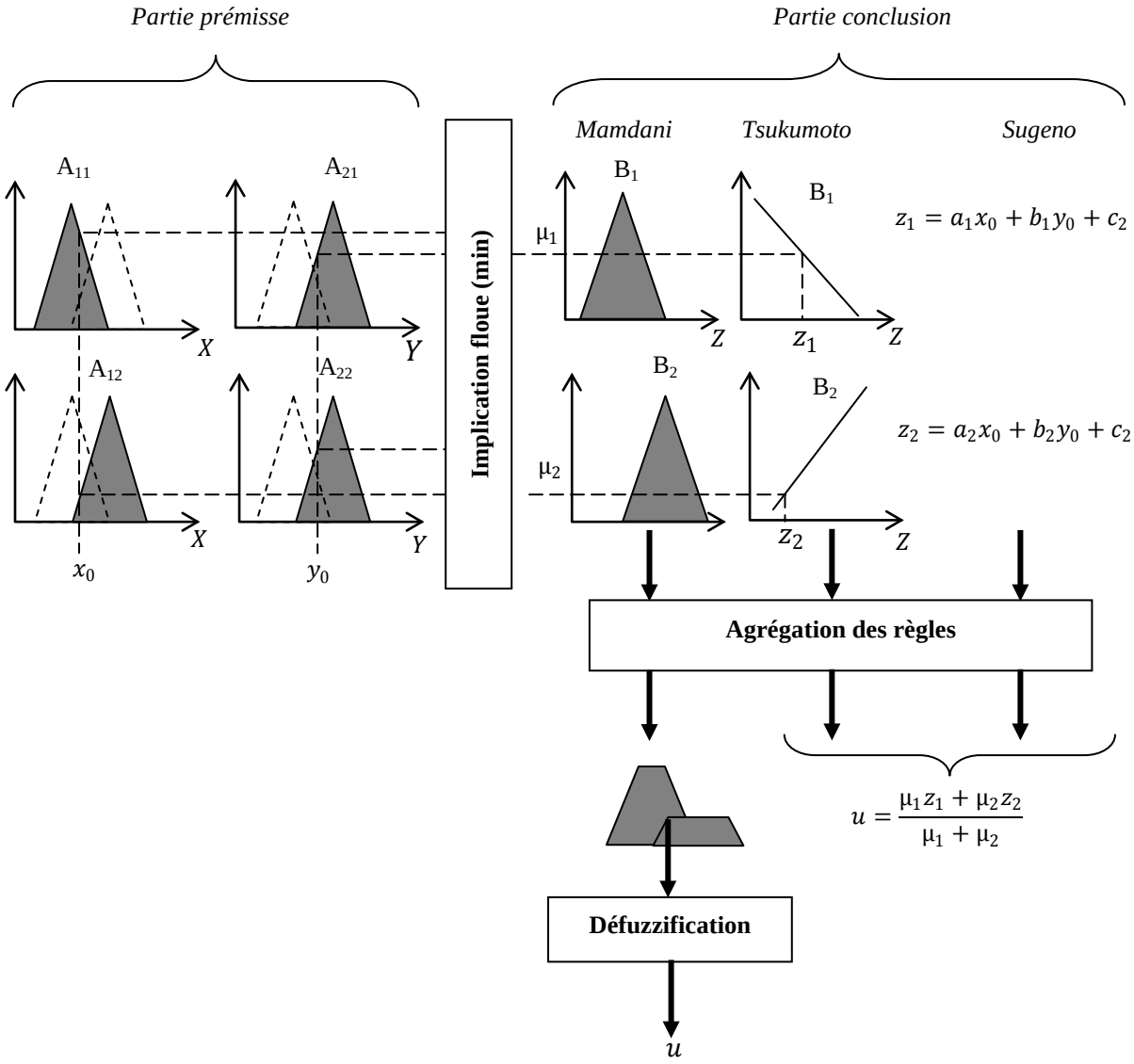


Figure 1.6 : Les différents modèles d'inférence floue

1.3.4 Défuzzification

Le rôle de la défuzzification est de transformer la partie floue issue de l'inférence en une grandeur numérique. Malheureusement, il n'y a pas une procédure systématique pour choisir la stratégie de défuzzification. Comme on s'intéresse à l'application de la logique floue en commande, un critère de choix d'une méthode de défuzzification est la simplicité du calcul. Ce critère a conduit aux méthodes de défuzzification suivantes : méthode du maximum, méthode des maxima, méthode du centre de gravité, méthode des hauteurs pondérées et la méthode des hauteurs pondérées modifiées.

1.3.4.1 Méthode du maximum

La méthode du maximum examine l'ensemble flou $\acute{B}$ issu de l'inférence et choisit comme sortie la valeur y pour laquelle $\mu_{\acute{B}}(y)$ est un maximum. Cependant, ce défuzzificateur présente un certain inconvénient lorsqu'il existe plusieurs valeurs pour lesquelles $\mu_{\acute{B}}(y)$ est un maximum.

1.3.4.2 Méthode de la moyenne des maxima

Cette méthode examine l'ensemble flou $\acute{B}$ issu de l'inférence et détermine en premier temps les valeurs pour lesquelles $\mu_{\acute{B}}(y_i)$ est un maximum. Ensuite, on calcule la moyenne de ces valeurs comme résultat de défuzzification.

1.3.4.3 Méthode du centre de gravité

Dans ce cas, on calcule le centre de gravité $\overline{y}$ de l'ensemble flou résultant $\acute{B}$ et considère cette valeur comme résultat de défuzzification :

$$\overline{y} = \frac{\int_S y\mu_{\acute{B}}(y)dy}{\int_S \mu_{\acute{B}}(y)dy} \tag{1.46}$$

où S représente le support de *B'*. Souvent on utilise la version discrète de l'intégral :

$$\overline{y} = \frac{\sum_{i=1}^{I} y_i\mu_{\acute{B}}(y_i)}{\sum_{i=1}^{I} \mu_{\acute{B}}(y_i)} \tag{1.47}$$

Il est à noter que le centre de gravité est généralement difficile à calculer. De ce fait, cette méthode est la plus coûteuse en temps de calcul.

1.3.4.4 Méthode des hauteurs pondérées

La valeur réelle de sortie est donnée par la relation suivante :

$$y = \frac{\sum_{i=1}^{M} v_i\overline{y_i}}{\sum_{i=1}^{M} v_i} \tag{1.48}$$

où : v_i représente le degré de confiance ou d'activation de la règle R_i et $\overline{y_i}$ le centre de gravité de l'ensemble flou de la variable de sortie associée à la règle R_i.

Cette méthode est très simple à implémenter. En effet, les centres de gravité des fonctions d'appartenance sont connus à priori. Cependant, elle soufre d'un inconvénient du fait qu'elle n'utilise pas la forme entière des fonctions d'appartenance de la conclusion. Elle utilise seulement le centre $\overline{y_i}$ du support de la fonction d'appartenance de la conclusion sans tenir compte du fait que la fonction d'appartenance est étroite ou large.

1.3.4.5 Méthode des hauteurs pondérées modifiée

Contrairement à la méthode précédente, la sortie du système flou est calculée en tenant compte du support de la fonction d'appartenance de la conclusion, par l'expression :

$$y = \frac{\sum_{i=1}^{M} v_i \overline{y_i} / {\sigma_i}^2}{\sum_{i=1}^{M} v_i / {\sigma_i}^2} \tag{1.49}$$

Où σ_i est une mesure de l'étendue de la fonction d'appartenance de la variable de sortie associée à la règle R_i. Pour des fonctions d'appartenance triangulaires ou trapézoïdales, σ_i peut être le support du triangle ou du trapèze, et pour une fonction gaussienne, σ_i peut être sa variance.

1.4 Contrôleurs flous de type PID

Bien que la base de règles soit le noyau d'un contrôleur flou, le choix approprié des variables linguistiques et leurs fonctions d'appartenance reste une étape importante dans la conception d'un contrôleur flou. Théoriquement, le nombre d'entrées d'un régulateur flou n'est pas limité. En pratique, cependant, il n'est pas rationnel d'utiliser plus de trois variables d'entrée puisque la détermination de la base de règles devient trop complexe.

Dans le cas de certains processus d'ordre élevé, on utilisera plutôt une technique qui consiste à décomposer le régulateur en blocs de commande à deux entrées (décentralisation de la commande).

Typiquement, dans un contrôleur flou, les variables linguistiques sont : L'erreur, la variation de cette erreur et la commande. Ce choix peut être vu comme une extension de la version classique des régulateurs PID [YAG94, JAN07].

Les différentes formes de commande par PID flou sont résumées dans les points suivants.

1.4.1 Contrôleur Proportionnel Flou (PF)

La sortie du contrôleur est proportionnel à la valeur de l'erreur $e(k) = y_r(k) - y(k)$ entre la consigne y_r est la sortie du système sous contrôle y. Les règles floues sont de la forme :

$$Si\ e(k)\ est\ E\ Alors\ u(k)\ est\ U$$

La loi de commande est alors de la forme :

$$u(k) = f\big(e(k)\big) \tag{1.50}$$

f est la fonction du système d'inférence floue utilisé.

1.4.2 Contrôleur Proportionnel Intégral Flou (PIF)

Un régulateur PI flou est décrit à l'aide de règles floues qui définissent la relation entre la variation de la commande $\Delta u(k) = u(k) - u(k-1)$ d'une part, et l'erreur $e(k)$ et sa variation $\Delta e(k) = e(k) - e(k-1)$ d'autre part. Les règles floues sont de la forme :

$$Si\ e(k)\ est\ E\ et\ \ \Delta e(k)\ est\ DE\ Alors\ \Delta u(k)\ est\ \Delta U$$

et la loi de commande est donnée par :

$$\Delta u(k) = f(e(k), \Delta e(k)) \tag{1.51}$$

1.4.3 Contrôleur Proportionnel Dérivé Flou (PDF)

Si la sortie du régulateur flou est la commande elle-même, on dit que le régulateur est du type PD flou. Les règles floues sont alors de la forme :

$$Si\ e(k)\ est\ E\ et\ \ \Delta e(k)\ est\ DE\ Alors\ u(k)\ est\ U$$

et la loi de commande est donnée par :

$$u(k) = f(e(k), \Delta e(k)) \tag{1.52}$$

1.4.4 Contrôleur Proportionnel Intégral Dérivé Flou (PIDF)

Il est obtenu par l'ajout d'une entrée supplémentaire, qui est la somme des erreurs $\sum e(k) = \sum_{i=0}^{k} e(k-i)$. Les règles prennent la forme :

$$Si\ e(k)\ est\ E\ et\ \ \Delta e(k)\ est\ DE\ et\ \sum e(k)\ \ est\ SE\ Alors\ u(k)\ est\ U$$

Et la loi de commande devient :

$$u(k) = f(e(k), \Delta e(k), \sum e(k)) \tag{1.53}$$

1.5 Paramètres de réglage d'un contrôleur flou

La conception d'un contrôleur flou se décompose essentiellement en deux phases. La première consiste en l'optimisation structurelle qui passe par la détermination de méthode de fuzzification, le type et le nombre des fonctions d'appartenance, le type et nombre des règles floues, la méthode du raisonnement flou et la stratégie de défuzzification. Cette phase est généralement effectuée empiriquement en choisissant à priori les paramètres cités ci-dessus.

La deuxième phase consiste en l'optimisation paramétrique. Dans cette phase un réglage de paramètres est effectué afin d'obtenir les meilleures performances par le contrôleur flou ; les paramètres à régler sont généralement :

- Le domaine de chaque variable ;
- Les positions modales des fonctions d'appartenance dans le domaine de chaque variable ;
- Les paramètres liés aux fonctions d'appartenance.

Le réglage peut être effectué manuellement ou par des méthodes d'optimisation et d'apprentissage. Ces approches permettent de tenir compte à la fois des connaissances d'un expert humain, de l'incertitude et de l'imprécision des données traitées par le contrôleur.

1.6 Conclusion

Dans ce chapitre, nous avons présenté en bref la théorie de la commande par logique floue. Les notions de base de la logique floue les plus pertinentes en commande floue sont exposées et l'architecture de base d'un contrôleur flou est présentée. Le fonctionnement d'un contrôleur flou dépend d'un nombre important de paramètres (méthode de fuzzification, le type des fonctions d'appartenance, le type des règles floues, la méthode du raisonnement flou et la stratégie de défuzzification) qu'il faut déterminer lors de la conception. Comme ces paramètres s'influencent mutuellement, leur réglage n'est donc pas aisé. Par contre, les contrôleurs flous présentent la possibilité d'incorporer des connaissances expertes dans leurs structures, ce qui peut aider à la recherche des paramètres optimaux des contrôleurs flous.

Dans le chapitre 2 et le chapitre 3 nous exposerons respectivement les principes de l'apprentissage par renforcement et l'optimisation par colonies de fourmis, ces méthodes permettent d'explorer de façon très efficace l'espace des solutions possibles d'un problème d'optimisation. Ensuite, dans les chapitres 5 et 6 nous verrons comment ces algorithmes peuvent être utilisés pour le réglage des paramètres d'un contrôleur flou.

Chapitre 2

Apprentissage par renforcement

2.1 Introduction

L'apprentissage par renforcement [WAT89, SUT98, KAE96] recouvre un ensemble de méthodes d'apprentissage automatique faiblement supervisé. Dans l'apprentissage supervisé, l'agent connaît à tout moment l'erreur qu'il commet : pour chaque vecteur d'entrée, la sortie désirée correspondante est connue. L'agent peut donc utiliser la différence entre sa sortie et la sortie désirée pour corriger ses paramètres. A l'inverse, dans l'apprentissage par renforcement l'état désiré n'est pas connu à l'avance, ceci est très utile quand les détails d'une tâche ou l'environnement d'un agent sont inconnus ou difficiles à assimiler. La qualité d'une action est évaluée par un signal scalaire dit le renforcement.

L'apprentissage par renforcement a pour objectif de maximiser la performance du système en renforçant les meilleures actions. L'agent obtient des informations de l'état de l'environnement et agit également sur l'environnement puis reçoit une estimation de sa performance sous forme de récompense ou de punition. Le modèle standard d'interaction entre l'agent et son environnement est présenté sur la figure 2.1. A chaque pas d'interaction, l'agent perçoit l'état actuel s_t de l'environnement et puis choisit une action a_t. L'agent change l'état selon la dynamique de l'environnement, et la qualité de la transition de l'état est communiquée à l'agent au moyen d'un signal scalaire, le

renforcement r_t. Le composant de l'environnement qui fournit ce signal est généralement appelé « critique ». L'objectif de l'apprentissage par renforcement est d'associer à chaque état du système une action qui permet de maximiser la récompense.

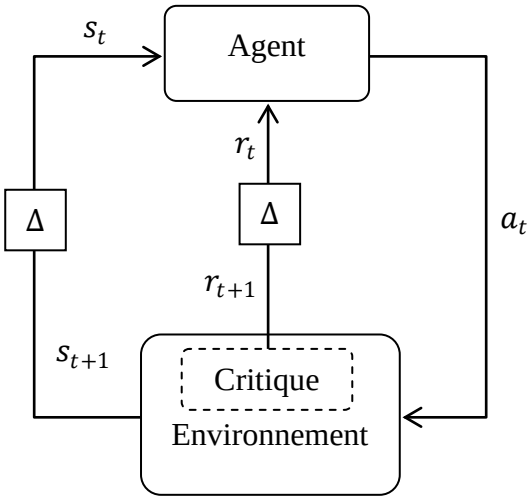


Figure 2.1 : Modèle standard de l'apprentissage par renforcement

Dans ce chapitre nous présentons une introduction à l'apprentissage par renforcement avec ces fondements mathématiques et nous décrivons les principaux algorithmes.

2.2 Fonction de retour

Comme nous l'avons vu dans l'introduction, le but de l'agent est de maximiser les récompenses futures cumulées. La fonction de retour, ou tout simplement le retour $R(t)$, est une mesure à long terme des récompenses. Il existe trois modèles principaux.

2.2.1 Modèle de retour à horizon fini

Dans ce cas, l'horizon correspond à un nombre fini de pas dans le futur. Il existe un état terminal et la suite d'actions entre l'état initial et l'état terminal est appelée une période. On a :

$$R(t) = \sum_{k=1}^{h} r_{t+k-1} = r_t + r_{t+1} + \cdots + r_{t+h-1} \quad (2.1)$$

où h est le nombre d'étapes avant l'état terminal. Ce nombre h peut être fixé à priori quelconque mais fini.

2.2.2 Modèle de retour à horizon infini γ − pondéré

On utilise ce modèle quand la suite des actions est infinie. Le modèle précédent peut alors ne pas être défini. On introduit un facteur d'atténuation, appelé aussi facteur d'oubli (discount factor), γ, $0 \leq \gamma \leq 1$; le critère devient :

$$R(t) = \sum_{k=0}^{\infty} \gamma^k r_{t+k} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \quad (2.2)$$

La valeur de γ permet de moduler la période sur laquelle l'agent prend en compte les renforcements si $\gamma = 0$, l'agent est « myope » et ne considère que les récompenses immédiates ; plus γ est proche de 1, plus l'agent vise à long terme.

2.2.3 Modèle de retour moyen

L'agent cherche à maximiser la moyenne de tous les renforcements qu'il recevra :

$$R(t) = \lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n} \gamma^k r_{t+k} \quad (2.3)$$

2.3 Processus de décision markovien

En général, les algorithmes d'apprentissage par renforcement se placent dans le cadre des processus de décision markovien (PDM). L'interaction de l'agent avec son environnement est donc modélisée par un processus de décision markovien [BEL57].

Un PDM est défini par un quadruplet (S, A, T, R) tel que :

S est un ensemble fini d'états,

A est un ensemble fini des actions,

T est une fonction de transition markovienne de $S \times A \times S$ dans $[0,1]$; $T(s, a, s')$ représente la probabilité d'aller de s dans s' en effectuant l'action a,

R est une fonction markovienne de renforcement de $S \times A \times S$ dans $\mathcal{R}$; $R(s, a, s')$ représente le renforcement obtenu en effectuant l'action a dans l'état s et en retrouvant alors l'état s'.

On dit que la fonction de transition est markovienne car la probabilité de transition entre s et s', lorsque l'on effectue l'action a, ne dépend pas des états précédemment visités et/ou des actions précédemment effectuées. Similairement on dit que la fonction de renforcements est markovienne car le renforcement reçu lors de la transition entre s et s', lorsque l'on effectue l'action a, ne dépend pas des états précédemment visités et/ou des actions précédemment effectuées.

Nous ne considérons dans ce mémoire que des PDM finis, c'est-à-dire pour lesquels les ensembles S et A sont finis. L'agent et l'environnement interagissent à chaque pas de temps d'une séquence $0,1,2,3,\ldots$ (cette séquence peut être infinie). Donc, à chaque pas de temps, l'agent perçoit l'état $s \in S$ dans lequel il se trouve, effectue une action $a \in A$ et au pas de temps suivant, l'agent perçoit son nouvel état $s' \in S$ et reçoit un renforcement r de valeur moyenne $R(s, a, s')$.

2.4 Politique et valeur de politique

2.4.1 Politique

La politique, notée π, est la règle avec laquelle un agent choisit une action dans un état donné. Une politique est dite déterministe ou stationnaire si elle spécifie la même action chaque fois qu'un état est visité. Une politique est dite non- déterministe ou stochastique si elle spécifie une action par la même distribution de probabilité sur l'ensemble des actions à chaque fois qu'un état est visité. Une telle politique est définie par une fonction sur tout l'ensemble des couples $états - actions$ dans l'intervalle $[0,1]$; $\pi: S \times A \longrightarrow [0,1]$ avec la contrainte suivante: $\sum_{a \in A} \pi(s,a) = 1$. Pour chaque état s, π nous donne la probabilité de choisir une action donnée a. Pendant l'apprentissage, le comportement de l'agent change et la politique est ni stationnaire ni stochastique ; cependant, les politiques optimales que l'agent cherche à construire seront stationnaires.

2.4.2 Valeur de politique

Dans cette section, nous allons définir la valeur associée à une politique donnée pour un PDM donné.

2.4.2.1 Valeur d'un état pour une politique donnée

La valeur d'un état $s \in S$ pour une politique π, notée $V^{\pi}(s)$ est définie par le renforcement espéré lorsqu'on part de l'état s et on suit la politique π par la suite. En utilisant le modèle de retour à horizon infini $\gamma - pondéré$, la fonction V^{π} se définit alors comme suit :

$$V^{\pi}(s) = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s\} \quad (2.4)$$

où E_{π} représente l'espérance lorsqu'on suit la politique π.

2.4.2.2 Valeur d'un couple état-action pour une politique donnée

Il est également possible de définir une autre fonction de valeur représentant la valeur d'un couple état-action. Cette fonction, notée $Q^{\pi}(s,a)$, est l'espérance des sommes actualisées des renforcements reçus lorsque l'on part de l'état s et que l'on exécute l'action a en s et que l'on suit la politique π par la suite. En utilisant le modèle de retour à horizon infini $\gamma - pondéré$ $Q^{\pi}(s,a)$ se définit comme suit :

$$Q^{\pi}(s,a) = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} | s_t = s\,, a_t = a\} \quad (2.5)$$

Cette fonction sera primordiale pour les algorithmes d'apprentissage que nous présenterons à la section 2.7.

2.4.2.3 Equation de Bellman

Les fonctions V^{π} et Q^{π} pour une politique π vérifient une relation fondamentale appelée *équation de Bellman* [BEL57]. Pour tout $s \in S$ on a :

$$V^{\pi}(s) = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} | s_t = s\} = E_{\pi}\{r_t + \gamma \sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} | s_t = s\}$$

L'équation de Bellman pour $V^{\pi}(s,a)$ est donnée par:

Si π est déterministe :

$$V^{\pi}(s) = \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')] \quad (2.6)$$

Si π est non-déterministe :

$$V^{\pi}(s) = \sum_{a \in A} \pi(s,a) \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')] \quad (2.7)$$

De la même manière, pour la fonction Q^{π} et pour tout $s \in S$ et tout $a \in A$ (π soit déterministe ou non), l'équation de Bellman pour $Q^{\pi}(s,a)$ est donnée par:

$$Q^{\pi}(s,a) = \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')] \quad (2.8)$$

Remarquons qu'on a les relations suivantes entre V^{π} et Q^{π} :

Si π est déterministe :

$$V^{\pi}(s) = Q^{\pi}(s,a) \quad (2.9)$$

Si π est non-déterministe :

$$V^{\pi}(s) = \sum_{a \in A} \pi(s,a)\, Q^{\pi}(s,a) \quad (2.10)$$

2.4.2.4 Calcul de la valeur d'une politique

L'équation de Bellman nous donne un premier moyen pour le calcul de la valeur d'une politique donnée. En écrivant l'équation de Bellman pour chaque état de S, on obtient un système d'équations linéaires. Si l'on connaît l'état initial et toutes les probabilités de

transitions, la valeur V^{π} d'une politique π est calculée en résolvant ce système d'équations.

2.4.3 Politique optimale

Dans cette section, nous définissons la notion d'une politique optimale pour tout PDM fini.

2.4.3.1 Amélioration d'une politique

Pour deux politiques déterministes π et π', tel que pour tout $s \in S$: $Q^{\pi}(s,\pi'(s)) \geq V^{\pi}(s)$, on peut montrer que, pour tout s on a : $V^{\pi'}(s) \geq V^{\pi}(s)$.

Similairement, pour deux politiques non-déterministes π et π' tel que pour tout $s \in S$: $\sum_{a \in A(s)} \pi'(s,a) Q^{\pi}(s,a) \geq V^{\pi}(s)$, on peut montrer que, pour tout s on a : $V^{\pi'}(s) \geq V^{\pi}(s)$
Ceci va nous servir dans le reste de ce chapitre et notamment dans les deux sections suivantes.

2.4.3.2 Existence d'une politique optimale

Les fonctions de valeur définissent un ordre partiel sur les politiques. On dit qu'une politique π_1 est supérieure à une autre politique π_2 $(\pi_1 \geq \pi_2)$ si et seulement si : $\forall\, s \in S, V^{\pi_1}(s) \geq V^{\pi_2}(s)$. Une politique π est dite optimale si et seulement si pour toute politique π' :$\forall\, s \in S, V^{\pi}(s) \geq V^{\pi'}(s)$. Une telle politique sera notée π^*.

La politique optimale π^*vérifie donc :

$$\forall\, s \in S, V^{\pi^*}(s) \geq V^{\pi}(s)$$

On note que s'il existe une politique optimale π^* non-déterministe alors il existe une politique déterministe π telle que : $\pi \geq \pi^*$. On peut démontrer que pour tout PDM fini, il existe une politique déterministe optimale (pour le critère de la somme actualisée des renforcements) si $0 \leq \gamma < 1$.

2.4.3.3 Equation de Bellman pour une politique optimale

La fonction de valeur V^* d'une politique optimale π^* vérifie l'équation de Bellman (2.6) :

$$V^*(s) = \sum_{s' \in S} T(s,\pi^*(s),s')[R(s,\pi^*(s),s') + \gamma V^*(s')] \qquad (2.11)$$

La fonction de valeur Q^* d'une politique optimale vérifie aussi l'équation de Bellman (2.8) :

$$Q^*(s,a) = \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] \qquad (2.12)$$

2.4.3.4 Equation d'optimalité de Bellman

Comme toute politique, la politique optimale vérifie l'équation (2.9), donc on a :

$$V^*(s) = Q^*(s, \pi^*(s,a)) \tag{2.13}$$

On a par définition :

$$V^*(s) = \max_{a \in A(s)} Q^*(s,a) \tag{2.14}$$

Bellman a montré que la fonction de valeur optimale représente la solution unique de l'équation suivante :

$$V^*(s) = \max_{a \in A(s)} \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] \tag{2.15}$$

Cette dernière équation est l'*équation d'optimalité de Bellman* pour la fonction V^*. On obtient de la même manière l'*équation d'optimalité de Bellman* pour la fonction Q^* :

$$Q^*(s,a) = \sum_{s' \in S} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a' \in A(s)} Q^*(s,a')\right] \tag{2.16}$$

2.5 La programmation dynamique

La programmation dynamique (PD) [BEL57] est un ensemble de techniques pour le calcul de la politique optimale dans les PDM déterministes. Tous les algorithmes de la PD exigent à la fois un modèle de transition et une fonction de récompense, c'est-à-dire que tous les états et toutes les transitions sont connus. Ces hypothèses ne sont pas vérifiées dans le cas de l'apprentissage par renforcement, mais l'étude des algorithmes de la programmation dynamique aide à comprendre les méthodes de l'apprentissage par renforcement. Cette méthode utilise des fonctions de valeur pour faire implicitement la recherche de meilleures politiques. Nous présentons dans ce qui suit deux algorithmes permettant de construire itérativement une politique déterministe optimale.

2.5.1 Algorithme d'itération des politiques (*Policy iteration*)

L'idée de l'algorithme d'itération des politiques est d'évaluer une certaine politique et, en fonction de cette évaluation, on calcule une nouvelle politique qui est meilleure que l'ancienne et ainsi de suite. On obtient alors le schéma suivant :

$$\pi_0 \xrightarrow{Evaluation} V^{\pi_0} \xrightarrow{Action} \pi_1 \xrightarrow{Evaluation} V^{\pi_1} \xrightarrow{Action} \pi_2 \xrightarrow{Evaluation} \dots \xrightarrow{Action} \pi^* \xrightarrow{Evaluation} V^*$$

Durant la $i^{\text{ème}}$ étape d'évaluation, l'algorithme calcule la valeur d'un état V^{π_i} pour la présente politique π_i par la conversion de l'équation de Bellman (2.7) en une règle de mise à jours itérative.

$$V_{t+1}(s) = \sum_{a \in A} \pi(s,a) \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V_t(s')] \tag{2.17}$$

A chaque pas d'itération, l'algorithme remplace la valeur ancienne de tout état s par une nouvelle valeur obtenue à partir des valeurs anciennes des états successeurs de s et les récompenses immédiates, pondérées par les probabilités de transition $T(s,a,s')$. Notons qu'il est garanti que la séquence $\{V_t\}$ converge à la valeur réelle V^π quand $t \longrightarrow \infty$. Le pseudo code de l'algorithme complet de l'étape d'évaluation est donné par Algorithme 2.1.

Algorithme 2.1 : Algorithme d'évaluation itérative d'une politique

Entrée :
$MDP: M = (S, A, T, R)$
Politique : π
Sortie :
V^π
Début
Initialiser pour tout $s \in S$ $V_0(s)$
Répéter
$\Delta \leftarrow 0$
Pour tout $s \in S$ **Faire**
$V_{t+1}(s) \leftarrow \sum_{a \in A} \pi(s,a) \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V_t(s')]$
$\Delta \leftarrow \max(\Delta, |V_{t+1}(s) - V_t(s)|)$
Fin Pour
Jusqu'à $\Delta < \varepsilon$
Retourner $V \approx V^\pi$
Fin

Durant la i^{ème} étape d'amélioration, l'algorithme améliore π_i en générant une nouvelle politique π_{i+1} de tel sorte que $V_{t+1} \geq V_t$. Considérons un état $s \in S$, et supposons que π_i est telle que dans s l'action a est choisie. Afin de déterminer si une autre action a' est meilleure que a dans s nous calculons la fonction valeur de l'action $Q^{\pi_i}(s,a')$:

$$Q^{\pi_i}(s,a') = \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V^{\pi_i}(s')] \quad (2.18)$$

qui estime l'utilité de prendre a' dans s et suivre la politique π_i par la suite. Si $Q^{\pi_i}(s,a') \geq V^{\pi_i}$, la nouvelle politique π_{i+1} est une amélioration de la politique originale π_i. Par l'extension de cette idée à tous les états, on obtient l'algorithme pour l'étape d'amélioration de politique, qui permet de trouver une nouvelle politique gloutonne π_{i+1} tel que :

$$\pi_{i+1}(s) = arg\,max_a\, Q^{\pi_i}(s,a) = arg\,max_a \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V^{\pi_i}(s')] \quad (2.19)$$

Cette étape assure l'amélioration de politique dans le sens que la nouvelle politique π_{i+1} est au moins aussi bonne que la politique originale. Comme dans un PDM fini il existe un nombre fini de politiques, et la séquence des politiques s'améliore à chaque étape,

l'algorithme d'itération des politiques converge à la politique optimale dans un nombre fini d'itérations. L'Algorithme 2.2 résume la procédure d'itération des politiques.

Algorithme 2.2 : Algorithme d'itération sur les politiques (*Policy iteration*)

Entrée :

$MDP: M = (S, A, T, R)$

Sortie :

$V^*,\ \pi^*$

Début

Choisir arbitrairement une politique π_i

Répéter

Evaluer la politique π_i par l'algorithme d'évaluation itérative (Algorithme 2.1)

Améliorer π_i pour chaque état :

$$\pi_{i+1}(s) \leftarrow arg\ max_a \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V^{\pi_i}(s')]$$

Jusqu'à $\pi_{i+1}(s) = \pi_i(s)$

Retourner $\pi \approx \pi^*$

Fin

2.5.2 Algorithmes d'itération des valeurs (*Value iteration*)

Dans cette méthode, on ne cherche pas à déterminer explicitement la politique mais l'action de valeur optimale pour chaque état. On définit la valeur d'une action, $Q^{\pi}(s, a)$, comme le retour escompté si l'action a est appliquée dans l'état s et si la politique π est suivie ensuite :

$$Q_{t+1}(s, a) = \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V_t(s')] \quad (2.20)$$

La règle de mise à jour pour l'itération sur les valeurs devient alors :

$$V_{t+1}(s) = \max_a Q_{t+1}(s, a) = \max_a \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V_t(s')] \quad (2.21)$$

L'Algorithme 2.3 résume les étapes de la procédure d'itération des valeurs.

On peut prouver que cet algorithme converge à la fonction de valeur optimale : $V_t \longrightarrow V^*$ quand $t \longrightarrow \infty$. Comme pour l'itération sur les politiques, l'itération sur les valeurs s'arrête quand l'amélioration de la fonction de valeur est inférieure à un seuil donné.

2.6 Les différences temporelles(TD)

Dans les techniques des différences temporelles [SUT88, SUT98], l'agent apprend directement de l'expérience sans faire recours à aucun modèle de l'environnement.

Algorithme 2.3: Algorithme d'itérations sur les valeurs (*Value iteration*)

Entrée :

$MDP: M = (S, A, T, R)$

Politique : π

Sortie :

π^*

Début

Initialiser $V_0(s)$ pour tout $s \in S$

Répéter

$\Delta \leftarrow 0$

Pour tout $s \in S$ **Faire**

$V_{t+1}(s) \leftarrow max_a \sum_{a \in A} \pi(s,a) \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V_t(s')]$

$\Delta \leftarrow \max(\Delta, |V_{t+1}(s) - V_t(s)|)$

Fin Pour

Jusqu'à $\Delta < \varepsilon$

Pour tout $s \in S$ **Faire**

$\pi_{i+1}(s) \leftarrow arg\, max_a \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V^{\pi_i}(s')]$

Fin Pour

Retourner $\pi \approx \pi^*$

Fin

2.6.1 Algorithme TD(0)

L'algorithme le plus simple des différences temporelles est appelé TD(0) (Algorithme 2.4). A chaque pas de temps, l'agent à l'état s prend une action a selon sa politique actuelle π et l'estimée de la fonction valeur dans l'état s est mise à jour comme suit :

$$V_t(s) \leftarrow (1 - \alpha_t) \times V_t(s) + \alpha_t \times \left(r + \gamma V_t(s')\right) \quad (2.22)$$

où α_t est le pas d'apprentissage, s' l'état obtenu par la prise de l'action a et r le signal de renforcement. La convergence de l'algorithme à V^π quand $t \longrightarrow \infty$ peut être démontrée, étant donné que les signaux de renforcement sont bornés, que le processus est Markovien et que le pas d'apprentissage vérifie les conditions suivantes :

$$\sum_{t=1}^{\infty} \alpha_t = \infty; \ \sum_{t=1}^{\infty} {\alpha_t}^2 < \infty$$

Comme l'algorithme TD(0) donne une estimation de la fonction valeur V^π, il ne peut pas être utilisé pour l'apprentissage d'une politique optimale. Dans ce cas, il est nécessaire de calculer une estimation de la fonction de valeur de l'action $Q^*(s,a)$.

2.6.2 Algorithme TD(λ)

L'algorithme précédent TD(0) ne prend en compte que l'étape qui suit dans l'évaluation. De plus seul l'état courant est concerné. L'algorithme TD(λ) recouvre une classe d'algorithmes qui mettent à jour tous les états dernièrement visités à l'aide de la

différence temporelle de l'état courant. Une pondération de cette mise à jour est déterminée par l'ancienneté de la dernière visite. Dans ce sens, ces algorithmes utilisent la notion de trace ou d'éligibilité.

Une fonction d'éligibilité qualifie, à l'aide d'un nombre réel, l'ancienneté de la dernière visite de chaque couple état-action. L'éligibilité d'un couple (s,a) est notée $e(s,a)$. A chaque pas d'échantillonnage, les valeurs d'éligibilité diminuent de façon exponentielle. L'éligibilité peut être définie par plusieurs façons [SUT98, GLO02, KAE96]. L'éligibilité accumulatrice :

$$e_t(s) = \begin{cases} 1 + \gamma\lambda e_{t-1}(s) & si\ s = s_t \\ \gamma\lambda e_{t-1}(s) & sinon \end{cases} \quad (2.23)$$

Tandis qu'une éligibilité remplaçante correspond à :

$$e_t(s) = \begin{cases} 1 & si\ s = s_t \\ \gamma\lambda e_{t-1}(s) & sinon \end{cases} \quad (2.24)$$

La mise à jour devient donc :

$$V_t(s) \leftarrow V_t(s) + \alpha \times \left(r + \gamma V_t(s') - V_t(s)\right) e_t(s) \quad (2.25)$$

Le paramètre λ $(0 \leq \lambda \leq 1)$ est appelé facteur de proximité (*recency factor*).

Pour plus de détail sur l'éligibilité, consulter la référence [SUT98].

Algorithme 2.4: Algorithme du *TD(0)*

Entrée :

MDP: $M = (S, A, ?, ?)$

π : une politique

Sortie :

$V \approx V^{\pi}$

Début

Initialiser $V_0(s)$ arbitrairement pour tout $s \in S$

Initialiser s

Pour tout épisode **Faire**

Initialiser s

Répéter

$a \leftarrow \pi(s)$

Effectuer l'action a

Recevoir le renforcement r et le nouvel état s'

$V_t(s) \leftarrow (1-\alpha) \times V_t(s) + \alpha \times \left(r + \gamma V_t(s')\right)$

$s \leftarrow s'$

Jusqu'à $s\ terminal$

Fin Pour

Retourner $V \approx V^{\pi}$

Fin

2.7 Apprentissage par renforcement

L'inconvénient principal de la programmation dynamique est la supposition de la disponibilité à la fois du modèle de transition et de la fonction de récompense. Dans le cadre de l'apprentissage par renforcement, on suppose que les probabilités de transition et la fonction de récompense sont inconnues.

L'idée de base de l'apprentissage par renforcement est d'utiliser l'expérience ou, autrement dit, utiliser les parcours effectués dans l'espace d'états pour améliorer la performance.

Dans cette section, on présente deux variantes d'algorithmes d'apprentissage par renforcement ; le Q-learning et le Sarsa.

Algorithme 2.5: Algorithme du *Q-learning*

Entrée :

$MDP: M = (S, A, ?, ?)$

Une politique d'exploration/exploitation: PEE

Sortie :

$Q \approx Q^*$

Début

Initialiser $Q_0(s, a)$ arbitrairement pour tout $s \in S$ et $a \in A$

Initialiser s

Pour tout épisode **Faire**

Répéter

$a \longleftarrow PEE$

Effectuer l'action a

Recevoir le renforcement r et le nouvel état s'

$Q_t(s, a) \longleftarrow (1 - \alpha) \times Q_t(s, a) + \alpha \times \left(r + \gamma \max_{a' \in A(s')} Q_t(s', a')\right)$

$s \longleftarrow s'$

Jusqu'à s *terminal*

Fin Pour

Retourner $Q \approx Q^*$

Fin

2.7.1 Algorithme *Q-Learning*

L'algorithme du Q-learning a été introduit par Watkins [WAT89]. Il est sans doute l'algorithme d'apprentissage par renforcement le plus utilisé. Son succès revient à sa simplicité et la démonstration de sa convergence [WAT92].

La première version du *Q-Learning* est basée sur les différences temporelles d'ordre 0, TD(0), en considérant seulement le pas suivant (*one-step Q-Learning*). A chaque pas de temps, l'agent observe l'état dans lequel il se trouve, s_t, et exécute une action, a_t, en

fonction de l'estimation qu'il fait du retour, à cette étape, selon le critère $\gamma - pondéré$. Il remet ensuite à jour son estimation de la valeur de l'action en prenant en compte le renforcement immédiat, r_{t+1}. L'algorithme 2.5 présente le pseudo-code du Q-learning.

Algorithme 2.6: Algorithme *Sarsa*

Entrée :
$MDP: M = (S, A, ?, ?)$
Une politique d'exploration/exploitation: PEE
Sortie :
$Q \approx Q^*$
Début
Initialiser $Q_0(s, a)$ arbitrairement pour tout $s \in S$ et $a \in A$
Initialiser s
$a \longleftarrow PEE$
Pour tout épisode **Faire**
Répéter
Effectuer l'action a
Recevoir le renforcement r et le nouvel état s'
$a' \longleftarrow PEE$
$Q_t(s, a) \longleftarrow (1 - \alpha) \times Q_t(s, a) + \alpha \times (r + \gamma Q_t(s', a'))$
$a \longleftarrow a'; s \longleftarrow s'$
Jusqu'à $s\ terminal$
Fin Pour
Retourner $Q \approx Q^*$
Fin

La politique d'exploration/exploitation PEE permet de choisir une action dans chaque état. Elle doit assurer que chaque action ait une probabilité non nulle d'être choisie. La PEE souvent utilisée est la politique d'exploration/exploitation pseudo-stochastique $\varepsilon - gloutonne\ (\varepsilon - greedy)$ (voir la section 2.8).

2.7.2 Algorithme *Sarsa*

Nous présenterons dans cette section un deuxième algorithme d'apprentissage d'une politique optimale. Cet algorithme va nous permettre de présenter les notions d'algorithme on-policy et off-policy. Dans l'algorithme Sarsa, on se sert de la PEE suivie pour estimer la valeur du prochain état s'. L'algorithme du Q-learning estime la valeur du prochain état s' sans prendre en considération la PEE. En effet, on estime la valeur de l'état s' grâce au maximum que l'on peut obtenir dans cet état. L'algorithme Sarsa est un algorithme dit on-policy car la PEE aura un effet sur la politique apprise. L'algorithme du Q-learning est dit off-policy car la PEE n'aura pas d'effet sur la politique apprise. La convergence de

l'algorithme Sarsa vers une politique optimale dépend de la PEE. Le pseudo code de cette méthode d'apprentissage est donné par l'algorithme 2.6.

2.7.3 Algorithmes Q(λ), Sarsa(λ)

Q(λ) et Sarsa(λ) [SUT98] sont respectivement la généralisation des algorithmes Q-learning et Sarsa qui utilisent les traces d'éligibilité (voir section 2.6.2).

2.8 Le dilemme exploration/exploitation

Contrairement à l'apprentissage supervisé, l'agent qui apprend par renforcement ne connait pas explicitement les actions qu'il doit entreprendre. L'agent doit donc expérimenter toutes les actions à sa disposition afin d'évaluer leurs qualités respectives. Il explore alors activement son environnement de façon à découvrir une politique optimale. Mais l'agent peut aussi, et doit utiliser la connaissance qu'il acquiert sur la valeur des actions pour biaiser son exploration. En effet, entre deux actions, il semble préférable de choisir celle ayant la plus forte valeur à l'instant présent. C'est celle qui nous apportera le plus de récompenses dans l'état actuel de la connaissance de l'agent. Mais comme cette connaissance n'est que partielle, il est nécessaire d'expérimenter toutes les actions. Comme nous l'avons vu dans la section précédente, l'agent va suivre une politique d'exploration/exploitation PEE lui indiquant quelle action entreprendre dans un état donné. Nous présentons ici quelques unes des méthodes les plus utilisées dans la littérature.

La politique optimale est obtenue en choisissant l'action qui, à chaque état, maximise la fonction de qualité :

$$a = \arg\max_{v \in U_x} Q^*(x, v) \tag{2.26}$$

Cette politique est appelée « *gloutonne* » (*greedy*). Cependant, au début de l'apprentissage, les valeurs $Q(x,u)$ ne sont pas significatives et la politique gloutonne n'est pas applicable. Pour obtenir une estimation utile de Q , il faut balayer et évaluer l'ensemble des actions possibles, pour tous les états : c'est ce que l'on appelle la phase d'exploration par rapport à l'exploitation, une fois l'apprentissage terminé. Il existe différentes politiques d'exploration [KAE96, GLO99]. Les deux méthodes d'exploration les plus utilisées sont décrites ci-dessous. Dans la suite, l'action choisie selon une politique d'exploration/exploitation donnée sera notée $PEE(s)$.

2.8.1 Méthode pseudo-stochastique

L'action avec la meilleure valeur Q a une probabilité P d'être choisie. Sinon, une action est choisie aléatoirement parmi toutes les actions possibles dans l'état donné.

2.8.2 Distribution de Boltzmann

L'action a dans l'état s est choisie avec la probabilité $p(a|s)$ donnée par:

$$p(a|s) = \frac{exp\left(\frac{1}{T}Q(s,a)\right)}{\sum_v exp\left(\frac{1}{T}Q(s,v)\right)} \quad (2.27)$$

Le paramètre T appelé *température* permet le contrôle du taux d'exploration. Plus le paramètre T est grand, plus le taux d'exploration sera important.

2.9 Conclusion

Ce chapitre nous a permis de nous initier au domaine de l'apprentissage par renforcement, de présenter ses fondements théoriques, et de décrire les principaux algorithmes standards. Nous avons vu que, dans l'apprentissage par renforcement, l'agent interagit avec son environnement à des pas de temps discrets. A chaque pas de temps, l'agent perçoit l'état de son environnement dans lequel il se trouve, effectue une action, et au pas de temps suivant, l'agent perçoit son nouvel état et reçoit une récompense ou un renforcement en fonction de la qualité de la transition. Nous avons vu qu'un problème d'apprentissage par renforcement est décrit par un PDM sans modèle de transition. Nous verrons, au chapitre 4, comment utiliser l'apprentissage par renforcement pour l'optimisation des paramètres d'un contrôleur flou.

Chapitre 3

Optimisation par colonies de fourmis

3.1 Introduction

L'Optimisation par Colonies de Fourmis (OCF), en anglais «Ant Colony Optimization (ACO)» [DOR96, DOR04, DRE03] est une méthaheuristique récemment utilisée par des chercheurs pour résoudre les problèmes de l'optimisation combinatoire.

L'ACO est l'une des technique de l'intelligence des essaims (en anglais : *Swarm Intelligence*) [KEN01]. Son développement est basé sur l'imitation des essaims d'animaux ou d'insectes pour résoudre des problèmes d'optimisation. L'ACO est basé sur le comportement d'une colonie de fourmis à la recherche de la nourriture. Le premier algorithme basé sur l'analogie avec les colonies de fourmis est apparu au début des années quatre-vingt-dix, appelé *Ant System* [DOR96a, DOR96b], et ses performances ont été initialement illustrées sur le problème du voyageur de commerce. Malgré des premiers résultats encourageants montrant que les performances de *Ant System* sont comparables à celles des autres approches « généralistes » comme les algorithmes génétiques, l'algorithme *Ant System* s'est avéré non compétitif avec des approches spécialisées, développées spécialement pour le problème du voyageur de commerce. Ainsi, différentes améliorations ont été apportées à l'algorithme initial, donnant naissance aux différentes variantes de *Ant System*, comme par exemple *ACS* (*Ant Colony System*) [DOR97] et

MMAS (MAX-MIN *Ant System*) [STU00] qui obtiennent en pratique des résultats vraiment compétitifs, parfois meilleurs que les approches les plus performantes.

Ce chapitre présente une introduction à l'optimisation par colonies de fourmis avec l'explication de son inspiration biologique, décrit la formalisation d'un problème pour l'optimisation par colonies de fourmis, et donne les algorithmes de base existant dans la littérature.

3.2 Inspiration biologique

L'idée d'utiliser des fourmis pour l'optimisation est inspirée de l'observation du comportement de fourmis réelles à la recherche de la nourriture dans les environs du nid. On s'aperçoit qu'elles sont capables de trouver le plus court chemin reliant le nid à la source de nourriture.

Initialement, les fourmis explorent l'entourage de leur nid de façon aléatoire. Dès qu'une source de nourriture est retrouvée, elles évaluent la quantité et la qualité de la nourriture et emportent un peu de cette nourriture à leur nid. Pendant le voyage du nid à la source de nourriture et vice-versa, les fourmis déposent au passage sur le sol une substance chimique odorante appelée phéromones. Cette substance permet ainsi donc de créer une piste, sur laquelle les fourmis s'y retrouvent. En effet, d'autres fourmis peuvent détecter les phéromones grâce à des capteurs sur leurs antennes.

Les phéromones ont un rôle de marqueur de chemin : quand les fourmis choisissent leur chemin, elles ont tendance à choisir la piste qui porte la plus forte concentration de phéromones. Cela leur permet de retrouver le chemin vers leur nid lors du retour. D'autre part, les odeurs peuvent être utilisées par les autres fourmis pour retrouver les sources de nourritures trouvées par leurs congénères.

Cette communication indirecte entre les fourmis via les traces de phéromone leur permet le choix du chemin le plus court de leur nid à la source de nourriture.

3.2.1 Expériences

Pour bien étudier le comportement des fourmis, des chercheurs ont mené différentes expériences [GOS89, DEN90, BEK92]. Ces expériences ont permis l'élaboration d'un modèle comportemental mathématique et la naissance d'un panel d'algorithmes d'optimisation.

L'expérience la plus connue est celle du pont binaire [DEN90], dans laquelle, le nid des fourmis et la source de nourriture sont séparés par un pont binaire constitué de deux branches identiques (figure 3.1). Au départ, il n'y a aucune trace de phéromone sur les deux branches, chacune donc peut être choisie avec la même probabilité (figure 3.1 (a)). Après une phase de fluctuations, la quantité de phéromone a tendance à s'accumuler sur un chemin et il a été observé alors que les fourmis prenaient collectivement le même itinéraire après quelques minutes (figure 3.1 (b)).

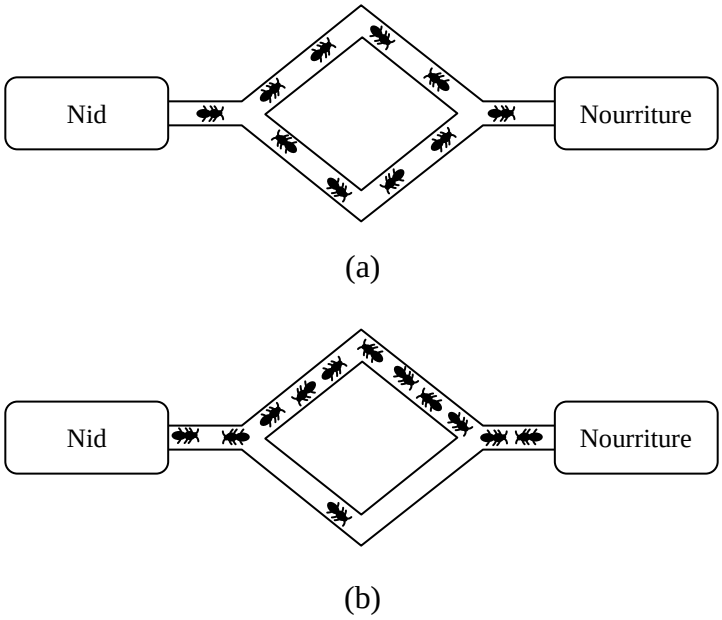


Figure 3.1 : Expérience du pont binaire de Deneubourg

Une variante de l'expérience du pont binaire considère un double pont avec des branches de longueurs différentes (figure 3.2). L'effet produit sera que l'itinéraire le plus court sera sélectionné. En effet, les premières fourmis qui ont pris le chemin le plus court (les deux branches les plus courtes) arrivent à la source de nourriture en premier, et retournent au nid avec la nourriture en premier. Ce chemin, marqué deux fois par les phéromones, attire plus les autres fourmis que le long chemin (les deux branches longues), qui lui est marqué une seule fois dans le sens de l'aller. Cet effet se renforce au fur et à mesure, jusqu'à ce que toutes les fourmis choisissent le chemin le plus court.

Dans une autre expérience, alors que les fourmis suivent une piste de phéromone (figure 3.3 (a)), un obstacle est placé afin de leur barrer la route (figure 3.3 (b)). Les fourmis ont alors le choix de contourner cet obstacle par la droite ou par la gauche. Puisque, il n'y a aucune trace de phéromone le long de l'obstacle, au début, les fourmis choisissent au

hasard leur chemin, il y'aura autant de fourmis qui partent à gauche qu'à droite (figure 3.3 (b)). Après un certain temps, les chercheurs ont observé que les fourmis qui ont choisi le chemin le plus court, parviennent à reconstituer plus rapidement la concentration en phéromones sur ce nouveau chemin que celles qui ont choisi le chemin le plus long. Ainsi la concentration de phéromone sur le plus court chemin va augmenter, incitant ainsi d'autres fourmis à choisir le chemin riche en phéromones. Du à ce processus auto-catalytique, les fourmis vont finalement choisir le chemin le plus court (figure 3.3 (c)).

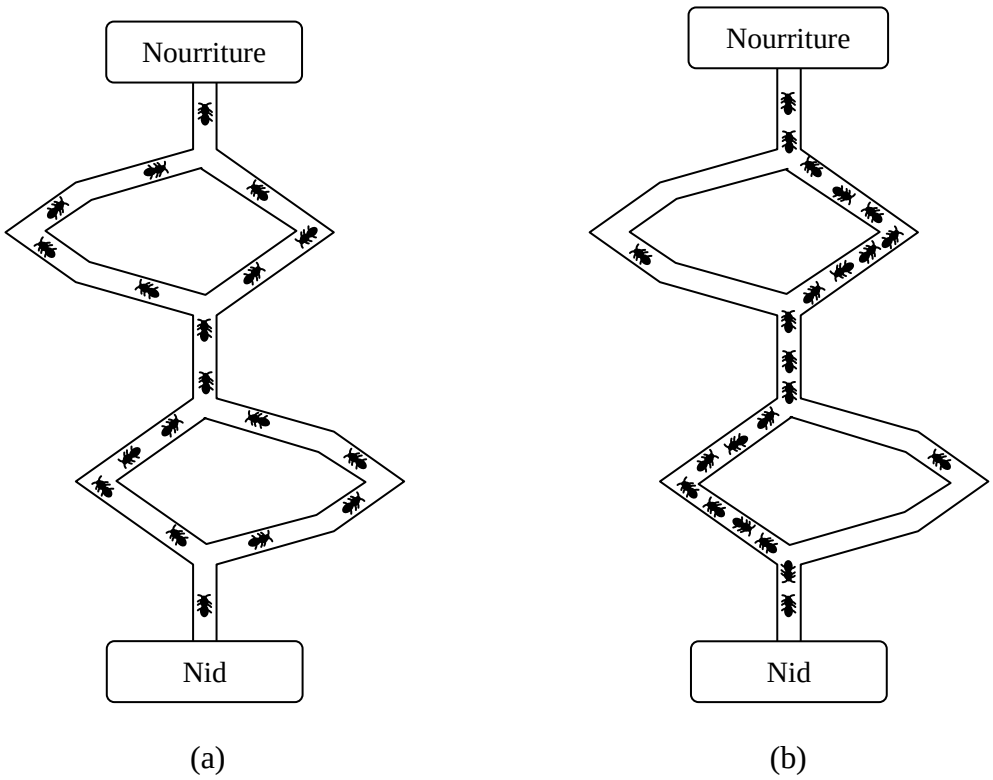


Figure 3.2 : Expérience du double pont binaire : (a) au début, (b) à la fin

3.2.2 Fourmis artificielles

Ce comportement naturel des fourmis réelles a été modélisé par des fourmis artificielles et adapté à la résolution des problèmes d'optimisation combinatoires sous le nom de méthaheuristique « Optimisation par Colonies de Fourmis».

Les fourmis artificielles ont aussi d'autres caractéristiques, qui ne trouvent pas leur équivalent dans la nature. En particulier, elles peuvent avoir une mémoire, qui leur permet de se souvenir de leurs actions passées. Dans la plupart des cas, les fourmis ne déposent une trace de phéromone qu'après avoir effectué un chemin complet, et non de façon incrémentale au fur et à mesure de leur progression. Enfin, la probabilité pour une fourmi

artificielle de choisir une piste ne dépend généralement pas uniquement des traces de phéromone, mais aussi d'heuristiques dépendantes du problème permettant d'évaluer localement la qualité du chemin.

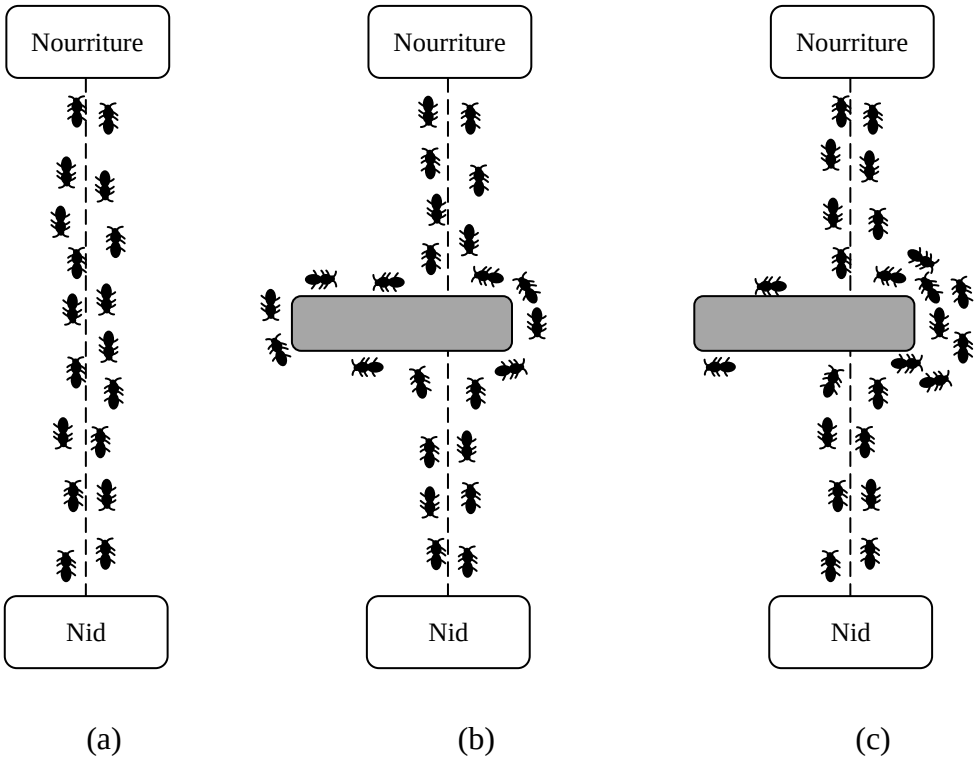


Figure 3.3 : Expérience de l'obstacle ; (a) les fourmis font des aller et retours entre le nid et la source de nourriture, (b) introduction de l'obstacle entre le nid et la source de nourriture, au début, les fourmis explorent les deux itinéraires, (c) Après, les fourmis choisissent le chemin le plus court

3.3 De l'inspiration au modèle mathématique

L'optimisation par colonies de fourmis a été formalisée en une méthaheuristique pour les problèmes d'optimisation combinatoire [DOR99]. Une méthaheuristique est un ensemble de concepts algorithmiques qui peuvent être utilisés pour définir des méthodes heuristiques appliqués à une grande variété de problèmes.

Les problèmes d'optimisation combinatoires traités par l'ACO sont généralement représentés par un modèle $M = (S, \Omega, f)$ avec :

S : représente l'espace de recherche défini sur un ensemble fini de variables discrètes X_i, $i = 1, \dots, n$;

Ω : représente l'ensemble des contraintes du problème ;

f : la fonction objective à minimiser.

Les valeurs possibles de la variable générique X_i sont définies par l'ensemble $D_i = \left\{v_i^1,\ v_i^2, \dots, v_i^{|D_i|}\right\}$. Une solution possible $s \in S$ est un assignement complet dans lequel à chaque variable est assignée une valeur en respectant les contraintes de l'ensemble Ω. Une solution $s^* \in S$ est dite optimum global si et seulement si : $f(s^*) \leq f(s), \forall\ s \in S$.

Le modèle combinatoire du problème $M = (S, \Omega, f)$ est utilisé pour le calcul d'un modèle de phéromone utilisé par ACO. Au début, une variable de décision $X_i = v_i^j$ (c.à.d., une variable X_i avec une valeur assignée de son domaine D_i) est une solution composante représentée par c_{ij}. L'ensemble de toutes les solutions composantes possibles est représenté par C. Un paramètre T_{ij} représentant la trace de phéromone est associé à chaque composant c_{ij}. L'ensemble de tous les paramètres de trace de phéromones T_{ij} est noté T. La valeur du paramètre T_{ij} est représentée par τ_{ij} (nommée la valeur du phéromone). Cette valeur est donc utilisée et adaptée par l'algorithme ACO pendant la recherche. Elle permet de modéliser la distribution de probabilité des différents composants de la solution.

Dans ACO, les fourmis artificielles construisent une solution à un problème d'optimisation combinatoire en traversant un graphe $Gc = (N, A)$, où N représente l'ensemble des nœuds et A représente l'ensemble des arcs connectant les nœuds. L'ensemble des composants C peut être associé à N ou à A. Les contraintes du problème sont directement implémentées dans les règles de déplacement des fourmis. Les fourmis se déplacent d'un nœud à un autre le long des arcs, en construisant progressivement une solution partielle. En plus, les fourmis déposent une certaine quantité de phéromone sur les composants ; soit sur les nœuds ou sur les arcs. La quantité de phéromone déposée $\Delta\tau$ peut dépondre de la solution trouvée.

La méthaheuristique ACO est donnée par l'algorithme 3.1. Elle consiste en une étape d'initialisation et une boucle de trois composantes de calcul. L'itération d'une boucle consiste en la construction d'une solution par toutes les fourmis, l'amélioration de la solution par un algorithme de recherche locale, et une mise à jour de la phéromone. Dans la suite on va expliquer en détail le rôle des trois composantes arithmétiques.

Algorithme 3.1: Optimisation par colonies de fourmis

Entrées :

Graphe : $Gc = (N, A)$

Sortie :

$la\ meilleur\ solution$

Début

Initialiser les paramètres et les pistes de phéromone

Quand le critère de performance n'est pas satisfait

Pour chaque fourmi $k = 1, \dots, nombre\ de\ fourmis$

Construire les solutions

Fin Pour

Faire la mise à jour de la phéromone

Fin Quand

Retourner $la\ meilleur\ solution$

Fin

3.3.1 Construction d'une solution par les fourmis

Un ensemble de m fourmis artificielles construisent les solutions à partir des éléments de l'ensemble fini des composants possibles de la solution $C = \{c_{ij}\}, i = 1, \dots, n, j = 1, \dots, |D_i|$.

La construction d'une solution commence avec une solution partielle vide $s^P = \phi$. Puis, à chaque étape de la construction, la solution partielle s^P est étendue par l'ajout d'une composante de la solution possible de l'ensemble des voisins possibles $N(s^P) \subseteq C$. Le processus de la construction des solutions peut être vu comme un chemin dans le graphe $Gc = (N, A)$. Les chemins autorisés dans Gc sont définis implicitement par le mécanisme de construction de la solution qui définit l'ensemble $N(s^P)$ par rapport à une solution partielle s^P.

Le choix d'une composante de la solution dans $N(s^P)$ est donné par une loi de probabilité à chaque itération de la construction. Les règles exactes pour le choix probabiliste des composants de la solution varient selon les différentes variantes de l'OCF. La règle la plus utilisée est la suivante :

$$Pr(c_{ij}/s^P) = \frac{{\tau_{ij}}^{\alpha} \cdot \eta(c_{ij})^{\beta}}{\sum_{c_{il} \in N(s^P)} {\tau_{il}}^{\alpha} \cdot \eta(c_{ij})^{\beta}},\ \forall c_{ij} \in N(s^P) \qquad (3.1)$$

où η est une fonction de pondération, c'est une fonction qui, parfois en fonction de la solution partielle actuelle, assigne à chaque étape de construction une valeur heuristique $\eta(c_{ij})$ pour chaque composante possible de la solution $c_{ij} \in N(s^P)$. Par ailleurs, α et β

sont des coefficients positifs, leurs valeurs déterminent les degrés d'importance relative de la phéromone et de l'information heuristique.

3.3.2 La recherche locale

Après la construction des solutions et avant la mise à jour de la phéromone, souvent des actions optionnelles appelés les actions de démon (en anglais *daemon actions*) peuvent être nécessaires. Elles peuvent être utilisées pour implémenter des actions centralisées qui ne peuvent pas être réalisées par des fourmis individuelles. L'action de démon la plus utilisée consiste en l'application de la recherche locale dans les solutions construites. Les solutions localement optimisées sont donc utilisées pour décider à quelle phéromone il faut faire la mise à jour.

3.3.3 Mise à jour de la phéromone

L'objectif de la mise à jour de la phéromone est d'augmenter les valeurs de la phéromone associées aux bonnes solutions et de diminuer celles des solutions médiocres. Souvent, cette opération est réalisé par :

1. La diminution de toutes les valeurs de la phéromone par l'évaporation de la phéromone.
2. L'augmentation des valeurs de la phéromone associées à un ensemble choisi de bonnes solutions S_{upd} :

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho \sum_{s \in S_{upd} \mid c_{ij} \in s} F(s) \qquad (3.2)$$

avec S_{upd} l'ensemble des solutions utilisées pour la mise à jour, $\rho \in [0,1]$ est un paramètre appelé le taux d'évaporation, et $F: S \rightarrow \mathbb{R}_0^+$ est une fonction tel que $f(s) < f(s') \Rightarrow F(s) \geq F(s'), \forall\, s \neq s' \in S$, $F(.)$ est généralement appelé fonction de performance.

L'évaporation est utilisée pour éviter la convergence précoce de l'algorithme. Elle implémente une forme utile d'oubli, qui favorise l'exploration de nouveaux domaines de l'espace de recherche. Les algorithmes de l'ACO diffèrent essentiellement dans la façon de la mise à jour de la phéromone. Dans la section suivante, nous allons présenter différentes versions de l' ACO.

3.4 Différentes variantes de l'ACO

3.4.1 Algorithme « Ant System (AS) »

L'importance du premier algorithme inspiré du comportement des fourmis *« Ant System »* réside essentiellement dans sa caractéristique d'être le premier prototype de base à un nombre d'algorithmes basé sur les fourmis. Il est initialement développé pour le problème du voyageur de commerce (PVC) (en anglais : *Travelling Salesman Problem (TSP)*) [DOR96a, DOR96b]. Il est intéressant d'approfondir le principe du premier prototype pour bien comprendre le mode de fonctionnement des algorithmes de colonies de fourmis.

Le PVC consiste à trouver le trajet le plus court (désigné par un tour) reliant n villes données, chaque ville ne devant être visitée qu'une seule fois. Le problème est plus généralement défini comme un graphe complètement connecté $Gc = (N, A)$ ou les villes sont les nœuds N et les trajets entre ces villes, les arêtes A.

Dans l'algorithme AS, à chaque itération t $(0 \leq t \leq t_{max})$, chaque fourmi k $(k = 1, \dots, m)$ parcourt le graphe et construit un trajet complet de $n = |N|$ étapes (on note $|N|$ le cardinal de l'ensemble N). Pour chaque fourmi, le trajet entre une ville i et une ville j dépend de :

- la liste des villes déjà visitées, qui définit les mouvements possibles à chaque pas, quand la fourmi k est sur la ville i : J_i^k ;
- l'inverse de la distance entre les villes : $\eta_{ij} = \frac{1}{d_{ij}}$, appelée visibilité. Cette information « statique » est utilisée pour diriger le choix des fourmis vers des villes proches, et éviter les villes trop lointaines ;
- la quantité de phéromone déposée sur l'arête reliant les deux villes, appelée l'intensité de la piste. Ce paramètre définit l'attractivité d'une partie du trajet global et change à chaque passage d'une fourmi. C'est, en quelque sorte, une mémoire globale du système, qui évolue par apprentissage.

La règle de déplacement (appelée règle aléatoire de transition proportionnelle) est la suivante :

$$P_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}(t)^\alpha \cdot \eta_{ij}{}^\beta}{\sum_{l \in J_i^k} \tau_{il}(t)^\alpha \cdot \eta_{ij}{}^\beta}, & si\ j \in J_i^k \\ 0 & si\ j \notin J_i^k \end{cases} \quad (3.3)$$

où α et β sont deux paramètres contrôlant l'importance relative de l'intensité de la piste, τ_{ij}, et la visibilité, η_{ij}. Avec $\alpha = 0$, seule la visibilité de la ville est prise en compte ; la

ville la plus proche est donc choisie à chaque pas. Au contraire, avec $\beta = 0$, seules les pistes de phéromone jouent. Pour éviter une sélection trop rapide d'un trajet, un compromis entre ces deux paramètres, jouant sur les comportements de diversification et d'intensification est nécessaire.

Après un tour complet, chaque fourmi laisse une certaine quantité de phéromone $\Delta\tau_{ij}^{k}(t)$ sur l'ensemble de son parcours, quantité qui dépend de la qualité de la solution trouvée :

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{Q}{L_k(t)} & si\ (i,j) \in T^k(t) \\ 0 & si\ (i,j) \notin T^k(t) \end{cases} \quad (3.4)$$

où $T^k(t)$ est le trajet effectué par la fourmi k à l'itération t, $L_k(t)$ la longueur de la tournée et Q un paramètre fixé.

L'algorithme ne serait pas complet sans le processus d'évaporation des pistes de phéromone. En effet, pour éviter d'être piégé dans des solutions sous-optimales, il est nécessaire de permettre au système « d'oublier » les mauvaises solutions. On contrebalance donc l'additivité des pistes par une décroissance constante des valeurs des arêtes à chaque itération. La règle de mise à jour des pistes est donc :

$$\tau_{ij}(t+1) \leftarrow (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \quad (3.5)$$

où m est le nombre de fourmis et ρ le taux d'évaporation. La quantité initiale de phéromone sur les arêtes est une distribution uniforme d'une petite quantité $\tau_0 \geq 0$. Le pseudo-code de AS est donné par l'algorithme 3.2.

3.4.2 *Ant System* et élitisme

La première variante de l'algorithme « Ant system » a été proposée dans [DOR96a], elle est caractérisée par l'introduction de fourmis « élitistes ». Dans cette version, la meilleure fourmi (celle qui a effectue le trajet le plus court) dépose une quantité de phéromone plus grande, dans l'optique d'accroître la probabilité des autres fourmis d'explorer la solution la plus prometteuse.

3.4.3 *Ant-Q*

Dans cette variante de AS, la règle de mise à jour locale est inspirée du « Q-learning » [GAM95]. Cependant, aucune amélioration par rapport à l'algorithme AS n'a pu être démontrée. Cet algorithme n'est d'ailleurs, de l'aveu même des auteurs, qu'une pré-version du « *Ant Colony System* (ACS».

Algorithme 3.2: Algorithme « *Ant System* »

Entrée :

Graphe $Gc = (N, A)$

Sortie :

Le tour le plus court T^*

Début

Initialiser les pistes de phéromone $\tau_{ij} = \tau_0$

Pour $t = 1, \dots, t_{max}$

Pour chaque fourmi $k = 1, \dots, m$

Choisir une ville au hasard

Pour Chaque ville non visité i

Choisir une ville j dans la liste J_i^k des villes restantes suivant la relation (3.3)

Fin Pour

Faire la mise à jour de la phéromone suivant la relation (3.4)

Fin Pour

Faire l'évaporation de la phéromone suivant la relation (3.5)

Fin Pour

Retourner T^*

Fin

3.4.4 *Ant Colony System (ACS)*

L'algorithme « *Ant Colony System (ACS)* » a été introduit pour améliorer les performances du premier algorithme sur des problèmes de grandes tailles [DOR97]. *ACS* est fondé sur des médications du *AS* :

1. ACS introduit une règle de transition dépendant d'un paramètre q_0 $(0 \leq q_0 \leq 1)$, qui définit une balance diversification/intensification. Une fourmi k sur une ville i choisira une ville j par la règle :

$$\Delta\tau_{ij}^k(t) = \begin{cases} argmax_{u \in J_i^k} \{\tau_{iu}(t) \cdot {\eta_{ij}}^{\beta}\} & si\ q \leq q_0 \\ J & si\ q > q_0 \end{cases} \quad (3.6)$$

où q est une variable aléatoire uniformément distribuée sur $[0,1]$ et $J \in J_i^k$ sélectionnée aléatoirement selon la probabilité :

$$P_{ij}^k = \frac{\tau_{ij}(t) \cdot {\eta_{ij}}^{\beta}}{\sum_{l \in J_i^k} \tau_{il}(t) \cdot {\eta_{il}}^{\beta}} \quad (3.7)$$

En fonction du paramètre q_0, il y a donc deux comportements possibles : si $q > q_0$ le choix se fait de la même façon que pour l'algorithme *AS*, et le système tend à effectuer une diversification ; si $q \leq q_0$, le système tend au contraire vers une intensification. En effet,

pour $q \leq q_0$, l'algorithme exploite davantage l'information récoltée par le système, il ne peut pas choisir un trajet non exploré.

2. La gestion des pistes est séparée en deux niveaux : une mise à jour locale et une mise à jour globale. Chaque fourmi dépose une piste lors de la mise à jour locale :

$$\tau_{ij}(t+1) \leftarrow (1-\rho)\tau_{ij}(t) + \rho\tau_0 \tag{3.8}$$

où τ_0 est la valeur initiale de la piste. A chaque passage, les arêtes visitées voient leur quantité de phéromone diminuer, ce qui favorise la diversification par la prise en compte des trajets non explorés. A chaque itération, la mise à jour globale s'effectue comme suit :

$$\tau_{ij}(t+1) \leftarrow (1-\rho)\tau_{ij}(t) + \rho\Delta\tau_{ij}(t) \tag{3.9}$$

où les arêtes (i,j) appartiennent au meilleur tour T^+ de longueur L^+ et où $\Delta\tau_{ij}(t) = \frac{1}{L^+}$. Ici, seule la meilleure piste est donc mise à jour, ce qui participe à une intensification par sélection de la meilleure solution.

3. Le système utilise une liste de candidats. Cette liste stocke pour chaque ville ces plus proches voisines, classées par distances croissantes. Une fourmi ne prendra en compte une arête vers une ville en dehors de la liste que si celle-ci a déjà été explorée. Concrètement, si toutes les arêtes ont déjà été visitées dans la liste des candidats, le choix se fera en fonction de la règle (3.7), sinon c'est la plus proche des villes non visitées qui sera choisie.

3.4.5 *ACS-3-opt*

Cette variante est une hybridation entre le ACS et la recherche locale de type 3-opt [DOR97]. Ici, la recherche locale est lancée pour améliorer les solutions trouvées par les fourmis (et donc les ramener à l'optimum local le plus proche).

3.4.6 *Max-Min Ant System (MMAS)*

Cette variante (notée MMAS) est fondée sur l'algorithme *AS* et présente quelques différences notables [STU00] :

1. Seule la meilleure fourmi met à jour une piste de phéromone.
2. Les valeurs des pistes sont bornées par τ_{min} et τ_{max} .
3. Les pistes sont initialisées à la valeur maximum τ_{max} .
4. La mise à jour des pistes se fait de façon proportionnelle, les pistes les plus fortes sont moins renforcées que les plus faibles.
5. Une réinitialisation des pistes peut être effectuée.

L'équation (3.5) prend la nouvelle forme suivante :

$$\tau_{ij}(t+1) \leftarrow (1-\rho)\tau_{ij}(t) + \rho\Delta\tau_{ij}^{*}(t) \tag{3.10}$$

où $\Delta\tau_{ij}^{*}$ est la valeur de mise à jour du phéromone définie par :

$$\Delta\tau_{ij}^{*}(t) = \begin{cases} \frac{1}{L_*} & si\ la\ meilleure\ fourmi\ utlise\ l'arc\ (i,j) dans\ son\ tour \\ 0 & sinon \end{cases} \tag{3.11}$$

L_* est la longueur du tour de la meilleure fourmi. Il peut être le meilleur tour de l'itération actuelle ou la meilleure solution obtenue dès le début de l'exécution de l'algorithme, ou encore une combinaison des deux.

Concernant les valeurs limites minimale et maximale permises de la phéromone, respectivement τ_{min} et τ_{max}, sont choisies expérimentalement selon le problème considéré.

Le processus de mise à jour de la phéromone est conclu par la vérification que toutes les valeurs de phéromone entre les limites imposées :

$$\tau_{ij}(t) = \begin{cases} \tau_{min} & si\ \tau_{ij} < \tau_{min} \\ \tau_{ij} & si\ \tau_{max} \leq \tau_{ij} \leq \tau_{min} \\ \tau_{max} & si\ \tau_{ij} > \tau_{\max} \end{cases} \tag{3.12}$$

L'algorithme *MMAS* a donné des meilleurs résultats par rapport à l'algorithme de base *AS*.

3.5 Dilemme intensification/diversification

Le processus de diversification et d'intensification est analogue à la politique d'exploration et exploitation dans l'apprentissage par renforcement. Par intensification, on entend l'exploitation de l'information accumulée par le système à un moment donné. Par contre, la diversification est l'exploration de régions de l'espace de recherche imparfaitement prises en compte. Bien souvent, il s'agit de choisir où et quand « injecter de l'aléatoire » dans le système (diversification) et/ou améliorer une solution (intensification).

Dans les algorithmes de type ACO, comme dans la plupart des cas, il existe plusieurs façons de gérer l'emploi de ces deux facettes des métaheuristiques d'optimisation. La plus évidente passe par le réglage via les deux paramètres α et β, qui déterminent l'influence relative des pistes de phéromone et de l'information heuristique. Plus la valeur de α sera élevée, plus l'intensification sera importante, car plus les pistes auront une influence sur le choix des fourmis. A l'inverse, plus α sera faible, plus la diversification sera forte, car les

fourmis éviteront les pistes. Le paramètre β agit de façon similaire. On doit donc gérer conjointement les deux paramètres pour régler ces aspects.

Ce choix diversification/intensification peut s'effectuer de manière statique avant le lancement de l'algorithme, en utilisant une connaissance à priori du problème, ou de manière dynamique, en laissant le système décider du meilleur réglage. Deux approches sont possibles : un réglage par les paramètres ou l'introduction de nouveaux processus. Dans ces algorithmes fondés en grande partie sur l'utilisation de l'auto-organisation, ces deux approches peuvent être équivalentes, et un changement de paramètre peut induire un comportement complètement différent du système, au niveau global.

3.6 Conclusion

Dans ce chapitre, nous avons présenté les fondements théoriques essentiels pour comprendre le principe de l‘optimisation par colonies de fourmis. Nous avons expliqué les origines de l’inspiration biologique de cette approche, et nous avons présenté son modèle mathématique de base. On a constaté que, dans le modèle mathématique, les fourmis artificielles sont dotées de capacités extranaturelles pour s’adapter à la résolution des problèmes d’optimisation difficiles. Nous avons aussi décrit l’ensemble des algorithmes d’optimisation par colonies de fourmis. Ces algorithmes diffèrent essentiellement au niveau de la règle de mise à jour de la phéromone et la règle de sélection heuristique.

L’optimisation par colonies de fourmis a été appliquée avec succès aux différents problèmes d’optimisation. Dans le cadre de cette thèse, nous verrons dans le sixième chapitre comment adapter l’optimisation par colonies de fourmis aux problèmes de réglage des contrôleurs flous.

Chapitre 4

Contrôleurs PID flous typiques

4.1 Introduction

Le choix des variables de contrôle est essentiel pour caractériser un contrôleur flou. En particulier, les entrées d'un contrôleur flou sont en fonction de l'erreur entre la consigne et la sortie du système. Ce choix peut être justifié par une analogie aux régulateurs PID classiques, et aussi par l'inaccessibilité à tous les états du système. Généralement, les entrées d'un contrôleur flou sont l'erreur et la variation de l'erreur et les contrôleurs flous construits sont de type-PI ou de type-PD suivant la sortie dans la base des règles ; si la sortie est le signal de commande il est dit contrôleur PD flou (PDF) et si la sortie est la variation de la commande il est dit contrôleur PI flou (PIF) [JAN07, YAG94]. Un contrôleur flou type-PID (PIDF) peut être construit par l'introduction d'une troisième variable d'entée en plus de l'erreur et de sa variation qui peut être la somme des erreurs ou le taux de changement de l'erreur [YAG94]. Dans le premier, la loi de commande est absolue et dans le deuxième la loi de commande est incrémentale. D'autres alternatives pour construire des lois de commande PIDF sont proposées dans la littérature par la combinaison des deux actions PI et PD ; PI+D [PIV98], PD+I [KIM02, TAN01a], PI+PD [LU04, SAN02], PI-PD [ROY04, VEE04], I-PD [CHA06] ou P+ID [LI01].

L'inconvénient majeur des contrôleurs flous est le grand nombre de paramètres à régler. En revanche, les contrôleurs PID classiques ont uniquement au maximum trois paramètres qui peuvent être réglés facilement par tâtonnement ou par des méthodes de réglage classiques, telles que celles de Ziegler-Nichols [AST95]. Cependant, la comparaison entre les contrôleurs classiques et les contrôleurs flous peut faciliter l'extension des méthodes de réglage existantes au cas des contrôleurs flous.

Plusieurs travaux de recherche ont été consacrés à l'étude des structures analytiques des contrôleurs flous et leurs comparaisons avec les contrôleurs classiques. Dans [MOH08], les auteurs ont étudié le plus simple des contrôleurs PDF avec deux fonctions d'appartenance triangulaires en utilisant différentes méthodes d'inférence. Les auteurs montrent la supériorité des contrôleurs PDFs sur les PD classiques à travers des exemples de simulation. Dans [MOO95], les auteurs montrent que pour un contrôleur flou avec une conception et un choix particuliers de ses paramètres, il existe un PI classique dont la sortie est identique à celle du contrôleur flou. Ils utilisent des fonctions d'appartenance triangulaires équidistantes pour les entrées et la sortie, des règles floues linéaires, le produit algébrique comme opérateur d'implication floue, et la moyenne pondérée comme opérateur de défuzzification. Dans [MUZ95], les auteurs montrent que les PID classiques sont un cas particulier de la commande floue; les PID classiques peuvent être réalisés par la méthode d'inférence produit-somme-centre de gravité. Ils utilisent des fonctions d'appartenance triangulaires pour les entrées et pour la sortie des singletons dont les valeurs sont en fonction des valeurs limites des univers de discours des entrées. Pour montrer leurs résultats, ils utilisent un cas particulier : un PDF avec quatre règles floues et après ils généralisent ce résultat dans le cas d'un nombre de règles quelconque sans faire de calcul. Les paramètres des contrôleurs obtenus n'ont aucun sens physique ce qui rend leur réglage une tache très difficile tant qu'on ne peut pas introduire de la connaissance dans sa structure. Dans [DIN03], les auteurs ont montré que des contrôleurs flous PID typiques avec des fonctions d'appartenance triangulaires ou trapézoïdales pour les entrées, des règles floues linéaires, l'opérateur d'implication de Zadeh et le barycentre comme opérateur de défuzzification, sont équivalents à des contrôleurs PID non linéaires. Dans [JAN07], l'auteur a noté qu'un contrôleur flou peut être linéaire ou non linéaire selon le choix de ses paramètres de conception. L'auteur a montré aussi qu'avec un choix approprié des paramètres de conception, les contrôleurs PI, PD et PI+D flous peuvent être approchés respectivement par des contrôleurs PI, PD et PID linéaires.

Dans ce chapitre, nous allons démontrer mathématiquement et d'une manière claire, qu'avec un choix particulier de leurs paramètres de conception, des contrôleurs PID flous de type Takagi-Sugeno (TS) sont équivalents aux contrôleurs classiques. Ces contrôleurs utilisent un nombre quelconque de fonctions d'appartenance triangulaires uniformément distribuées sur des univers de discours symétriques pour les entrées, et des singletons uniformément distribués sur un univers de discours symétrique pour la sortie, Les contrôleurs flous ainsi construits sont très pratiques. Du fait d'une part, qu'ils sont comparables aux contrôleurs classiques par leurs structures analytiques, et d'autre part ils ont des paramètres interprétables physiquement et présentent la faculté d'introduire des connaissances à priori vues leurs conceptions de base.

4.2 Contrôleurs PI et PD flous

Les contrôleurs PI et PD classiques sont définies soit par des lois discrètes absolues :

$$u_{PD}(n) = K_P e(n) + K_D \Delta e(n) \tag{4.1}$$

$$u_{PI}(n) = K_P e(n) + K_I \sum e(n) \tag{4.2}$$

ou par des lois incrémentales données par :

$$\Delta u_{PD}(n) = K_P \Delta e(n) + K_D \Delta^2 e(n) \tag{4.3}$$

$$\Delta u_{PI}(n) = K_P \Delta e(n) + K_I e(n) \tag{4.4}$$

avec :

$$u_x(n) = u_x(n-1) + \Delta u_x(n) \tag{4.5}$$

K_P est le gain de l'action proportionnelle, K_I est le gain de l'action intégrale, K_D est le gain de l'action de dérivation, u_x est le signal de commande, Δu_x est la variation de la commande et n est la contraction de nT où n est un entier positif et T le pas d'échantillonnage.

L'erreur e, la variation de l'erreur Δe, la somme des erreurs $\sum e$ et la variation de vitesse du changement de l'erreur $\Delta^2 e$, sont exprimées comme suit :

$$e(n) = y_r(n) - y(n) \tag{4.6}$$

$$\Delta e(n) = e(n) - e(n-1) \tag{4.7}$$

$$\sum e(n) = \sum_{i=1}^{n} e(i) \tag{4.8}$$

$$\Delta^2 e(n) = \Delta e(n) - \Delta e(n-1) \tag{4.9}$$

avec : y_r est le signal de référence et y est la sortie du système.

Les lois de commande PIF et PDF sont données par :

$$u_{PIF}(n) = f(e(n), \sum e(n)) \tag{4.10}$$

$$u_{PDF}(n) = f(e(n), \Delta e(n)) \tag{4.11}$$

ou par :

$$\Delta u_{PIF}(n) = f(e(n), \Delta e(n)) \tag{4.12}$$

$$\Delta u_{PDF}(n) = f(\Delta e(n), \Delta^2 e(n)) \tag{4.13}$$

avec :

$$u_x(n) = u_x(n-1) + \Delta u_x(n) \tag{4.14}$$

f est une fonction du système d'inférence floue.

Dans ce qui suit, nous étudions une classe particulière de contrôleurs PIF et PDF.

4.2.1 Paramètres de conception et mode d'inférence floue

4.2.1.1 Variables de fuzzification

On prend des univers de discours symétriques pour chacune des variables d'entrée et de sortie : [-E, E] pour l'erreur, [-D, D] pour la variation de l'erreur et [-U, U] pour la commande. Pour chacune des variables d'entrée, on définit $2m+1$ fonctions d'appartenance triangulaires uniformément distribuées sur leurs univers de discours respectifs et pour la sortie on définit $4m+1$ singletons uniformément distants (voir figure 4.1), avec *m* un entier positif et :

$$\alpha = \frac{E}{m}, \beta = \frac{D}{m}, \lambda = \frac{U}{2m} \tag{4.15}$$

Etant donné les variables d'entrée e et Δe avec : $i\alpha \leq e \leq (i+1)\alpha$ et $j\beta \leq \Delta e \leq (j+1)\beta$ où : $-m \leq i, j, k \leq$m, seules les fonctions d'appartenance : E_i et E_{i+1} pour e, et D_i et D_{i+1} pour Δe seront activées avec un degré d'appartenance différent de zéro.

La figure 4.2 montre les fonctions d'appartenance activées dont les degrés d'appartenance correspondants sont exprimés comme suit :

$$\mu_{E_i}(e(n)) = \frac{-1}{(i+1)\alpha - i\alpha} e(n) + \frac{(i+1)\alpha}{(i+1)\alpha - i\alpha} = \frac{-1}{\alpha} e(n) + i + 1 \tag{4.16}$$

$$\mu_{E_{i+1}}(e(n)) = \frac{1}{(i+1)\alpha - j\alpha} e(n) - \frac{i\alpha}{(i+1)\alpha - i\alpha} = \frac{1}{\alpha} e(n) - i \tag{4.17}$$

Et de la même manière on obtient :

$$\mu_{D_j}(\Delta e(n)) = \frac{-1}{\beta} \Delta e(n) + j + 1 \tag{4.18}$$

$$\mu_{D_{j+1}}(\Delta e(n)) = \frac{1}{\beta} \Delta e(n) - j \tag{4.19}$$

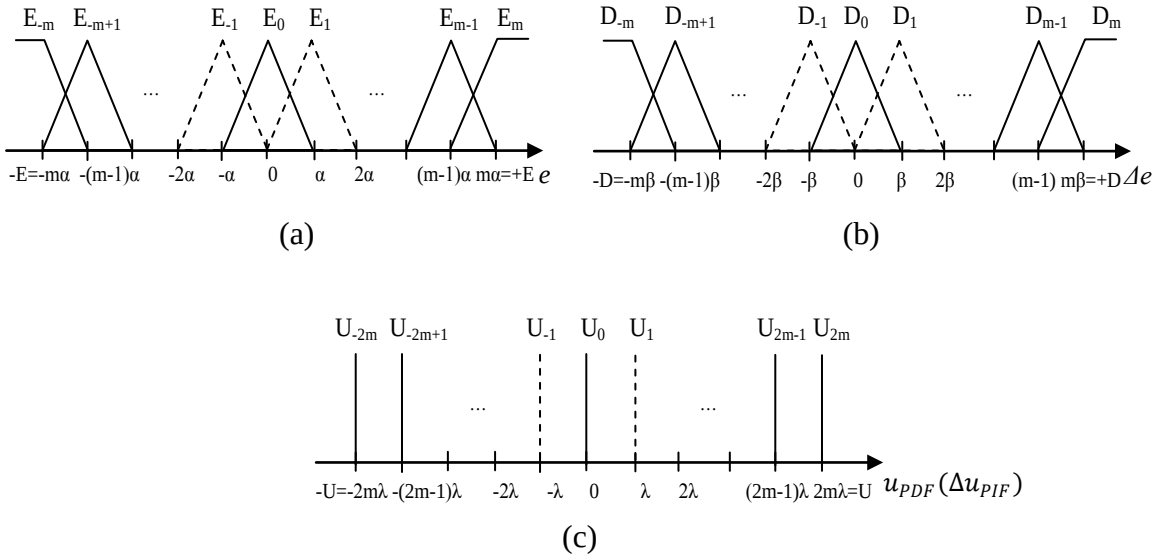


Figure 4.1 : Variables de fuzzification du PDF (PIF). (a) Fonctions d'appartenance pour e . (b) Fonctions d'appartenance pour Δe. (c) Singletons pour $u_{PDF}(\Delta u_{PIF})$

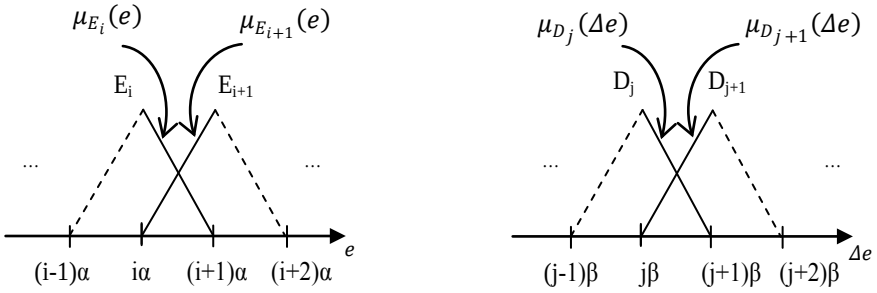


Figure 4.2 : Les fonctions d'appartenance activées

4.2.1.2 Base de règles floues

La base de règles est présentée dans le tableau 4.1, c'est une collection de règles de la forme:

$$\text{Règle } ij\text{:}Si\ e\ est\ E_i\ et\ \Delta e\ est\ D_j\ Alors\ O\ est\ U_{i+j} \tag{4.20}$$

avec : $-m \leq i, j \leq m$

Les indices positifs, négatifs et nuls pour les entrées et la sortie font respectivement référence aux termes "*Positif* (P_l)" "*Négatif* (N_l)" et "*Zero* (Z). Les entrées et la sortie sont donc interprétées symboliquement comme suit :

$$E_l, D_l, U_l = \begin{cases} N_{-l} & si \quad l < 0 \\ Z & si \quad l = 0 \\ P_l & si \quad l > 0 \end{cases} \tag{4.21}$$

Ceci conduit à la base de règles symbolique donnée par le tableau 4.2.

	Δe						
e	D_{-m}	D_{-m+1}	…	D_0	…	D_{m-1}	D_m
E_{-m}	U_{-2m}	U_{-2m+1}	…	U_{-m}	…	U	U_0
E_{-m+1}	U_{-2m+1}	U_{-2m+2}	…	U_{-m+1}	…	U_0	U_1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
E_0	U_{-m}	U_{-m+1}	…	U_0	…	U_{m-1}	U_m
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
E_{m-1}	U_1	U_0	…	U_{m-1}	…	U_{2m-2}	U_{2m-1}
E_m	U_0	U_1	…	U_m	…	U_{2m-1}	U_{2m}

Tableau 4.1 : Base de règles floues

	Δe						
e	N_m	N_{m+1}	…	Z	…	P_{m-1}	P_m
N_m	N_{2m}	N_{2m-1}	…	N_m	…	N_1	Z
N_{m+1}	N_{2m-1}	N_{2m-2}	…	N_{m-1}	…	Z	P_1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Z	N_m	N_{m-1}	…	Z	…	P_{m-1}	P_m
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
P_{m-1}	N_1	Z	…	P_{m-1}	…	P_{2m-2}	P_{2m-1}
P_m	Z	P_1	…	P_m	…	P_{2m-1}	P_{2m}

Tableau 4.2: Base de règles symbolique

4.2.1.3 Défuzzification

La commande inférée par la règle de la somme pondérée est :

$$O(n) = \frac{\sum_{i=-m,j=-m}^{m,m} v_{i,j} U_{i+j}}{\sum_{i=-m,j=-m}^{m,m} v_{i,j}} \tag{4.22}$$

où: $O \in \{u_{PDF}, \Delta u_{PIF}\}$ et $v_{i,j}$ la valeur de vérité de la règle ij calculée par la méthode du produit algébrique donné par :

$$v_{i,j} = \mu_{E_i}(e(n)).\mu_{D_j}(\Delta e(n)) \tag{4.23}$$

avec la condition :

$$\sum_{i=-m,j=-m}^{m,m} v_{i,j} = 1 \tag{4.24}$$

4.2.2 Structure analytique

Nous allons calculer analytiquement la commande PIF (PDF) qui utilise pour les variables d'entrée des fonctions d'appartenance triangulaires équidistantes, pour la commande des singletons équidistants, le produit algébrique pour l'opération d'implication floue et la moyenne pondérée pour la défuzzification [BOU08c].

Etant donné des variables d'entrée; $i\alpha \leq e \leq (i+1)\alpha$ et $j\beta \leq \Delta e \leq (j+1)\beta$, avec: $-m \leq i,j \leq m$, il y a toujours uniquement quatres règles floues qui sont activées à la fois avec un degré de vérité non nul qui sont exprimées comme suit :

- $SI\ e\ est\ E_i\ et\ \Delta e\ est\ D_j\ ALORS\ O\ est\ U_{i+j}$
- $SI\ e\ est\ E_i\ et\ \Delta e\ est\ D_{j+1}\ ALORS\ O\ est\ U_{i+j+1}$
- $SI\ e\ est\ E_{i+1}\ et\ \Delta e\ est\ D_j\ ALORS\ O\ est\ U_{i+j+1}$
- $SI\ e\ est\ E_{i+1}\ et\ \Delta e\ est\ D_{j+1}\ ALORS\ O\ est\ U_{i+j+2}$

La commande inférée est calculée en utilisant la relation (4.20) :

$$O(n) = v_{i,j}U_{i+j} + v_{i,j+1}U_{i+j+1} + v_{i+1,j}U_{i+j+1} + v_{i+1,j+1}U_{i+j+2}$$

$$= \mu_{A_i}(e(n))\mu_{B_j}(\Delta e(n))U_{i+j} + \mu_{A_i}(e(n))\mu_{B_{j+1}}(\Delta e(n))U_{i+j+1}$$

$$+ \mu_{A_{i+1}}(e(n))\mu_{B_j}(\Delta e(kn))U_{i+j+1} + \mu_{A_{i+1}}(e(n))\mu_{B_{i+1}}(\Delta e(n))U_{i+j+2}$$

$$= \left[\frac{-1}{\alpha}e + (i+1)\right]\left[\frac{-1}{\beta}\Delta e(n) + (j+1)\right](i+j)\lambda + \left[\frac{-1}{\alpha}e(n) + (i+1)\right]\left[\frac{1}{\beta}\Delta e(n) - j\right](i+$$

$$j+1)\lambda + \left[\frac{1}{\alpha}e(n) - i\right]\left[\frac{-1}{\beta}\Delta e(n) + (j+1)\right](i+j+1)\lambda + \left[\frac{1}{\alpha}e(n) - i\right]\left[\frac{1}{\beta}\Delta e(n) -$$

$$j\right](i+j+2)\lambda$$

$$=$$

$$\left[\frac{1}{\alpha\beta}e(n)\Delta e(n) - \frac{(j+1)}{\alpha}e(n) - \frac{(i+1)}{\beta}\Delta e(n) + (i+1)(j+1)\right](i+j)\lambda + \left[\frac{-1}{\alpha\beta}e(n)\Delta e(n) +\right.$$

$$\left.\frac{j}{\alpha}e(n) + \frac{(i+1)}{\beta}\Delta e(n) - (i+1)j\right](i+j+1)\lambda + \left[\frac{-1}{\alpha\beta}e(n)\Delta e(n) + \frac{(j+1)}{\alpha}e(n) + \frac{i}{\beta}\Delta e(n) -\right.$$

$$\left.i(j+1)\right](i+j+1)\lambda + \left[\frac{1}{\alpha\beta}e(n)\Delta e(n) - \frac{j}{\alpha}e(k) - \frac{i}{\beta}\Delta e(n) + ij\right](i+j+2)\lambda$$

$$= [-(j+1)(i+j) + j(i+j+1) + (j+1)(i+j+1) - j(i+j+2)]\frac{\lambda}{\alpha} + [-(i+1)(i+j) + (i+1)(i+j+1) + i(i+j+1) - i(i+j+2)]\frac{\lambda}{\beta}\Delta e(n) + [(i+j) - (i+j+1) - (i+j+1) + (i+j+2)]\frac{\lambda}{\alpha\beta}e(n)\Delta e(n) + [(i+1)(j+1)(\mathrm{i}+\mathrm{j}) - j(i+1)(\mathrm{i}+\mathrm{j}+1) - i(j+1)(\mathrm{i}+\mathrm{j}+1) + ij(\mathrm{i}+\mathrm{j}+2)]\lambda$$

$$= [-ij - j^2 - i - j + ij + j^2 + j + ij + j^2 + j + i + j + 1 - ij - j^2 - 2j]\frac{\lambda}{\alpha}e(n) + [-i^2 - ij - i - j + i^2 + ij + i + i + j + 1 + i^2 + ij + +i - i^2 - ij - 2i]\frac{\lambda}{\beta}\Delta e(n) + [i + j - i - j - 1 - i - j - 1 + i + j + 2]\frac{\lambda}{\alpha\beta}e(n)\Delta e(n) + [i^2j + i^2 + ij + i + ij^2 + j^2 + ij + j - i^2j - ij^2 - ij - ij - j^2 - j - i^2j - ij^2 - ij - i^2 - ij - i + i^2j + ij^2 + 2ij]\lambda$$

$$= (1)\frac{\lambda}{\alpha}e(n) + (1)\frac{\lambda}{\beta}\Delta e(n) + (0)\frac{\lambda}{\alpha\beta}e(n)\Delta e(n) + (0)\lambda$$

$$= \frac{\lambda}{\alpha}e(n) + \frac{\lambda}{\beta}\Delta e(n)$$

Alors :

$$O(n) = \frac{\lambda}{\alpha}e(n) + \frac{\lambda}{\beta}\Delta e(n) \quad (4.25)$$

Quant il s'agit d'un PDF :

$$u_{PDF}(n) = \frac{\lambda}{\alpha}e(n) + \frac{\lambda}{\beta}\Delta e(n) \quad (4.26)$$

Remplaçons (4.15) dans (4.25), on obtient :

$$u_{PDF}(n) = \frac{U}{2E}e(n) + \frac{U}{2D}\Delta e(n) \quad (4.27)$$

Et de la même manière, quand il s'agit d'un PIF, la variation de la commande est :

$$\Delta u_{PIF}(n) = \frac{\lambda}{\alpha}e(n) + \frac{\lambda}{\beta}\Delta e(n) \quad (4.28)$$

Remplaçons (4.15) dans (4.28), on obtient :

$$\Delta u_{PIF}(n) = \frac{U}{2E}e(n) + \frac{U}{2D}\Delta e(n) \quad (4.29)$$

Donc la sortie du contrôleur PIF :

$$u_{PIF}(n) = \frac{U}{2D}e(n) + \frac{U}{2E}\sum e(n) \quad (4.30)$$

On constate des équations (4.26) et (4.1), que les contrôleurs typiques PDF sont identiques aux contrôleurs PD classiques avec les paramètres suivants :

$$K_P = \frac{U}{2E} \quad et\ K_D = \frac{U}{2D} \quad (4.31)$$

Et à partir des équations (4.29) et (4.2), on constate que les contrôleurs typiques PIF présentés sont identiques aux contrôleurs PI classiques avec les paramètres suivants :

$$K_P = \frac{U}{2D},\ K_I = \frac{U}{2E} \tag{4.32}$$

Il en résulte que le problème de réglage des paramètres K_P, K_D et K_I devient un problème de réglage des paramètres *E*, *D* et *U*.

4.3 Contrôleur PID flou

La loi de commande PID classique discrète est définie soit par une loi absolue donnée par :

$$u_{PID}(n) = K_P e(n) + K_I \sum e(n) + K_D \Delta e(n) \tag{4.33}$$

ou par une loi incrémentale donnée par :

$$\Delta u_{PID}(n) = K_P \Delta e(n) + K_I e(n) + K_D \Delta^2 e(n) \tag{4.34}$$

La loi de commande PIDF est donnée par :

$$u_{PIDF}(n) = f(e(n), \Delta e(n), \sum e(n)) \tag{4 .35}$$

ou par :

$$\Delta u_{PIDF}(n) = f(e(n), \Delta e(n), \Delta^2 e(n)) \tag{4.36}$$

f est la fonction du système d'inférence floue, u_x est la sortie du contrôleur flou et Δu_x est la variation de la commande.

4.3.1 Paramètres de conception et mode inférence floue

4.3.1.1 Variables de fuzzification

On prend des univers de discours symétriques pour chacune des variables d'entrée et de sortie ; [-*E*, *E*] pour l'erreur, [-*D*, *D*] pour la variation de l'erreur, [-*S*, *S*] pour la somme des erreurs et [-*U*, *U*] pour la commande. Pour chacune des variables d'entrée on définit $2m + 1$ fonctions d'appartenance triangulaires uniformément distribuées sur leur univers de discours respectifs. Pour la sortie on définit $6m + 1$ singletons uniformément distribués sur l'univers de discours (voir figure 4.3) où m est un entier positif et :

$$\alpha = \frac{E}{m}, \beta = \frac{D}{m}, \gamma = \frac{S}{m}, \lambda = \frac{U}{3m} \tag{4.37}$$

Etant donné les variables d'entrée e, Δe, et $\sum e$ avec $i\alpha \le e \le (i+1)\alpha$, $j\beta \le \Delta e \le (j+1)\beta$ et $k\gamma \le \sum e \le (k+1)\gamma$ où : $-m \le i, j, k \le$m, seules les fonctions d'appartenance : E_i et E_{i+1} pour e, D_iet D_{i+1} pour Δe et , S_j et S_{j+1} pour $\sum e$ seront activées avec un degré d'appartenance différent de zéro.

La figure 4.4 montre les fonctions d'appartenance activées et les degrés d'appartenance correspondants qui sont exprimés comme suit :

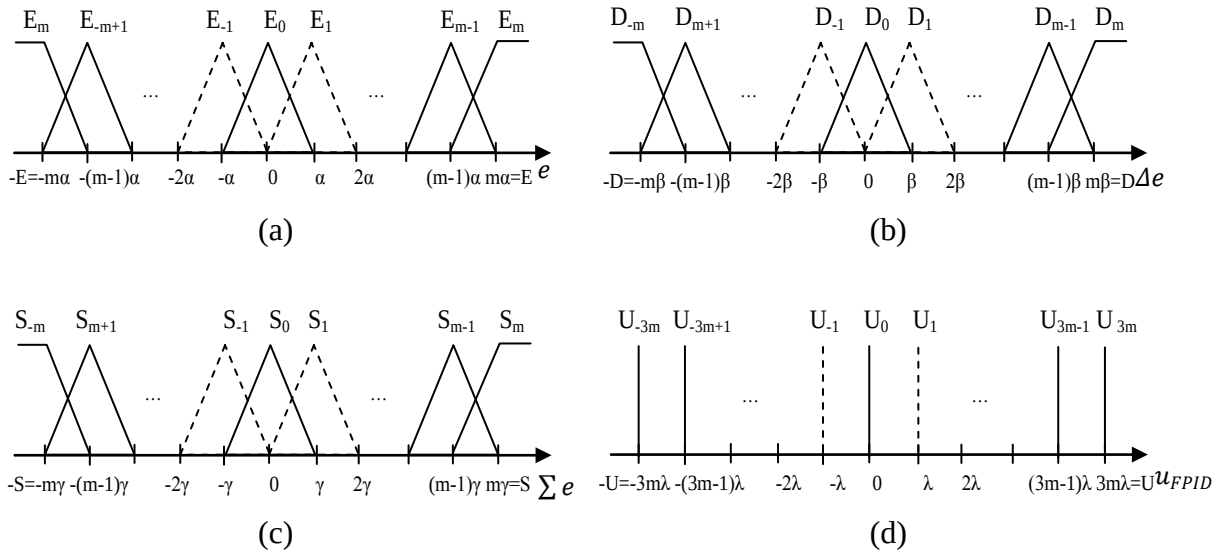


Figure 4.3 : Variables de fuzzification du PIDF. (a) Fonctions d'appartenance pour e. (b) Fonctions d'appartenance pour Δe. (c) Fonctions d'appartenance pour $\sum e$. (d) Singletons pour u_{PIDF}

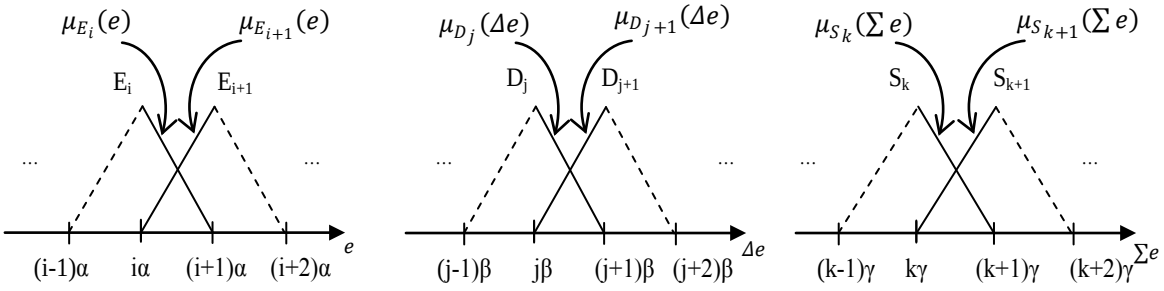


Figure 4.4 : Les fonctions d'appartenance activées

$$\mu_{E_i}(e(n)) = \frac{-1}{\alpha}e(n) + i + 1 \tag{4.38}$$

$$\mu_{E_{i+1}}(e(n)) = \frac{1}{\alpha}e(n) - i \tag{4.39}$$

$$\mu_{D_j}(\Delta e(n)) = \frac{-1}{\beta}\Delta e(n) + j + 1 \tag{4.40}$$

$$\mu_{D_{j+1}}(\Delta e(n)) = \frac{1}{\beta}\Delta e(n) - j \tag{4.41}$$

$$\mu_{S_k}(\sum e(n)) = \frac{1}{\gamma}\sum e(n) + k + 1 \tag{4.42}$$

$$\mu_{S_{k+1}}(\sum e(n)) = \frac{1}{\gamma}\sum e(n) - k \tag{4.43}$$

4.3.1.2 Base de règles floues

On propose ici une base de règles floues construite par une collection de règles de la forme :

Règle ijk : $Si\ e\ est\ E_i\ et\ \Delta e\ est\ D_j\ et\ \Delta e\ est\ S_k Alors\ u_{PIDF}\ est\ U_{i+j+k}$ (4.44)

avec: $-m \leq i,j,k \leq$m.

4.3.1.3 Défuzzification

La commande inférée est déterminée par la moyenne pondérée donné par :

$$u_{FPID}(n) = \frac{\sum_{i=-m,j=-m,k=-m}^{m,m,m} v_{i,j,k} U_{i+j+k}}{\sum_{i=-m,j=-m,k=-m}^{m,m,m} v_{i,j,k}} \quad (4.45)$$

avec :

$$\sum_{i=-m,j=-m,k=-m}^{m,m,m} v_{i,j,k} = 1 \quad (4.46)$$

$v_{i,j,k}$ est la valeur de vérité de la règle ijk calculée par le produit algébrique, donné par :

$$v_{i,j,k} = \mu_{E_i}(e)\mu_{D_j}(\Delta e)\mu_{S_k}(\sum e) \quad (4.47)$$

4.3.2 Structure analytique

Dans cette section, déterminons analytiquement l'expression du signal de sortie d'un contrôleur PIDF utilisant des fonctions d'appartenance triangulaires équidistantes pour les variables d'entrée, des singletons équidistants pour la variable de sortie, le produit algébrique pour l'opération d'implication floue et la moyenne pondérée pour la défuzzification [BOU08c]..

Etant donné des variables d'entrée; $i\alpha \leq e \leq (i+1)\alpha$, $j\beta \leq \Delta e \leq (j+1)\beta$ et $k\gamma \leq \sum e \leq (k+1)\gamma$, avec: $-m \leq i,j,k \leq m$, il y a toujours uniquement huit règles floues qui sont activées avec un degré de vérité non nul exprimées comme suit :

- $Si\ e\ est\ E_i\ et\ \Delta e\ est\ D_j\ et\ \sum e\ est\ S_k\ Alors\ u_{PIDF}\ est\ U_{i+j+k}$
- $Si\ e\ est\ E_i\ et\ \Delta e\ est\ D_j\ et\ \sum e\ est\ S_{k+1} Alors\ u_{PIDF}\ est\ U_{i+j+k+1}$
- $Si\ e\ est\ E_i\ et\ \Delta e\ est\ D_{j+1}\ et\ \sum e\ est\ S_k\ Alors\ u_{PIDF}\ est\ U_{i+j+K+1}$
- $Si\ e\ est\ E_i\ et\ \Delta e\ est\ D_{j+1}\ et\ \sum e\ est\ S_{k+1} Alors\ u_{PIDF}\ est\ U_{i+j+k+2}$
- $Si\ e\ est\ E_{i+1}\ et\ \Delta e\ est\ D_j\ et\ \sum e\ est\ S_k\ Alors\ u_{PIDF}\ est\ U_{i+j+k+1}$
- $Si\ e\ est\ E_{i+1}\ et\ \Delta e\ est\ D_j\ et\ \sum e\ est\ S_{k+1}\ Alors\ u_{PIDF}\ est\ U_{i+j+k+2}$

- $SI\ e\ est\ E_{i+1}\ et\ \Delta e\ est\ D_{j+1} et\ \sum e\ est\ S_k\ Alors\ u_{PIDF}\ est\ U_{i+j+k+2}$
- $Si\ e\ est\ E_{i+1}\ et\ \Delta e\ est\ D_{j+1}\ et\ \sum e\ est\ S_{k+1} Alors\ u_{PIDF}\ est\ U_{i+j+k+3}$

La commande inférée est calculée en utilisant la relation (4.45) :

$$u_{FPID} = v_{i,j,k}U_{i+j+k} + v_{i,j,k+1}U_{i+j+k+1} + v_{i,j+1,k}U_{i+j+k+1} + v_{i,j+1,k+1}U_{i+j+k+2} +$$
$$v_{i+1,j,k}U_{i+j+k+1} + v_{i+1,j,k+1}U_{i+j+k+2} + v_{i+1,j+1,k}U_{i+j+k+2} + v_{i+1,j+1,k+1}U_{i+j+k+3}$$
$$u_{FPID} = \mu_{E_i}(e)\mu_{D_j}(\Delta e)\mu_{U_j}(\sum e)U_{i+j+k} + \mu_{E_i}(e)\mu_{D_j}(\Delta e)\mu_{U_{j+1}}(\sum e)U_{i+j+k+1} +$$
$$\mu_{E_i}(e)\mu_{D_{j+1}}(\Delta e)\mu_{U_j}(\sum e)U_{i+j+k+1} + \mu_{E_i}(e)\mu_{D_{i+1}}(\Delta e)\mu_{U_{j+1}}(\sum e)U_{i+j+k+2}$$
$$+\mu_{E_{i+1}}(e)\mu_{D_j}(\Delta e)\mu_{U_j}(\sum e)U_{i+j+k+1} + \mu_{E_{i+1}}(e)\mu_{D_j}(\Delta e)\mu_{UC_{j+1}}(\sum e)U_{i+j+k+2} +$$
$$\mu_{E_{i+1}}(e)\mu_{D_{j+1}}(\Delta e)\mu_{U_j}(\Delta e)U_{i+j+k+2} + \mu_{E_{i+1}}(e)\mu_{D_{i+1}}(\Delta e)\mu_{U_{j+1}}(\sum e)U_{i+j+k+3}$$
$$= \left[\frac{-1}{\alpha}e + (i+1)\right]\left[\frac{-1}{\beta}\Delta e + (j+1)\right]\left[\frac{-1}{\gamma}\sum e + (k+1)\right](i+j+k)\lambda$$
$$+\left[\frac{-1}{\alpha}e + (i+1)\right]\left[\frac{-1}{\beta}\Delta e + (j+1)\right]\left[\frac{1}{\gamma}\sum e - k\right](i+j+k+1)\lambda$$
$$+\left[\frac{-1}{\alpha}e + (i+1)\right]\left[\frac{1}{\beta}\Delta e - j\right]\left[\frac{-1}{\gamma}\sum e + (k+1)\right](i+j+k+1)\lambda$$
$$+\left[\frac{-1}{\alpha}e + (i+1)\right]\left[\frac{1}{\beta}\Delta e - j\right]\left[\frac{1}{\gamma}\sum e - k\right](i+j+k+2)\lambda$$
$$+\left[\frac{1}{\alpha}e - i\right]\left[\frac{-1}{\beta}\Delta e + (j+1)\right]\left[\frac{-1}{\gamma}\sum e + (k+1)\right](i+j+k+1)\lambda$$
$$+\left[\frac{1}{\alpha}e - i\right]\left[\frac{-1}{\beta}\Delta e + (j+1)\right]\left[\frac{1}{\gamma}e\sum e - k\right](i+j+k+2)\lambda$$
$$+\left[\frac{1}{\alpha}e - i\right]\left[\frac{1}{\beta}\Delta e - j\right]\left[\frac{-1}{\gamma}\sum e + (k+1)\right](i+j+k+2)\lambda$$
$$+\left[\frac{1}{\alpha}e - i\right]\left[\frac{1}{\beta}\Delta e - j\right]\left[\frac{1}{\gamma}\sum e - k\right](i+j+k+3)\lambda$$
$$= \left[\frac{1}{\alpha\beta}e\Delta e - \frac{(j+1)}{\alpha}e - \frac{(i+1)}{\beta}\Delta e + (i+1)(j+1)\right]\left[\frac{-1}{\gamma}\sum e + (k+1)\right](i+j+k)\lambda$$
$$+\left[\frac{1}{\alpha\beta}e\Delta e - \frac{(j+1)}{\alpha}e - \frac{(i+1)}{\beta}\Delta e + (i+1)(j+1)\right]\left[\frac{1}{\gamma}\sum e - k\right](i+j+k+1)\lambda$$
$$+\left[\frac{-1}{\alpha\beta}e\Delta e + \frac{j}{\alpha}e + \frac{(i+1)}{\beta}\Delta e - (i+1)j\right]\left[\frac{-1}{\gamma}\sum e + (k+1)\right](i+j+k+1)$$
$$+\left[\frac{-1}{\alpha\beta}e\Delta e + \frac{j}{\alpha}e(k) + \frac{(i+1)}{\beta}\Delta e - (i+1)j\right]\left[\frac{1}{\gamma}\sum e - k\right](i+j+k+2)\lambda$$
$$+\left[\frac{-1}{\alpha\beta}e\Delta e + \frac{(j+1)}{\alpha}e + \frac{i}{\beta}\Delta e - i(j+1)\right]\left[\frac{-1}{\gamma}\sum e + (k+1)\right](i+j+k+1)\lambda$$
$$+\left[\frac{-1}{\alpha\beta}e\Delta e + \frac{(j+1)}{\alpha}e + \frac{i}{\beta}\Delta e - i(j+1)\right]\left[\frac{1}{\gamma}e\sum e - k\right](i+j+k+2)\lambda$$
$$+\left[\frac{1}{\alpha\beta}e\Delta e - \frac{j}{\alpha}e - \frac{i}{\beta}\Delta e + ij\right]\left[\frac{-1}{\gamma}\sum e + (k+1)\right](i+j+k+2)\lambda$$

$$+\left[\frac{1}{\alpha\beta}e\Delta e-\frac{j}{\alpha}e(k)-\frac{i}{\beta}\Delta e+ij\right]\left[\frac{1}{\gamma}\sum e-k\right](i+j+k+3)\lambda$$

$$=\left[\frac{-1}{\alpha\beta\gamma}e\Delta e\sum e+\frac{(j+1)}{\alpha\gamma}e\sum e+\frac{(i+1)}{\beta\gamma}\Delta e\sum e-(i+1)(j+1)\frac{1}{\gamma}\sum e+(k+1)\frac{1}{\alpha\beta}e\Delta e-\right.$$

$$\left.(k+1)\frac{(j+1)}{\alpha}e-(k+1)\frac{(i+1)}{\beta}\Delta e+(i+1)(j+1)(k+1)\right](i+j+k)\lambda$$

$$+\left[\frac{1}{\alpha\beta\gamma}e\Delta e\sum e-\frac{(j+1)}{\alpha\gamma}e\sum e-\frac{(i+1)}{\beta\gamma}\Delta e\sum e+(i+1)(j+1)\frac{1}{\gamma}\sum e-k\frac{1}{\alpha\beta}e\Delta e+k\frac{(j+1)}{\alpha}e+\right.$$

$$\left.k\frac{(i+1)}{\beta}\Delta e-(i+1)(j+1)k\right](i+j+k+1)\lambda$$

$$+\left[\frac{1}{\alpha\beta\gamma}e\Delta e\sum e-\frac{j}{\alpha\gamma}e\sum e-\frac{(i+1)}{\beta\gamma}\Delta e\sum e+(i+1)j\frac{1}{\gamma}\sum e-(k+1)\frac{1}{\alpha\beta}e\Delta e+\right.$$

$$\left.(k+1)\frac{j}{\alpha}e+(k+1)\frac{(i+1)}{\beta}\Delta e-(k+1)(i+1)j\right](i+j+k+1)\lambda$$

$$+\left[\frac{-1}{\alpha\beta\gamma}e\Delta e\sum e+\frac{j}{\alpha\gamma}e\sum e+\frac{(i+1)}{\beta\gamma}\Delta e\sum e-(i+1)j\frac{1}{\gamma}\sum e+k\frac{1}{\alpha\beta}e\Delta e-k\frac{j}{\alpha}e-\right.$$

$$\left.k\frac{(i+1)}{\beta}\Delta e+(i+1)jk\right](i+j+k+2)\lambda$$

$$+\left[\frac{1}{\alpha\beta\gamma}e\Delta e\sum e-\frac{(j+1)}{\alpha\gamma}e\sum e-\frac{i}{\beta\gamma}\Delta e\sum e+i(j+1)\frac{1}{\gamma}\sum e-(k+1)\frac{1}{\alpha\beta}e\Delta e+(k+\right.$$

$$\left.1)\frac{(j+1)}{\alpha}e+(k+1)\frac{i}{\beta}\Delta e-i(j+1)(k+1)\right](i+j+k+1)\lambda$$

$$+\left[\frac{-1}{\alpha\beta\gamma}e\Delta e\sum e+\frac{(j+1)}{\alpha\gamma}e\sum e+\frac{i}{\beta\gamma}\Delta e\sum e-i(j+1)\frac{1}{\gamma}\sum e+k\frac{1}{\alpha\beta}e\Delta e-k\frac{(j+1)}{\alpha}e-\right.$$

$$\left.k\frac{i}{\beta}\Delta e+i(j+1)k\right](i+j+k+2)\lambda$$

$$+\left[\frac{-1}{\alpha\beta\gamma}e\Delta e\sum e+\frac{j}{\alpha\gamma}e\sum e+\frac{i}{\beta\gamma}\Delta e\sum e-ij\frac{1}{\gamma}\sum e+(k+1)\frac{1}{\alpha\beta}e\Delta e-(k+1)\frac{j}{\alpha}e-\right.$$

$$\left.(k+1)\frac{i}{\beta}\Delta e+ij(k+1)\right](i+j+k+2)\lambda$$

$$+\left[\frac{1}{\alpha\beta\gamma}e\Delta e\sum e-\frac{j}{\alpha\gamma}e\sum e-\frac{i}{\beta\gamma}\Delta e\sum e+ij\frac{1}{\gamma}\sum e-k\frac{1}{\alpha\beta}e\Delta e+k\frac{j}{\alpha}e+k\frac{i}{\beta}\Delta e-ijk\right](i+$$

$$j+k+3)\lambda$$

$$=[-(i+j+k)+(i+j+k+1)+(i+j+k+1)-(i+j+k+2)+(i+j+k+$$

$$1)+(i+j+k+2)-(i+j+k+2)+(i+j+k+3)]\frac{\lambda}{\alpha\beta\gamma}e\Delta e\sum e$$

$$+[(k+1)(i+j+k)-k(i+j+k+1)-(k+1)(i+j+k+1)+k(i+j+k+2)-$$

$$(k+1)(i+j+k+1)+k(i+j+k+2)+(k+1)(i+j+k+2)-k(i+j+k+$$

$$3)]\frac{1}{\alpha\beta}e\Delta e$$

$$+[(j+1)(i+j+k)-(j+1)(i+j+k+1)-j(i+j+k+1)+j(i+j+k+2)-$$

$$(j+1)(i+j+k+1)+(j+1)(i+j+k+2)+j(i+j+k+2)-j(i+j+k+$$

$$3)]\frac{1}{\alpha\gamma}e\sum e$$

$$+[(i+1)(i+j+k)-(i+1)(i+j+k+1)-(i+1)(i+j+k+1)+(i+1)(i+j+k+2)-i(i+j+k+1)+i(i+j+k+2)+i(i+j+k+2)-i(i+j+k+3)]\frac{1}{\beta\gamma}\Delta e\sum e$$

$$+[-(k+1)(j+1)(i+j+k)+(k+1)j(i+j+k+1)+k(j+1)(i+j+k+1)-kj(i+j+k+2)+k(j+1)(i+j+k+1)+(k+1)(j+1)(i+j+k+2)\lambda-(k+1)j(i+j+k+2)\lambda+kj(i+j+k+3)]\frac{\lambda}{\alpha}e$$

$$+[-(k+1)(i+1)(i+j+k)+(k+1)i(i+j+k+1)+k(i+1)(i+j+k+1)-ki(i+j+k+2)+k(i+1)(i+j+k+1)+(k+1)(i+1)(i+j+k+2)\lambda-(k+1)i(i+j+k+2)\lambda+ki(i+j+k+3)]\frac{\lambda}{\beta}\Delta e$$

$$+[-(k+1)(i+1)(i+j+k)+(k+1)i(i+j+k+1)+k(i+1)(i+j+k+1)-ki(i+j+k+2)+k(i+1)(i+j+k+1)+(k+1)(i+1)(i+j+k+2)\lambda-(k+1)i(i+j+k+2)\lambda+ki(i+j+k+3)]\frac{\lambda}{\gamma}\sum e$$

$$+[(k+1)(j+1)(i+j+k)-k(j+1)(i+j+k+1)+(k+1)j(i+j+k+1)-kj(i+j+k+2)+k(j+1)(i+j+k+1)+(k+1)(j+1)(i+j+k+2)-(k+1)j(i+j+k+2)+kj(i+j+k+3)]\lambda$$

$$=(0)\frac{\lambda}{\alpha\beta}e\Delta e\sum e+(0)\frac{\lambda}{\alpha\beta}e\Delta e+(0)\frac{\lambda}{\alpha\gamma}e\sum e+(0)\frac{\lambda}{\beta\gamma}\Delta e\sum e+(1)\frac{\lambda}{\alpha}e+(1)\frac{\lambda}{\beta}\Delta e+(1)\frac{\lambda}{\gamma}\sum e+0\lambda$$

$$=\frac{\lambda}{\alpha}e(n)+\frac{\lambda}{\beta}\Delta e(n)+\frac{\lambda}{\gamma}\sum e(n)$$

Alors :

$$u_{PIDF}(n)=\frac{\lambda}{\alpha}e(n)+\frac{\lambda}{\beta}\Delta e(n)+\frac{\lambda}{\gamma}\sum e(n) \tag{4.48}$$

Remplaçons (4.35) dans (4.49), on obtient :

$$u_{PIDF}(n)=\frac{U}{3E}e(n)+\frac{U}{3D}\Delta e(n)+\frac{U}{3S}\sum e(n) \tag{4.49}$$

En comparant l'équation (4.49) à l'équation (4.33), on constate que le contrôleur PIDF est identique à un PID classique dont les paramètres sont définis comme suit :

$$K_P=\frac{U}{3E}\ ,K_D=\frac{U}{3D},\ K_I=\frac{U}{3S} \tag{4.50}$$

Alors, le problème de réglage K_P, K_I et K_D devient un problème de définition des univers de discours *E*, *D*, *S* et *U*. En particulier, quand *U* est fixé à priori, le réglage K_P, K_I et K_D est identique au réglage de $\frac{1}{E}$, $\frac{1}{D}$ et $\frac{1}{S}$ respectivement. Notons ici, qu'on obtient les mêmes résultats en suivant les mêmes étapes dans le cas des contrôleurs flous incrémentaux.

Pour résumer, le tableau 4.3 présente la relation entre les paramètres de réglage des contrôleurs PID classiques et les PID flous linéaires :

Contrôleur	K_P	K_I	K_D
PDF	$\frac{U}{2E}$	-	$\frac{U}{2D}$
PIF	$\frac{U}{2D}$	$\frac{U}{2E}$	-
PIDF	$\frac{U}{3E}$	$\frac{U}{3S}$	$\frac{U}{3D}$

Tableau 4.3 : Relation entre les paramètres des PID classique et les PID flous

Sur la base des résultats précédents, on constate que la loi de commande PID classique n'est qu'un cas particulier de la commande floue. Cependant, la structure des PIDF a l'avantage d'avoir des paramètres physiquement interprétables et une faculté d'introduire des connaissances expertes. Par conséquent, le problème de réglage des PIDF devient une tache moins difficile que celui d'un PID. En plus, les méthodes de réglage classique peuvent être utilisées dans le cas des PID flous linéaires.

Dans la section suivante, nous verrons comment adapter la méthode de la réponse fréquentielle de Ziegler-Nichols [AST95] pour le réglage des PID flous linéaires.

4.4 Réglage des contrôleurs PID flous par Ziegler-Nichols

4.4.1 Rappel

Les contrôleurs PID classiques peuvent être réglés par la méthode de réponse fréquentielle de Ziegler-Nichols. Les étapes de cette méthode sont résumées comme suit :

1. Augmenter le gain proportionnel jusqu'à amener le système à osciller de manière permanente, le gain résultant est le gain critique K_{cr}.
2. Relever la période d'oscillation de la réponse T_{cr}.
3. Calculer les paramètres du contrôleur à l'aide du tableau 4.4.

Contrôleur	K_P	K_I	K_D
P	$0.5K_{cr}$	-	-
PI	$0.45K_{cr}$	$0.54\frac{K_{cr}}{T_{cr}}$	-
PID	$0.6K_{cr}$	$1.2\frac{K_{cr}}{T_{cr}}$	$0.075K_{cr}T_{cr}$

Tableau 4.4 : Réglage des paramètres d'un contrôleur PID classique par la méthode de la réponse fréquentielle de Ziegler-Nichols

4.4.2 Adaptation de la méthode au réglage des PID flous

A partir des deux tableaux 4.3 et 4.4, si l'on fixe à priori un des paramètres d'un contrôleur flou, la méthode de Ziegler-Nichols peut être adapté au réglage contrôleurs PID flous. Si on fixe le paramètre U, le tableau 4.5 donne les règles de Ziegler-Nichols pour le réglage des contrôleurs PID flous linéaires.

Contrôleur	*E*	*S*	*D*
PIF	$0.93\frac{T_{cr}U}{K_{cr}}$	$1.11\frac{U}{K_{cr}}$	-
PIDF	$\frac{U}{1.8\,K_{cr}}$	$\frac{T_{cr}U}{3.6K_{cr}}$	$\frac{U}{0.225K_{cr}T_{cr}}$

Tableau 4.5 : Réglage des paramètres d'un contrôleur PID flou par la méthode de la réponse fréquentielle de Ziegler-Nichols

Exemple :

Soit le système décrit par la fonction de transfert suivante :

$$H(s) = \frac{1}{(s+1)^3}$$

A l'aide de la méthode de Ziegler-Nichols, on veut déterminer les paramètres d'un contrôleur PID flou utilisé pour commander ce système.

On utilise un contrôleur PID flou avec trois fonctions d'appartenance pour chaque variable d'entrée et sept singletons pour la sortie (figure 4.5), et la base de règles donnée par le tableau 4.6.

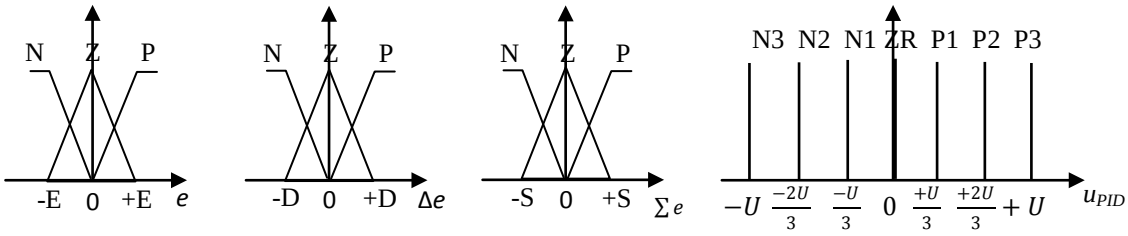


Figure 4.5 : Fonctions d'appartenance

	$\Delta e, \sum e$								
e	N,N	N,Z	N,P	Z,N	Z,Z	Z,P	P,N	P,Z	P,P
N	N3	N2	N1	N2	N1	Z	N1	Z	P1
Z	N2	N1	Z	N1	Z	P1	Z	P1	P2
P	N1	Z	P1	Z	P1	P2	P1	P2	P3

Tableau 4.6 : Base des règles floues

On insère dans le système bouclé un gain proportionnel qu'on augmente graduellement jusqu'à amener le système à une oscillation stable (figure 4.6). Ce gain critique $K_{cr} = 8$, et la période d'oscillation $T_{cr} = 3.75\ s$.

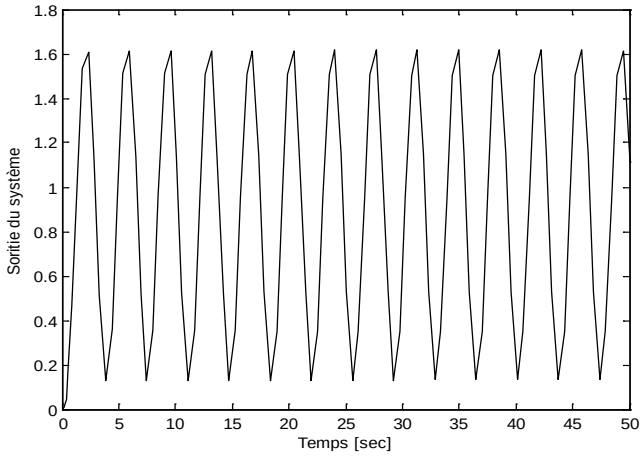


Figure 4.6 : Oscillation de Ziegler-Nichols du système $H(s)$

On fixe la valeur maximale de l'univers de discours de la commande $U = 3$ et on utilise le tableau 4.4 pour déterminer les autres paramètres du contrôleur PID flou, ce qui donne :

$E = 0.208,\ S = 0.391,\ D = 0.444$. La figure 4.7 présente la réponse du système commandé par le contrôleur PID flou.

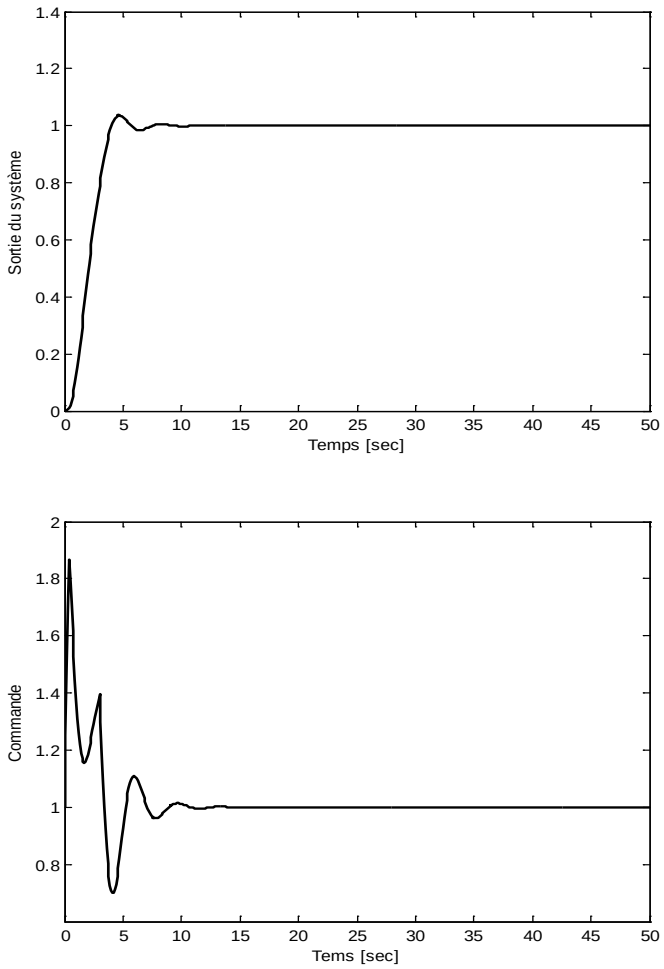


Figure 4.7 : Réponse du système $H(s)$ commandé par le contrôleur PID flou (Réglage par Ziegler-Nichols)

4.5 Conclusion

Dans ce chapitre, nous avons montré mathématiquement d'une manière claire, qu'avec un choix particulier des paramètres de conception d'un contrôleur flou, des contrôleurs PID flous de type Takagi-Sugeno avec un nombre quelconque de fonctions d'appartenance triangulaires uniformément distribuées sur des univers de discours symétriques pour les entrées et des singletons uniformément distribués sur un univers de discours symétriques pour la sortie, sont équivalents aux contrôleurs PID classiques. Les contrôleurs flous ainsi construits sont très pratiques du fait d'une part qu'ils sont comparables aux contrôleurs

classiques par leurs structures analytiques et d'autre part ils ont des paramètres interprétables physiquement vue leur conception de base. L'équivalence entre les PID classiques et les PID flous linéaires nous a permis d'adapter la méthode de la réponse en fréquence de Ziegler-Nichols pour le réglage de ces derniers. La méthode est illustrée par un exemple de simulation.

Chapitre 5

Optimisation des contrôleurs flous en utilisant l'apprentissage par renforcement

5.1 Introduction

Nous avons vu dans le chapitre 4, qu'avec un choix particulier des paramètres de conception des contrôleurs PID flous, ces derniers sont équivalents à des contrôleurs PID classiques par leurs formules analytiques. Mais vue leur conception de base, ces contrôleurs offrent plus de souplesse que leurs homologues classiques, du fait qu'ils sont facilement transposables au cas non linéaire.

Plusieurs méthodes de conception, permettant le réglage des différents paramètres d'un contrôleur flou, ont été proposées dans la littérature telles que : les méthodes de Ziegler-Nichols [JAN07], la supervision floue [ZHA06], méthodes adaptatives [BHA04], algorithmes génétiques [ADA95, KO06, TAN01a, TAN01b, WU07]. Ces méthodes sont capables de générer une solution optimale ou quasi-optimale.

Dans ce chapitre, on propose une méthode de réglage des contrôleurs flous en utilisant l'apprentissage par renforcement. Le problème consiste à la recherche des paramètres optimaux d'un contrôleur flou, les paramètres qui permettent la minimisation d'une fonction de performance du système de commande.

5.2 Rappel de l'algorithme Q-Learning

Nous avons vu dans le chapitre 2 que l'algorithme Q-learning est l'algorithme d'apprentissage par renforcement le plus utilisé en pratique en raison de sa simplicité et la démonstration de sa convergence. On rappelle ici en bref le fonctionnement de l'algorithme. Une valeur Q est assignée à tout couple état-action. La valeur Q reflète la récompense espérée à long terme en prenant l'action correspondante de l'ensemble des actions A à l'état correspondant dans l'espace des états S. Notons par $Q(s_t, a_t)$ la valeur quand l'action a_t est prise dans l'état s_t à l'instant t. Quand une récompense r est obtenue immédiatement après l'application de l'action a_t, la valeur $Q(s_t, a_t)$ est mise à jour comme suit :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \delta\left(r(s_t, a_t) + \gamma \max_{b \in A} Q(s_{t+1}, b) - Q(s_t, a_t)\right) \quad (5.1)$$

où : δ est le pas d'apprentissage et γ est le facteur d'atténuation.

La politique de sélection de l'action par l'agent de l'apprentissage dans l'algorithme du Q-Learning est donnée par la considération du dilemme entre l'exploration et l'exploitation des pairs état-action. Par la suite, la politique d'exploration/exploitation ε-gloutonne est utilisée pour la sélection d'une action :

$$a_t = \begin{cases} arg \max_{b \in A} Q(x_t, b) & q \leq \varepsilon \\ b \in A & q > \varepsilon \end{cases} \quad (5.2)$$

avec q un nombre choisi par une loi aléatoire uniforme sur [0,1], ε coefficient réglable dans [0,1], et b est choisie par une loi aléatoire uniforme dans A.

Après un temps d'apprentissage suffisant, le processus d'apprentissage peut être arrêté et la stratégie gloutonne (*greedy*) sera utilisée pour une exploitation finale des résultats d'apprentissage :

$$a_t = arg \max_{b \in A} Q(s_t, b) \quad (5.3)$$

L'algorithme 5.1 résume les différentes étapes du Q-learning.

Algorithme 5.1 : Algorithme du Q-Learning de base

Initialiser $Q(s_t, a_t)$ arbitrairement
Refaire (Pour chaque épisode)
Initialiser s_t
Répéter (pour chaque pas de l'épisode):
Choisir a_t avec une politique d'exploration/exploitation (exemple : ε-gloutonne)
Appliquer l'action a_t , Recevoir r et s_{t+1}
Mettre à jour les Q-valeur par l'équation 5.1.
$t \leftarrow t + 1$
Jusqu'à s_t final

5.3. Optimisation des contrôleurs PID flous linéaires par Q-learning

Dans cette section nous proposons une méthode de réglage des contrôleurs flous en utilisant l'algorithme d'apprentissage par renforcement Q-learning **[BOU08a].**

5.3.1 Présentation

Sans perte de généralité, considérons le contrôleur PID flou linéaire présenté dans le troisième chapitre. Les paramètres à régler sont uniquement les valeurs maximales des univers de discours des variables d'entrée du contrôleur, *E*, *D*, *S*, et de sortie, *U*. Soit donc le vecteur des paramètres :

$$P = [\, P_1\ P_2\ P_3\ P_4]^{\,t} = [E\ D\ S\ U]^t \tag{5.4}$$

Dans le cas des contrôleurs PID flous non linéaires, le vecteur des paramètres à optimiser est construit par l'ensemble des positions des fonctions d'appartenance des variables d'entrée et/ou les positions des singletons de la variable de sortie.

5.3.1.1 Discrétisation de l'espace de recherche

A chaque paramètre, on associe plusieurs valeurs candidates. Pour chaque paramètre, on a besoin de la valeur minimale et maximale possible: Les valeurs proposées à la candidature sont uniformément distribuées entre ces bornes. Soit P^i_{min} et P^i_{max} les bornes du $i^{ème}$ élément du vecteur de paramètres *P*, voir le tableau 5.1. Par conséquent, on prend :

$$P_{i1} = P^i_{min}\,, P_{i2} = P_{i1} + \frac{P^i_{max} - P^i_{min}}{J-1}, \ldots, P_{iJ} = P^i_{max} \tag{5.5}$$

où J est le nombre de valeurs candidates.

Paramètre	*Min*	*Max*
E	E_{min}	E_{max}
D	D_{min}	D_{max}
S	S_{min}	S_{max}
U	U_{min}	U_{max}

Tableau 5.1 : Définitions des bornes des paramètres

5.3.1.2 Fonction de performance

Sans perte de généralité, nous utilisons la moyenne de la somme des carrés des erreurs évaluée à la fin de la réponse en échelon du système de commande composé par un contrôleur flou et le système à commander. Soit la fonction de performance donnée par :

$$I = \frac{1}{NT - N_0 T} \sum_{k=N_0}^{N} e(k)^2 \tag{5.6}$$

avec : N est le nombre total des échantillons, N_0 le nombre des échantillons transitoires et T le pas d'échantillonnage.

La fonction de performance I définie par l'équation (5.6) prend en compte l'erreur sur l'intervalle de temps $[N_0 T, NT]$. En choisissant $t_0 = N_0 T$ le temps de pic approximatif t_{pic} pendant lequel la réponse indicielle du système en boucle fermé atteint le premier pic.

5.3.1.3 Signal de renforcement

Dans le but de minimiser la fonction du critère de performance I, nous utilisons le signal de renforcement r donné par :

$$r = \begin{cases} -R_1 \; si \; I > I^* \\ +R_2 \; si \; I \leq I^* \end{cases} \tag{5.7}$$

où R_1 et R_2 sont des coefficients positifs et I^* est un seuil de satisfaction.

Le signal r est perçu comme une récompense ou une punition selon que la valeur de la fonction de performance est supérieure ou inférieur au seuil de satisfaction I^*.

5.3.1.3 Fonctionnement de l'algorithme

Pour chaque valeur candidate, on associe une valeur Q qui sera mise à jour par l'algorithme Q-Learning ; a chaque paramètre P_{ij} , on associe une valeur Q_{ij} .

Le processus d'apprentissage consiste donc à déterminer les meilleurs paramètres, ceux qui optimiseront les renforcements futurs. Ainsi, comme les quantités sont initialement

inconnues, le contrôleur flou va explorer et tester plusieurs actions. La phase d'exploration est souvent longue. Bien que, comme les règles et les paramètres du contrôleur sont interprétables physiquement, cette phase peut être considérablement réduite par l'introduction des connaissances dans le contrôleur flou initial.

Les étapes de la méthode sont résumées sur l'algorithme 5.2.

Algorithme 5.2: Réglage des paramètres d'un PID flous par Q-learning

---

Initialiser Q_{ij} arbitrairement, $1 \leq i \leq M, 1 \leq j \leq N$
Observer l'état initial du système s_t.
Pour chaque épisode (réponse à un échelon) **faire**
  Pour chaque paramètre E, D et S
    Choisir une valeur dans la liste des valeurs possibles avec une *PEE*
  Fin Pour
  Répéter
    Calculer l'action a_t par le système d'inférence du PID flou.
    Appliquer l'action a_t et observer la sortie du système s_{t+1}.
    Evaluer la fonction de performance I.
    Recevoir le renforcement r.
    Faire la mise à jour des valeurs Q_{ik} par la relation (5.1)
    $t \leftarrow t + 1$
  J'usqu'à t_{final}
Fin Pour
Retourner les paramètres optimaux par la politique gloutonne (5.3)

5.3.2 Exemple de simulation

Considérons le problème de stabilisation du pendule inversé dans la position verticale pour démontrer l'efficacité de la méthode de réglage proposée. Le modèle dynamique non linéaire du pendule inversé est donné par :

$$\ddot{\theta} = \frac{g(m_p+m_c)\sin\theta-(m_p l\dot{\theta}^2\cos\theta\sin\theta)+\cos\theta u}{l(4/3(m_p+m_c)-m_p\cos^2\theta)} \quad (5.8)$$

avec : θ la position angulaire et u le signal de commande

Dans cet exemple de simulation, les paramètres suivants sont utilisés : $g = 9{,}8\ m/s^2, m_c = 1kg, m_p = 0{,}1kg, l = 0{,}5m.$

Notre objectif est d'optimiser un PID flou vis-à-vis la somme des carrés de l'erreur (5.6) à la fin d'une réponse indicielle du système de commande composé du pendule inversé et d'un contrôleur PID flou.

L'état initiale du pendule est $\theta(0) = 1\,rad$ (pendule suspendu) et l'état désiré $\theta_r = 0\,rad$ (pendule levé en verticale). La vitesse angulaire initiale $\dot{\theta}(0) = 0rad/s$. Le pas d'échantillonnage $T = 0{,}1s$, et la durée d'un épisode est 10s.

On utilise un contrôleur PID flou dont les fonctions d'appartenance pour les entrées et la sortie sont présentées sur la figure 5.1, et la base de règles donnée par le tableau 5.2.

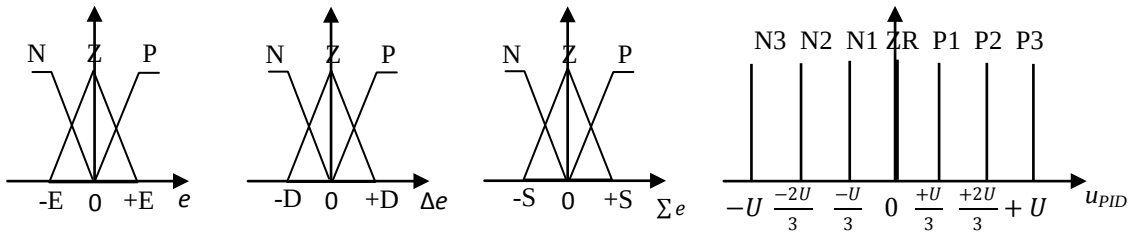


Figure 5.1 : Fonctions d'appartenance

	$\Delta e, \sum e$								
e	N,N	N,Z	N,P	Z,N	Z,Z	Z,P	P,N	P,Z	P,P
N	N3	N2	N1	N2	N1	Z	N1	Z	P1
Z	N2	N1	Z	N1	Z	P1	Z	P1	P2
P	N1	Z	P1	Z	P1	P2	P1	P2	P3

Tableau 5.2 : Base de règle du contrôleur

Le vecteur des paramètres de réglage est :

$$P = [E\ D\ S\ U]^T$$

On prend le signal de renforcement r comme suit :

$$r = \begin{cases} -100\ si\ I > I^* \\ +100\ si\ I \leq I^* \end{cases}, \qquad \text{avec: } I^* = 10^{-3} \tag{5.9}$$

Dans les simulations, la fonction de performance s'étends de $t_0 = 1\ s$ à $t_f = 10\ s$. Les bornes des paramètres sont données dans le tableau 5.3. Pour chaque paramètre, le nombre des valeurs candidates est $J = 20$. Toutes les Q-valeurs sont initialisées à zéro, la probabilité $\varepsilon = 0.9$ dans la politique ε-gloutonne, le pas d'apprentissage $\delta = 0.1$ et le facteur d'atténuation $\gamma = 0.8$.

Paramètre	*Min*	*Max*
E	0.1	1.5
D	0.01	3
S	400	600
U	10	50

Tableau 5.3 : Définition des bornes des paramètres

A cause de la nature pseudo-stochastique de l'algorithme Q-Learning, l'algorithme est exécuté 10 fois avec 100 itérations par exécution. Pour toutes les exécutions, l'algorithme converge à une valeur acceptable de la fonction de performance ($I < I^*$) pendant moins de 100 itérations. Parmi ces résultats, le meilleur avec la fonction de performance minimale est illustré comme suit.

Les figures 5.2 et 5.3 montrent respectivement la sortie commandée (position angulaire) et le signal de commande. L'évolution de la fonction de performance est présentée sur la figure 5.4 et l'évolution des paramètres de réglage est présentée sur les figures 5.5 à 5.8.

L'algorithme converge à la valeur de la fonction de performance : $\mathrm{I} = 5.8385 \times 10^{-6}$ qui correspond aux paramètres présentés dans le tableau 5.4.

Paramètre	*Valeur apprise*
E	1.5
D	0.4821
S	526.3157
U	47,8947

Tableau 4.4 : Les valeurs des paramètres après apprentissage

Il est clair que le nombre de solutions possibles dans cet exemple est 20^4=160000. Dans les simulations, l'algorithme converge à une solution acceptable dans moins de 100 itérations.

Notons ici que le contrôleur obtenu après apprentissage peut être non linéaire par rapport à la saturation des entrées et de la sortie du contrôleur par les fonctions d'appartenance aux limites des univers de discours.

Les résultats montrent que le pendule est stabilisé à la valeur désirée.

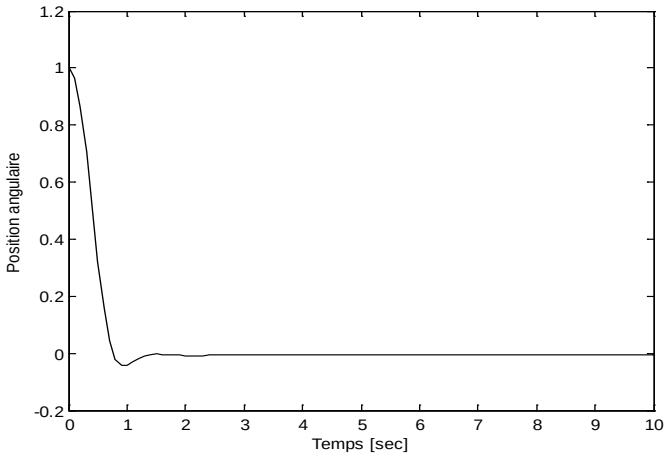


Figure 5.2 : Sortie commandée (Position Angulaire)

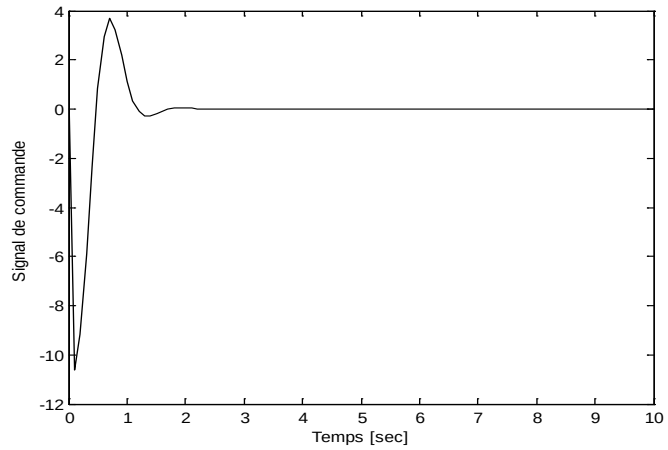


Figure 5.3 : Signal de commande

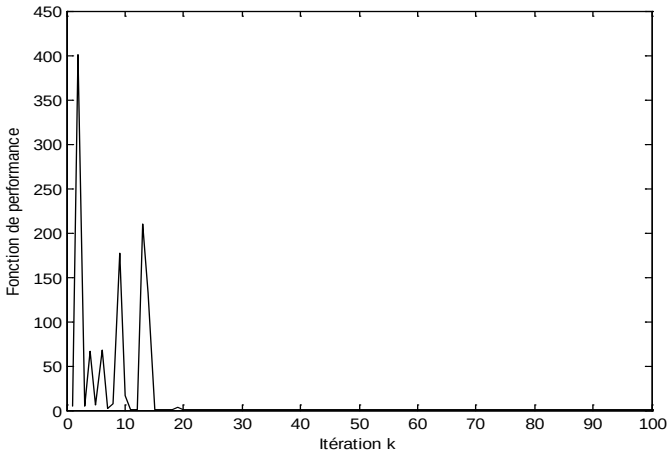


Figure 5.4 : Evolution de la fonction de performance

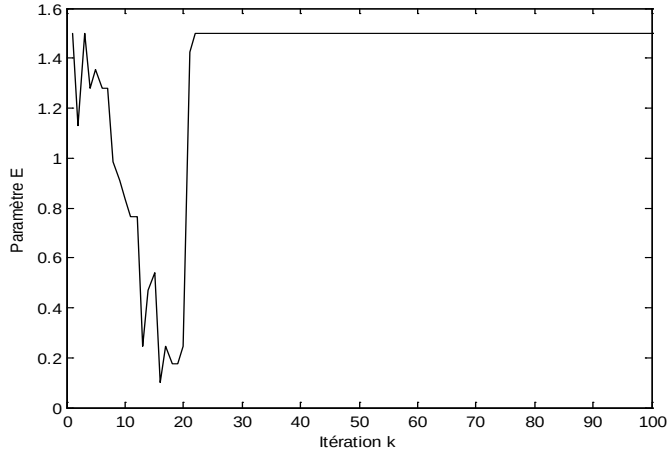


Figure 5.5 : Evolution des paramètres E

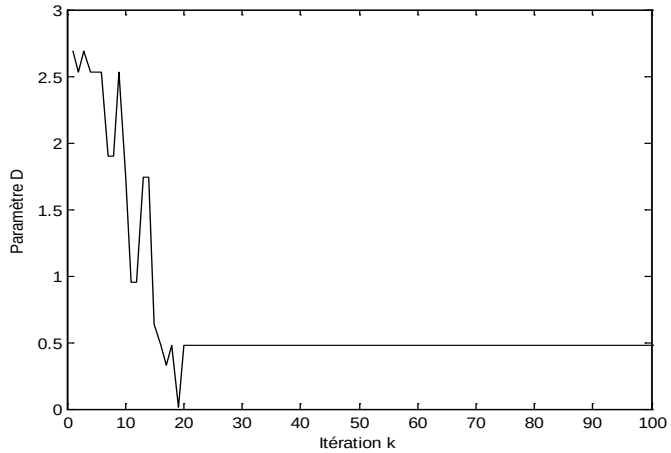


Figure 5.6 : Evolution des paramètres D

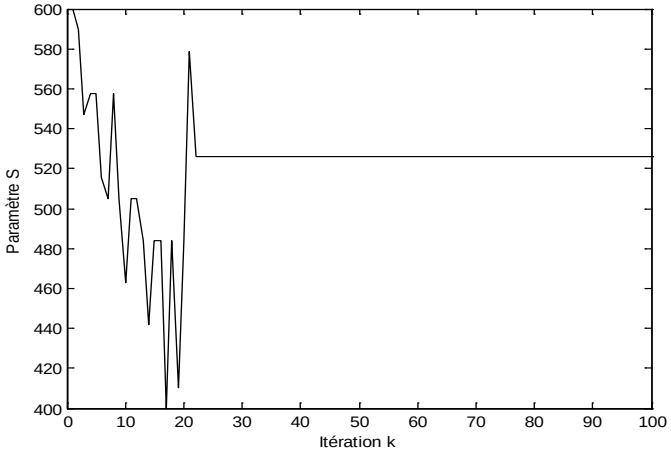


Figure 5.7 : Evolution du paramètre S

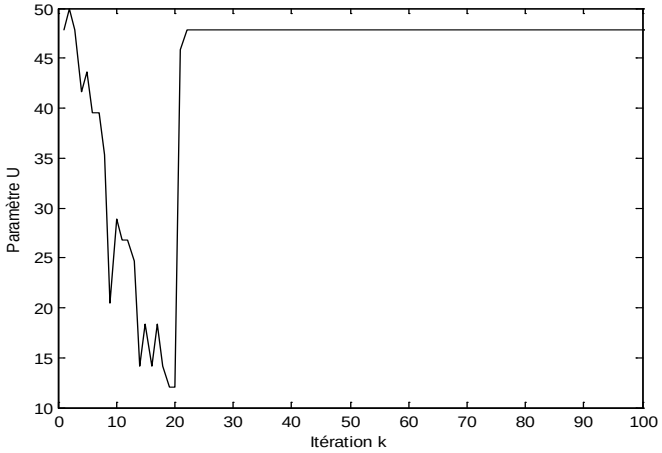


Figure 5.8 : Evolution du paramètre U

5.4 Optimisation des contrôleurs PID flous décentralisés par Q-learning

5.4.1 Présentation

Dans le cas d'une décentralisation de la commande dans un système mutivariable, le même principe sera utilisé. Le vecteur des paramètres regroupe tous les paramètres à optimiser de tous les contrôleurs et, la fonction du critère de performance combine la somme des fonctions de performance de tous les sous systèmes. Dans cette section, on va expliquer la méthode par un exemple de simulation.

5.4.2 Exemple de simulation : *Application à la stabilisation d'un simulateur d'hélicoptère*

5.4.2.1 Modèle du simulateur d'hélicoptère

Le simulateur d'hélicoptère [HUM02] est formé d'un corps portant deux pales (hélices) rigides commandées par deux moteurs à courant continu constituant le rotor sustentateur et le rotor anticouple. Les axes de rotation des hélices sont perpendiculaires. Le corps du simulateur à deux degrés de libertés, il peut pivoter autour d'un axe horizontal (angle de tangage ψ) fixé sur une fourche solidaire d'une tige verticale pivotant librement autour de son axe verticale (angle de lacet φ). Le système est multivariable à deux entrées et à deux sorties (MIMO 2x2), les sorties étant l'angle ψ et l'angle φ, et les entrées étant les tensions appliquées aux deux moteurs à courant continu. Le système est essentiellement non linéaire, instable en boucle ouverte et présente des couplages importants.

L'utilisateur du simulateur communique avec le système via l'interface de traitement de données. Les deux entrées (u_1 et u_2) sont normalisées dans l'intervalle $[-1, 1]$, où ''1'' est appelé *unité machine ''MU''*.

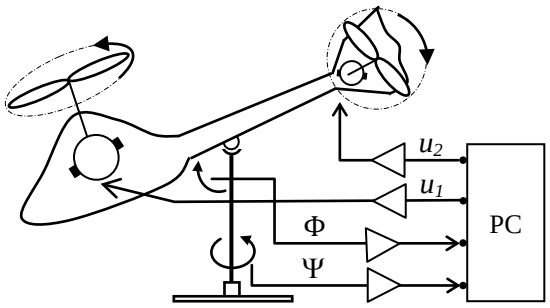


Figure 5.8 : Configuration du simulateur d'hélicoptère

Le modèle mathématique du simulateur est donné par :

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = 0.8764x_2\, sin\, x_1 + 3.4325x_4\, u_1\, cos\, x_1 + 0.4211\, x_2 - 0.0035{x_5}^2 - 46.35{x_6}^2 - 0.8076x_5x_6 - 0.0259x_5 - 2.9749x_6$$
$$\dot{x}_3 = x_4$$
$$\dot{x}_4 = 21.4010x_4 - 31.8841{x_8}^2 - 14.2029x_8 - 21.7150x_9 + 1.4010u_1$$
$$\dot{x}_5 = -6.6667x_5 - 2.7778x_6 + 2u_1$$
$$\dot{x}_6 = 4x_5$$
$$\dot{x}_7 = -8x_7 - 4x_8 + 2u_2$$
$$\dot{x}_8 = 4x_7$$
$$\dot{x}_9 = -1.3333x_9 + 0.0625u_1$$

avec: $x_1 = \Psi$, $x_2 = \frac{d\Psi}{dt}$, $x_3 = \Phi$, $x_4 = \frac{d\Phi}{dt}$, et les variables x_5 à x_9 sont les variables d'état qui représentent les deux moteurs et les effets de couplages, $u_1$et u_2 sont les deux signaux de commande normalisés.

5.4.2.2 Structure de commande

Le simulateur est vu comme une interconnexion de deux sous systèmes monovariables : le sous système d'élévation dont l'entrée est u_1 et la sortie est l'angle d'élévation Ψ et le sous système d'azimut dont l'entrée est u_2 et la sortie est l'angle d'azimut Φ.

Le problème consiste à stabiliser le simulateur d'hélicoptère autour d'un point de fonctionnement (Ψ_r, Φ_r). Pour cela, on utilise la technique de décentralisation de la commande. Les deux sous systèmes d'élévation et d'azimut sont commandés par deux contrôleurs flous de type PI. L'architecture du système de commande est présentée sur la figure 5.9.

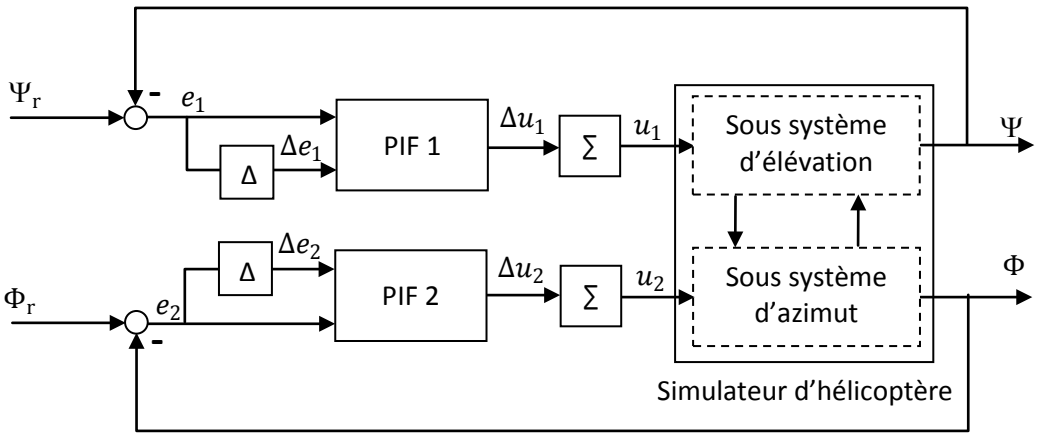


Figure 5.9 : Structure de commande

Les fonctions d'appartenances des deux contrôleurs sont représentées sur la figure 5.10 et la base de règles est donnée par le tableau 5.5.

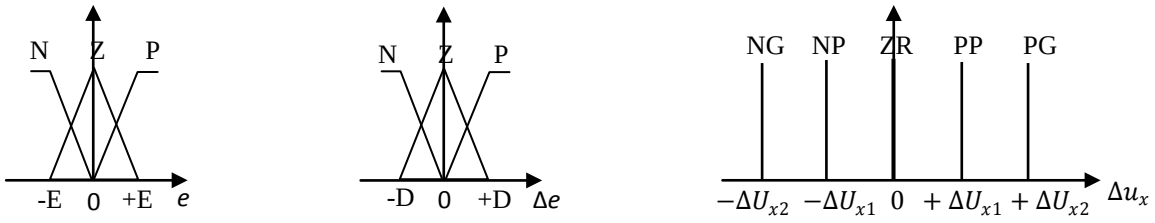


Figure 5.10 : Fonctions d'appartenance

	Δe		
e	N	Z	P
N	NG	NP	ZR
Z	NP	ZR	PP
P	ZR	PP	PG

Tableau 5.5 : Base de règles utilisée pour les deux contrôleurs PIF

5.4.2.3 Réglage des paramètres par Q-learning

Le vecteur des paramètres de réglage est composé des positions des fonctions d'appartenance des espaces d'entrés et les positions des singletons de l'espace de sortie pour les deux contrôleurs.

$$\mathrm{P} = [\mathrm{E}_1\ \mathrm{D}_1\ \Delta\mathrm{U}_{11}\ \Delta\mathrm{U}_{12}\ \mathrm{E}_2\ \mathrm{D}_2\ \Delta\mathrm{U}_{21}\ \Delta\mathrm{U}_{22}\ ]^{\mathrm{t}} \tag{5.10}$$

L'algorithme d'apprentissage par renforcement, le Q-Learning, est utilisé pour le réglage des paramètres des deux contrôleurs PIFs dans le but de minimiser la fonction de performance (5.11) composée de la somme des carrés des erreurs à la fin des réponses à deux échelons unitaires des deux sous systèmes :

$$I = \frac{1}{NT - N_1 T}\sum_{k=N_1}^{N} e_1(k)^2 + \frac{1}{NT - N_2 T}\sum_{k=N_2}^{N} e_2(k)^2 \tag{5.11}$$

avec $t_1 = N_1 T$ et $t_2 = N_2 T$ correspondent aux temps de pic approximatifs des deux sous systèmes d'élévation et d'azimut respectivement. Dans les simulations, on a pris : $t_1 = t_2 = 15s$, le pas d'échantillonnage $T = 0{,}1s$, et le temps d'un épisode est de 50s. Donc, la fonction de performance s'étend de $15\ s$ à $50\ s$.

On prend le signal de renforcement r définie comme suit :

$$r = \begin{cases} -100\ si\ I > I^* \\ +100\ si\ I \leq I^* \end{cases}, \qquad \text{avec :}\ I^* = 10^{-2} \tag{5.12}$$

Les bases de règles avec les intervalles de recherche correspondants aux conclusions des règles pour les deux contrôleurs sont données par le tableau 5.6.

	Δe		
e	N	Z	P
N	[-0.10, -0.01]	[-0.01, 0.00]	[0.00, 0.00]
Z	[-0.01, 0.00]	[0.00, 0.00]	[0.00, 0.01]
P	[0.00, 0.00]	[0.0, 0.01]	[0.01, 0.1]

Tableau 5.6 : Base de règles initiale

Les intervalles de recherche correspondants aux univers de discours sont présentés dans le tableau 5.7.

Paramètre	*Min*	*Max*
E1	0.1	1
D1	0.001	0.1
E2	0.1	1
D2	0.001	0.1

Tableau 5.7 : Bornes des paramètres de l'espace d'entrée

Le nombre de valeurs candidates pour chaque paramètres est $J = 10$.

L'algorithme est exécuté 5 fois et 300 itérations dans chaque exécution. Le meilleur résultat des cinq exécutions est présenté comme suit :

Le vecteur des paramètres à la fin de l'apprentissage est :

$$P = [E_1\ D_1\ \Delta U_{11}\ \Delta U_{12}\ E_2\ D_2\ \Delta U_{21}\ \Delta U_{22}\ ]^T$$
$$= [0.5\ 0.089\ 0.0067\ O.\,04\ 1.00\ 0.023\ 0.0033\ 0.09]^T$$

Et les bases des règles finales correspondantes aux deux contrôleurs PI flous des sous-systèmes d'élévation et d'azimut sont données respectivement par les tableaux 5.8 et 5.9.

	Δe		
e	N	Z	P
N	-0.04	-0.0067	0.00
Z	-0.0067	0.00	+0.0067
P	0.00	+0.0067	+0.04

Tableau 5.8 : Base de règles finale du PIF 1

	Δe		
e	N	Z	P
N	- 0.09	- 0.0033	0.00
Z	- 0.0033	0.00	+0.0033
P	0.00	+0.0033	+0.09

Tableau 5.9: Base de règles finale du PIF 2

La figure 5.11 montre l'évolution de la fonction de performance. Les figures 5.12 et 5.13 montrent respectivement l'angle d'élévation et l'angle d'azimut.

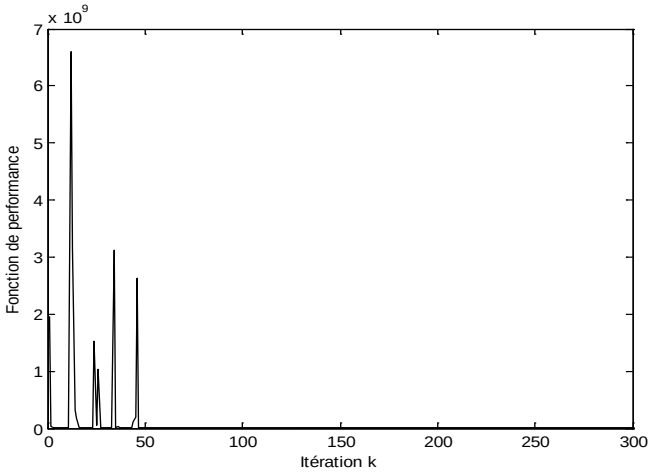


Figure 5.11 : Evolution de la fonction de performance

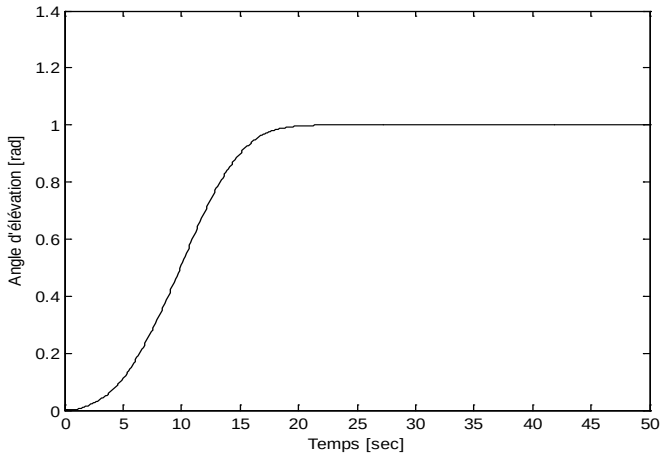


Figure 5.12 : Angle d'élévation

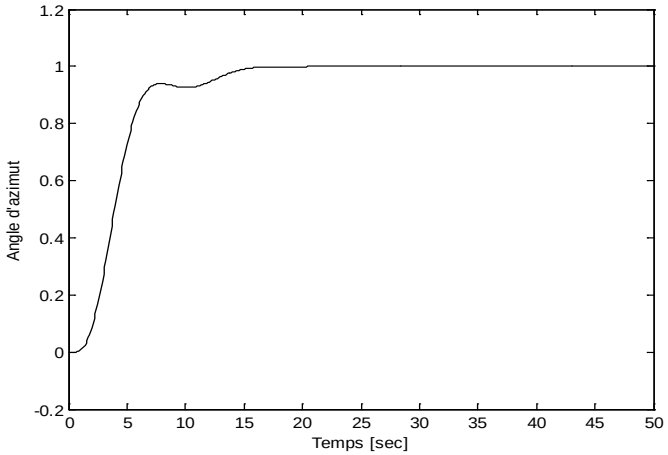


Figure 5.13 : Angle d'azimut

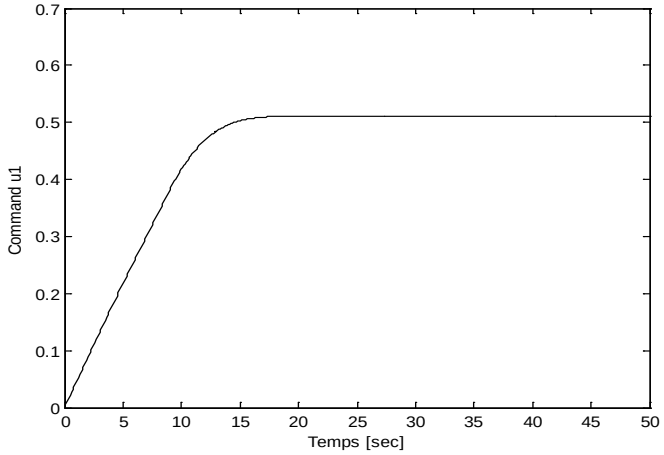


Figure 5.14 : Commande u_1

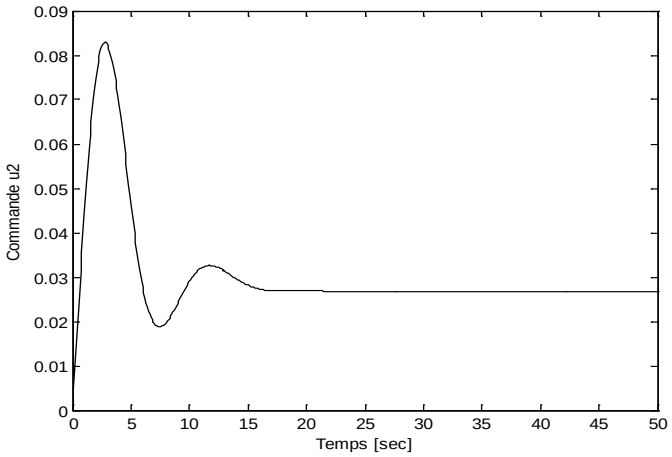


Figure 5.13 : Commande u_2

5.5 Optimisation des contrôleurs flous par Q-learning flou

5.5.1 Présentation

On trouve dans la littérature un algorithme Q-Learning associé aux systèmes flous, appelé le Q-Learning flou [GLO94, GLO99], cet algorithme à pour objectif l'optimisation des conclusions des règles flous. On trouve dans la littérature plusieurs applications de cet algorithme ; problèmes navigation des robots mobiles [BOU06b, ZAV03, KER06, GU04], optimisation des contrôleurs PI flous [BOU06a].

Lors de l'apprentissage par l'algorithme du Q-learning flou, le contrôleur doit donc ajuster les conclusions de ses règles sur la base du renforcement. On rappelle ici les principaux points de l'algorithme :

- chaque règle possède plusieurs conclusions potentielles ;
- chaque conclusion potentielle est associée à un indice de qualité ;
- le régulateur évalue la qualité de chaque conclusion potentielle, selon une politique d'exploration donnée ;
- à l'issue de l'apprentissage, chaque règle floue se voit associer à sa conclusion potentielle de meilleure qualité.

La règle *i* du contrôleur flou prend donc la forme suivante :

Si "$prémisse\ i$ " Alors $c_i = c_{i1}$ avec $Q_i = Q_{i1}$
ou $c_i = c_{i2}$ avec $Q_i = Q_{i2}$
$\vdots$
ou $c_i = c_{iJ}$ avec $Q_i = Q_{iJ}$

c_{ij} représente la j^{ème} conclusion potentielle et Q_{ij} la qualité associée. Les c_{ij} sont des interprétations raisonnables des labels correspondants.

L'algorithme Q-learning flou que nous proposons et que nous appellons algorithme Q-learning flou modifié [BOU09b, BOU08b] est différent de l'algorithme Q-learning flou standard. Dans l'algorithme Q-learning flou standard, la valeur de mise à jour est calculée par le système d'inférence flou et elle est commune à toutes les valeurs Q associées aux conclusions choisies. Dans notre algorithme, chaque valeur Q associée à une conclusion choisie est adaptée séparément par :

$$Q_{ik}(s_t, a_t) \leftarrow Q_{ik}(s_t, a_t) + \delta\left(r(s_t, a_t) + \gamma \max_{b \in A} Q_{ik}(s_{t+1}, b) - Q_{ik}(s_t, a_t)\right)\alpha_i(s_t) \quad (5.13)$$

avec k l'indice de la conclusion choisie par la politique d'exploration/exploitation utilisée et $\alpha_i(s_t)$ est la valeur de vérité de la règle i dans l'état s_t. Les autres valeurs Q_{ij}, avec $1 \leq j \leq J$ et $j \neq k$ sont laissés sans changement.

Le pseudo code du Q-learning modifié est donné par l'Algorithme 5.3.

Algorithme 5.3 : Q-learning flou Modifié

Initialiser Q_{ij} arbitrairement, $1 \leq i \leq M, 1 \leq j \leq J$
Observer l'état initial s_t.
Répéter pour chaque itération de temps
  Pour chaque règle i,
    Calculer la valeur de vérité $\alpha_i(s_t)$.
    Choisir $k, 1 \leq k \leq J$ avec la *PEE* et choisir la conclusion correspondante c_{ik}.
  Fin Pour
  Calculer l'action a_t par le système d'inférence flou.
  Appliquer l'action a_t et observer le nouvel état s_{t+1}.
  Recevoir le renforcement r.
  Calculer $\alpha_i(s_{t+1})$.
  Faire la mise à jour des valeurs Q_{ik} par la relation (5.13)
  $t \leftarrow t + 1$
Jusqu'à t_{final}
Retourner les conclusions optimales par la politique gloutonne (5.3)

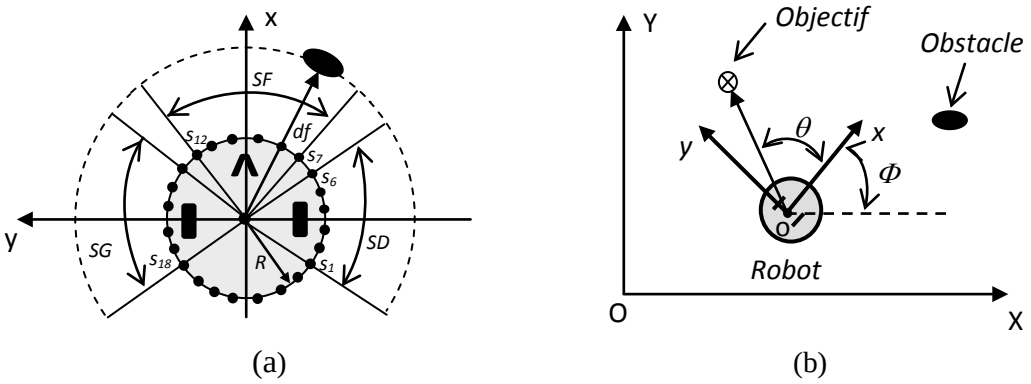
5.5.2 Exemple de simulation : *Navigation d'un robot mobile*

Considérons le problème de navigation d'un robot mobile. Le problème consiste à doter le robot avec une capacité d'éviter les obstacles et la recherche d'un objectif sans être piégé dans un minimum local.

5.5.2.1 Modèle du robot

Soit un modèle cylindrique d'un robot mobile de rayon R=20 cm. Le robot est omnidirectionnel et équipé de 24 capteurs ultrasoniques uniformément distribués sur un anneau. Chaque capteur, s_i avec i=1,2,…,24 à une porté de 10 cm à 250 cm et couvre un angle de vue de 15° et donne la distance du robot à l'obstacle l_i dans son champ de vue. Pour réduire la dimension de l'entrée, les capteurs au devant du robot sont arrangés en trois groupes (Figure 5.16 (a)); le groupe de gauche SG est composés des 6 capteurs voisins s_i $(i = 1,2, \dots,6)$, le groupe de face SF est composés des 6 capteurs voisins $s_i(i = 7,8, \dots,12)$ et le groupe de droite SD est composés des 6 capteurs voisins s_i $(i = 13,14, \dots,18)$. Avec cet arrangement des capteurs, les distances du centre du robot mobile à l'obstacle mesurées par les trois groupes de capteurs SD, SF et SG notées respectivement dd, df et dg sont exprimées comme suit :

$$\begin{cases} dd = R + min_{i=1,2,3}(L_i) \\ df = R + min_{i=4,5\dots,9}(L_i) \\ dg = R + min_{i=10,11,12}(L_i) \end{cases} \quad (5.14)$$



(a) (b)

Figure 5.16 : (a) Configuration du robot et arrangement des capteurs ; (b) Systèmes de coordonnées

Nous utilisons deux systèmes de coordonnées ; le système $X0Y$ dans l'espace et le système xoy du robot mobile où o est le centre du robot et l'axe x passe par au milieu entre les deux capteurs s_6 et s_7(Figure 5.16 (b)). Les actions du robot sont l'angle de braquage ΔΦ et sa vitesse linéaire. Pour le comportement de la recherche d'un objectif, le robot connaît la position de l'objectif définit par l'angle l'axe d'orientation x et la ligne connectant le centre du robot à l'objectif.

5.5.2.2 Navigateur flou

Les entrées du navigateur flou sont les distances dans les trois directions : droite dd, face df, et gauche dg. Les sorties sont la vitesse v du centre du robot et l'angle de braquage $\Delta\Phi$.

Deux variables linguistiques ; Proche (P) et Loin (L), sont utilisés pour décrire chacune des trois distances. Les variables linguistiques sont représentées par les fonctions d'appartenance de la figure 5.17. Avec d_m la distance minimale permise entre le robot et l'obstacle et d_s la distance de sécurité au delà de laquelle le robot peut se déplacer à grande vitesse.

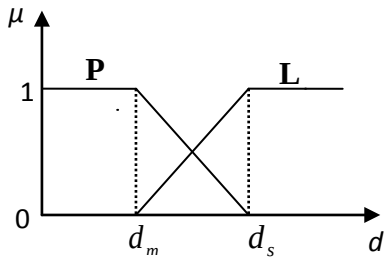


Figure 5.17 Fonctions d'appartenance des entées du navigateur flou

La base de règles pour l'évitement d'obstacle et la recherche d'un objectif est construite sur la base du résonnement humain. Elle peut être interprétée comme suit :

- Si le robot est loin des obstacles dans les trois directions, alors il tourne vers l'objectif et se dirige directement vers lui avec sa vitesse maximale.
- Si l'objectif n'est pas en face du robot mais il y a des obstacles, alors le robot longe l'obstacle le plus proche à sa droit ou à sa gauche.
- Si l'objectif est en face du robot ainsi que des obstacles, alors le robot tourne vers l'objectif en longeant l'obstacle le plus proche à sa droite où à sa gauche.

On définit les deux paramètres p et q comme suit :

$$p = \begin{cases} 0 & si\ dg \leq dd\ and\ dg < d_M \\ 1 & sinon \end{cases} \tag{5.15}$$

$$q = \begin{cases} 1 & si\ 90° < \theta < 270° \\ 0 & sinon \end{cases} \tag{5.16}$$

d_M est la distance maximale qui peut être détectée par un capteur.

Les deux paramètres p et q permettent au navigateur de sélectionner le mode de comportement ; si $p = 1$ $(p = 0)$ alors le mode de suivre l'obstacle à droite (à gauche) est

sélectionné, et si $q = 1$ le mode de recherche de l'objectif est sélectionné sinon le mode de longer les obstacles est activé.

Maintenant on regroupe les trois distances dans un triplet (dd, df, dg). Donc, la base de règle est exprimée comme suit :

$$Règle\ 1 : Si\ (PPP)\ Alors\ v_1 = ZR\ et\ \Delta\Phi_1 = p \times PG + (1-p) \times NG$$
$$Règle\ 2 : Si\ (PPL)\ Alors\ v_2 = ZR\ et\ \Delta\Phi_2 = NG$$
$$Règle\ 3 : Si\ (PLP)\ Alors\ v_3 = C \times V_{max}\ et\ \Delta\Phi_3 = ZR$$
$$Règle\ 4 : Si\ (PLL)\ Alors\ v_4 = V_{max}\ et\ \Delta\Phi_4 = p \times NP + (1-p) \times PP$$
$$Règle\ 5 : Si\ (LPP)\ Alors\ v_5 = ZR\ et\ \Delta\Phi_5 = PG$$
$$Règle\ 6 : Si\ (LPL)\ Alors\ v_6 = ZR\ et\ \Delta\Phi_6 = p \times PG + (1-p) \times NG$$
$$Règle\ 7 : Si\ (LLP)\ Alors\ v_7 = V_{max}\ et\ \Delta\Phi_7 = p \times NP + (1-p) \times PP$$
$$Règle\ 8 : Si\ (LLL)\ Alors\ v_8 = V_{max}\ et\ \Delta\Phi_8 = q \times \theta + (1-q) \times (p \times NM + (1-p) \times PM$$

V_{max} est la vitesse maximale du robot, $\Delta\Phi_i$ et v_i sont les conclusions des règles floues, et C est un facteur de réduction de vitesse. Dans les simulations, on a pris $V_{max} = 1\ m/s$ et $C = 0.1$.

Les conclusions des règles sont des singletons interprété linguistiquement par :

PG (*Positif Grand*), PM (*Positif Moyen*), PP (*Positif Petit*), ZR (environ *Zero*), NP (*Negatif Petit*), NM (*Negatif Moyen*), and NG (*Negatif Grand*).

Ici, vue la symétrie du robot, on prend : NG=-PG, NM=-PM, NP=-PP. Donc le nombre de paramètres sera réduit et la base des règles devient :

$$Règle\ 1 : Si\ (PPP)\ Alors\ v_1 = ZR\ et\ \Delta\Phi_1 = (2 \times p - 1) \times PG$$
$$Règle\ 2 : Si\ (PPL)\ Alors\ v_2 = ZR\ et\ \Delta\Phi_2 = NG$$
$$Règle\ 3 : Si\ (PLP)\ Alors\ v_3 = C \times V_{max}\ et\ \Delta\Phi_3 = ZR$$
$$Règle\ 4 : Si\ (PLL)\ Alors\ v_4 = V_{max}\ et\ \Delta\Phi_4 = (2 \times p - 1) \times NP$$
$$Règle\ 5 : Si\ (LPP)\ Alors\ v_5 = ZR\ et\ \Delta\Phi_5 = PG$$
$$Règle\ 6 : Si\ (LPL)\ Alors\ v_6 = ZR\ et\ \Delta\Phi_6 = (2 \times p - 1) \times NG$$
$$Règle\ 7 : Si\ (LLP)\ Alors\ v_7 = V_{max}\ et\ \Delta\Phi_7 = (2 \times p - 1) \times NP$$
$$Règle\ 8 : Si\ (LLL)\ Alors\ v_8 = V_{max}\ et\ \Delta\Phi_8 = q \times \theta + (1-q) \times (2 \times p - 1) \times PG$$

Les sorties du navigateur inférées par la méthode de la moyenne pondérée sont données par:

$$\Delta\Phi = \frac{\sum_{i=1}^{8} \alpha_i \Delta\Phi_i}{\sum_{i=1}^{8} \alpha_i} \tag{5.17}$$

$$v = \frac{\sum_{i=1}^{8} \alpha_i v_i}{\sum_{i=1}^{8} \alpha_i} \tag{5.18}$$

α_i la valeur de vérité de la règle *i* calculée par la méthode du produit algébrique.

5.5.2.3 Réglage des conclusions des règles floues

L'algorithme du Q-learning modifié est utilisé pour l'optimisation du navigateur. Le nombre des valeurs candidates pour chaque conclusion est $J = 5$ uniformément distribuées sur les intervalles présentés dans le tableau 5.10.

On utilise un signal de renforcement qui puni toute conclusion de règle activée quand le robot rentre en collusion avec un obstacle :

$$r = \begin{cases} -1 & si\ min(dd, df, dg) < d_m \\ 0 & sinon \end{cases} \quad (5.19)$$

Pendant la phase d'apprentissage (figure 5.18), seule la tâche de longer le mur à droite est sélectionnée ($q = 0$), le robot est laissé dans un environnement inconnu. Après un temps d'apprentissage suffisant, l'algorithme est arrêté.

	NP	NM	NG	ZR	PP	PG
Min	-40°	-40°	-80°	0°	5°	80°
Max	-5°	-5	-40°	0°	40°	40°

Tableau 5.10 : Conclusions des règles floues pour ΔΦ

Après la phase d'apprentissage, l'algorithme est arrêté et les conclusions des règles avec les meilleures sont choisies. La figure 5.19 montre les résultats de simulations pour la tâche de recherche de l'objectif à partir de quatre positions initiales différentes pour le départ du robot.

5.5.2.4 Comparaison avec des travaux antérieurs

Sur la figure 5.20, on montre les résultats de simulation avec la base des règles floues utilisée dans [ZAV03, DAH04, KER06] en plaçant le robot dans un environnement relativement complexe. On remarque que le robot est coincé et n'arrive pas à joindre son objectif. Dans la figure 5.21, on montre les résultats de simulation avec la méthode de navigation proposée en plaçant le robot même endroit, on remarque que le robot échappe du minimum local et rejoint son objectif.

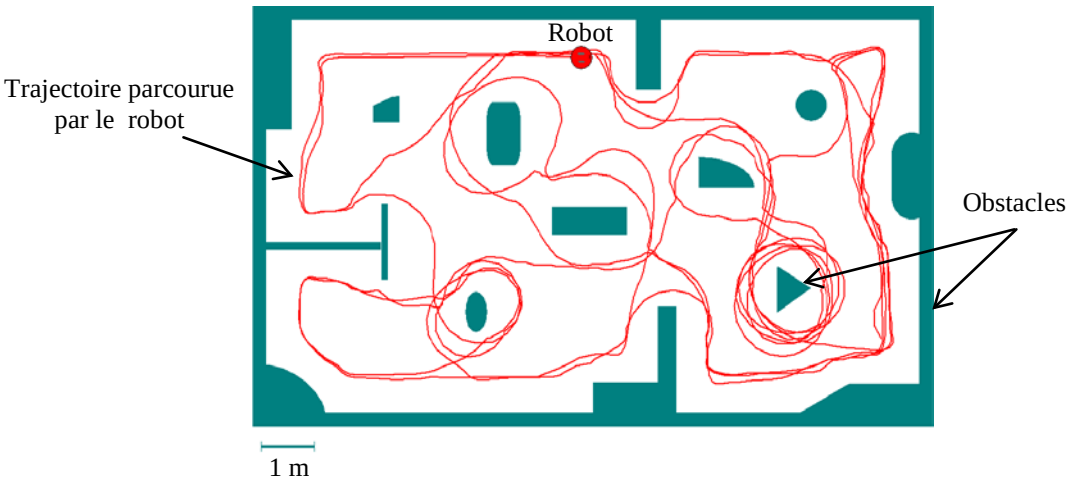


Figure 5.18 : Phase d'apprentissage

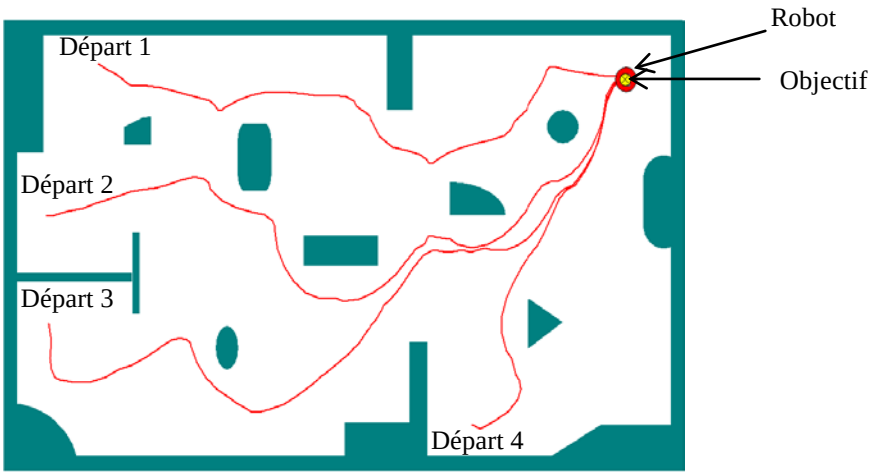


Figure 5.19 : Recherche de l'objectif

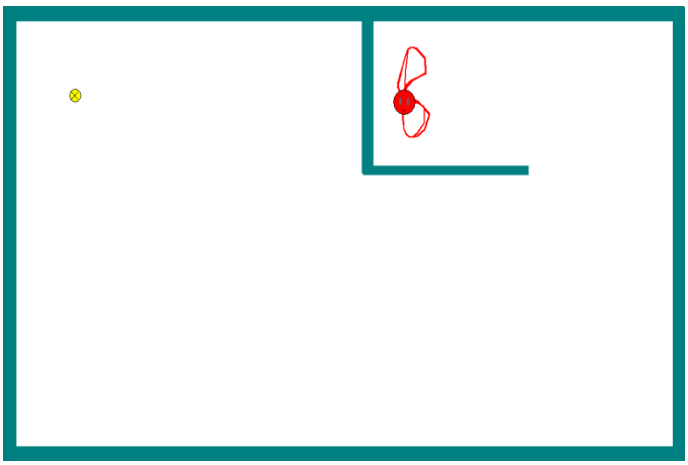


Figure 5.20 : Robot piégé dans un minimum local

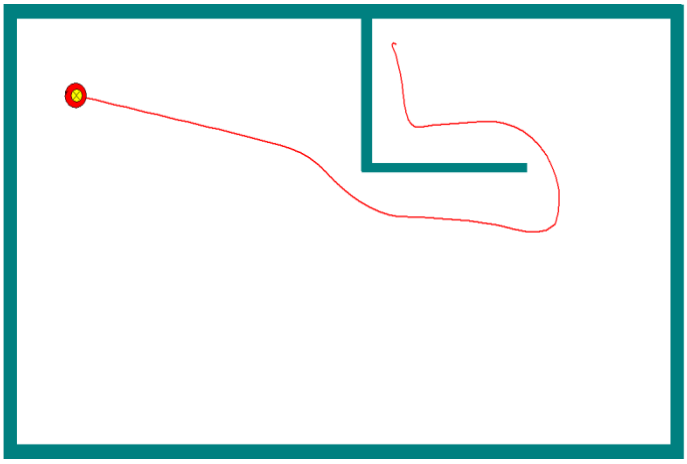


Figure 5.21 : Le robot échappe du minimum local

5.6 Conclusion

Dans ce chapitre, nous avons proposé des méthodes de réglage des contrôleurs flous par le Q-learning, l'algorithme d'apprentissage par renforcement le plus utilisé en pratique. Ces méthodes peuvent être appliquées à tout type de contrôleurs flous. En particulier les contrôleurs PID flous linéaires et non linéaires présentés dans le quatrième chapitre. Dans le cas des contrôleurs « linéaires », seules les valeurs maximales des univers de discours sont ajustées en ligne afin de minimiser un critère de performance. Dans le cas des contrôleurs flous non linéaires, on peut procéder de deux façons : soit commencer le réglage avec un contrôleur initialement « linéaire » et puis le comportement non linéaire est obtenu par ajustement des positions de toutes les fonctions d'appartenance sur l'espace d'entrée et/ou les positions des singletons sur l'espace de sortie ou soit par la considération dès le début que tous les paramètres sont soumis à l'apprentissage. La phase d'apprentissage peut être accélérée par l'introduction des connaissances dans le contrôleur initial. L'efficacité des méthodes proposées est montrée via des exemples de simulation.

Chapitre 6

Optimisation des contrôleurs flous par colonies de fourmis

6.1 Introduction

On a vu dans le troisième chapitre que l'optimisation par colonies de fourmis a été inspirée du comportement réel des colonies de fourmis à la recherche de la nourriture. Dans ces algorithmes, les ressources de calcul sont attribuées à un groupe de fourmis artificielles qui exploitent une forme de communication au moyen de l'environnement pour trouver le trajet le plus court du nid des fourmis à une source de nourriture. Ainsi, les fourmis artificielles coopèrent pour trouver des bonnes solutions. La mesure de performance est basée sur une fonction de qualité.

Plusieurs algorithmes d'optimisation par colonies de fourmis ont été proposés dans la littérature [DOR96a, DOR96b, DOR97, STU00, GAM95] et appliqués avec succès pour la résolution d'une variété de problèmes d'optimisation [CLE02, DOR97, DUA06, MEN02, SIM03, GAM99, STU97].

Chaque version de ces algorithmes d'optimisation par colonies de fourmis se caractérise essentiellement par la règle de mise à jour de la phéromone et par l'heuristique utilisée pour le choix des itinéraires par les fourmis artificielles.

Dans ce chapitre, nous proposons une méthode de réglage des paramètres des contrôleurs flous par un algorithme d'optimisation par colonies de fourmis. Notre algorithme utilise une règle de mise à jour de phéromone et une heuristique de sélection différentes de celles déjà utilisées dans la littérature.

6.2 Formulation du problème d'optimisation d'un contrôleur PID flou par ACO

Sans perte de généralité, considérons le problème de réglage des paramètres du PID flou linéaire présenté dans le chapitre 4, [BOU09a].

Le vecteur des paramètres est définit comme suit :

$$P = [\, P_1\ P_2\ P_3\ P_4]^{\,t} = [E\ D\ S\ U]^t$$

De la même manière que dans le cinquième chapitre, à chaque paramètre on attribue un ensemble de valeurs candidates. Pour simplifier, ces valeurs candidates sont uniformément distribuées sur un intervalle dont les bornes sont fixées a priori sur la base des connaissances sur le système.

Soit P^i_{min} et P^i_{max} les bornes du paramètre P_i, Donc :

$$P_{i1} = P^i_{min}, P_{i2} = P_{i1} + \frac{P^i_{max} - P^i_{min}}{J-1}, \dots, P_{iJ} = P^i_{max}$$

avec :

$$P_i \in \{\, P_{i1}, P_{i2}, \dots, P_{iJ}\}$$

Le problème consiste en la recherche des meilleurs paramètres qui permettent de minimiser la fonction de performance. C'est un problème d'optimisation combinatoire avec une complexité égal à J^4.

La figure 6.1 donne une représentation graphique du problème, les quatre ensembles de paramètres *E*, *D*, *S* et *U* sont arrangés dans quatre listes en colonne où chaque valeur candidate est représentée par un nœud.

Le tour d'une fourmi consiste en une combinaison des paramètres du contrôleur flou. A partir de son nid, une fourmi se déplace à travers *E*, *D*, *S* et *U*. Enfin, la fourmi atteint la source d'alimentation *F* qui est ajoutée ici juste par analogie au monde réel des fourmis.

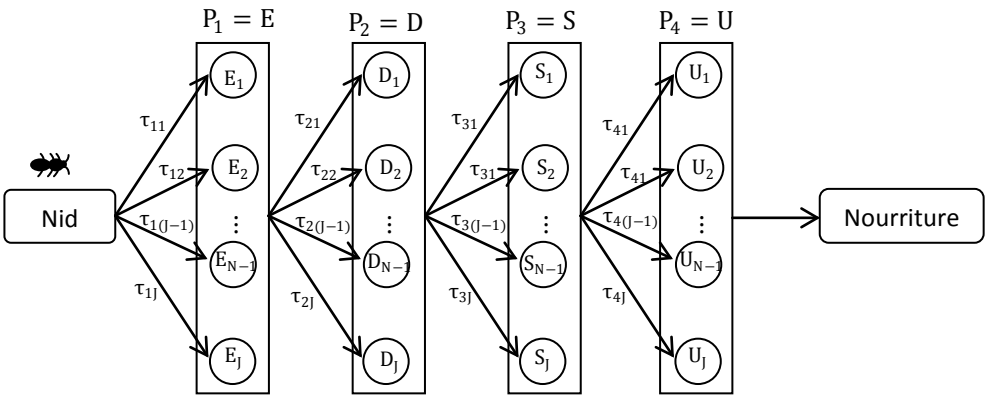


Figure 6.1: La relation entre le tour d'une fourmi et les paramètres du contrôleur flou

6.3 Règle de sélection des paramètres par les fourmis

Pour chaque paramètre, le nœud visité par une fourmi est sélectionné comme une valeur de ce paramètre. La sélection d'une valeur d'un paramètre est basée sur les traces de phéromone entre les vecteurs de paramètres. La dimension de la matrice de phéromone τ est $4\times N$ et chaque élément est noté par τ_{ij}, $i = 1,2,\dots,3$ et $j = 1,2,\dots,J$. Comme montré sur la figure 6.1, quand une fourmi arrive au vecteur P_x, la sélection de la valeur du paramètre suivant P_{x+1j} dans la liste des candidates P_{x+1}est donné en utilisant la règle de transition (6.1) qui dépond de la trace de phéromone $\tau_{x+1j}, j = 1,2,\dots,J$, et un peu d'heuristique.

$$j = \begin{cases} \arg\max_{y\in J}\{\tau_{iy}(t)\} & si\ q \le q_0 \\ j_1 & si\ q_0 < q < q_1 \\ j_2 & si\ q \ge q_1 \end{cases} \tag{6.1}$$

$$Pr_{ij_1}(t) = \frac{\tau_{ij_1}(t)}{\sum_{l=1}^{N}\tau_{il}(t)} \tag{6.2}$$

où q est une variable aléatoire uniformément distribuée sur $[0,1]$, q_0 et q_1 $(q_0 < q_1)$ sont deux paramètres de réglage dans $[0,1]$, j_1 appartient à la liste des candidats $j_1 \in \{1,2,\dots,J\}$ qui est sélectionné sur la base de la règle de probabilité (6.2) et j_2 est sélectionné sur la base d'une règle de probabilité uniforme sur la liste des candidats $j_2 \in \{1,2,\dots,J\}$.

6.4 Règle de mise à jour de la phéromone

Chaque fourmi modifie l'environnement par le dépôt de phéromone. Supposant qu'à l'instant initial $t = 0$ toutes les traces ont la même concentration τ_0, donc, $\tau_{ij}(0) = \tau_0$ ($i = 1,2,\dots,4$ et $j = 1,2,\dots,J$).

Quand une fourmi passe par un nœud P_{ij}, la concentration de phéromone de ce nœud sera mise à jour immédiatement par la règle suivante :

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \rho\frac{1}{I} \tag{6.3}$$

avec ρ un coefficient qui définit la quantité de mise à jour de la phéromone et I la fonction de performance.

La règle de mise à jour de phéromone (6.3) montre que le meilleur tour recevra la plus grande concentration, i.e., plus I est petit, plus la portion de mise à jour de phéromone est grande.

Les fourmis vont explorer et tester plusieurs solutions. La phase d'exploration est souvent longue. Cependant, les règles floues et les paramètres du contrôleur flou sont interprétables, cette phase sera remarquablement réduite par l'introduction des connaissances apriori dans le contrôleur initial.

Après un temps d'apprentissage suffisant, le processus de recherche peut être arrêté et la stratégie gloutonne (*greedy*) sera utilisée pour une exploitation finale des résultats obtenues. Les paramètres optimaux sont ceux qui correspondent aux traces de phéromone les plus concentrés. Pour chaque paramètre P_i, la valeur optimale P_{ij_o} est choisie par la règle gloutonne suivante :

$$j_0 = arg\,max_{y\in J}\{\tau_{iy}(t)\} \tag{6.4}$$

Le pseudo code de l'algorithme est donné par l'algorithme 6.1

Algorithme 6.1 : Algorithme d'optimisation des paramètres d'un PIDF par ACO

Initialiser les traces de phéromone $\tau_{ij}(0) \leftarrow \tau_0$, ($i = 1,2,\dots,4$ and $j = 1,2,\dots,J$).
Pour chaque fourmi k, $k = 1,2,\dots,m$
    Choisir le vecteur de paramètres P en utilisant les équations (6.4) et (6.5).
    Calculer l'action u par le système d'inférence.
    Appliquer l'action u.
    Evaluer la fonction de qualité.
    Mettre à jour les traces de phéromone par l'équation (6.6)
Fin pour

6.5 Simulations

6.5.1 Exemple 1 : *Stabilisation d'un pendule inversé par un PID flou*

On reprend l'exemple de stabilisation du pendule inversé avec un PID flou linéaire présenté dans la section 5.3.2.

Le vecteur des paramètres est :

$$P = [E\ D\ S\ U]^T$$

Les bornes des paramètres sont données par le tableau 6.1. Pour chaque paramètre, le nombre de valeur candidates $J = 100$. Les coefficients de l'algorithme de colonie de fourmis sont choisies comme suit : $\tau_0 = 0.1$, $q_0 = 0.5$, $q_1 = 0.75$, $\delta = 0.1$.

Paramètre	*Min*	*Max*
E	0.1	3
D	0.1	5
S	400	600
U	10	60

Tableau 6.1 : Définition des bornes de paramètres

L'algorithme d'optimisation par colonies de fourmis est exécuté 5 fois pour 100 tours de fourmis (itérations). A chaque exécution l'algorithme converge à une solution acceptable après moins de 20 itérations. Parmi ces cinq exécutions, le meilleur résultat correspondant à la valeur minimale de fonction de performance.

La figure 6.2 montre l'évolution en fonction du nombre de tours de la fonction de performance quand l'algorithme de colonie de fourmis proposé est utilisé pour le réglage des paramètres du PID flou. Les figures 6.3 et 6.4 présentent respectivement la sortie du système et le signal de commande. L'évolution des paramètres *E, D, S* et *U* est illustrée par les figures 6.5 à 6.8.

L'algorithme converge à la fonction de performance $I = 1.7782 \times 10^{-4}$ qui correspond aux valeurs finales des paramètres données dans le tableau 6.2.

Paramètre	*Valeur finale*
E	1.1545
D	0.4000
S	480.8081
U	58.9899

Tableau 6.2 : Les valeurs finales des paramètres.

Le nombre de solutions possibles dans cet exemple est: $100^4 = 10^8$. Dans les simulations, l'algorithme converge à une solution relativement acceptable après moins de 20 itérations et la sortie de commande suit la référence.

On note que le contrôleur PID flou final peut être non linéaire vis-à-vis la saturation de ces entrées par les fonctions d'appartenance aux limites des univers de discours.

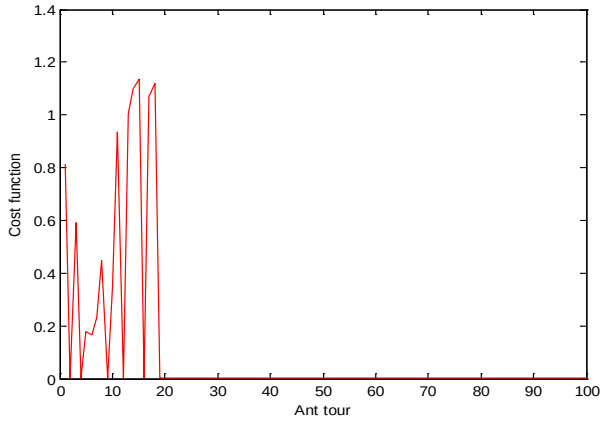


Figure 6.2 : Evolution de la fonction de performance avec l'algorithme ACO proposé.

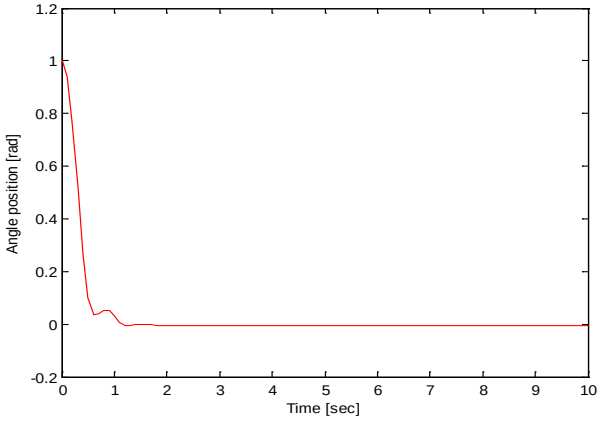


Figure 6.3 : Sortie du système (position angulaire).

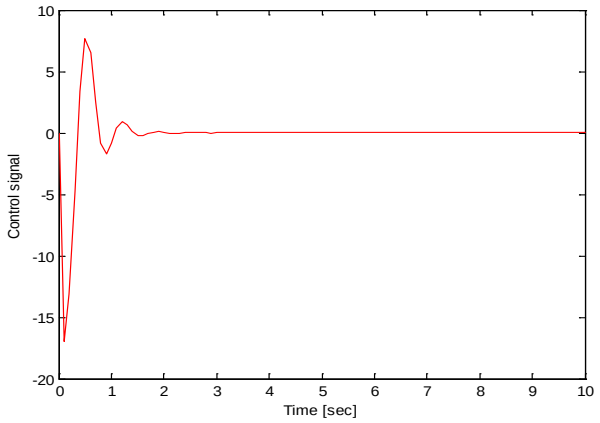


Figure 6.4: Signal de commande.

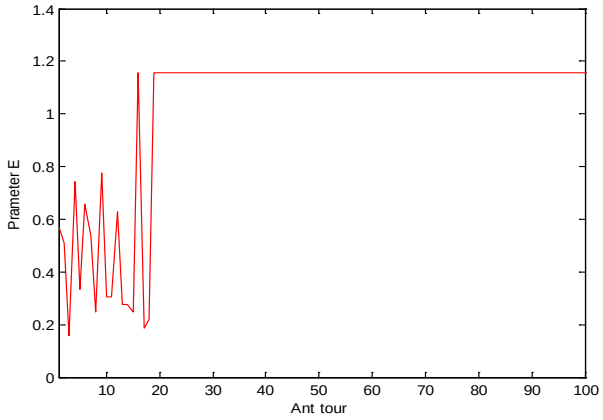


Figure 6.5 : Evolution du paramètre E.

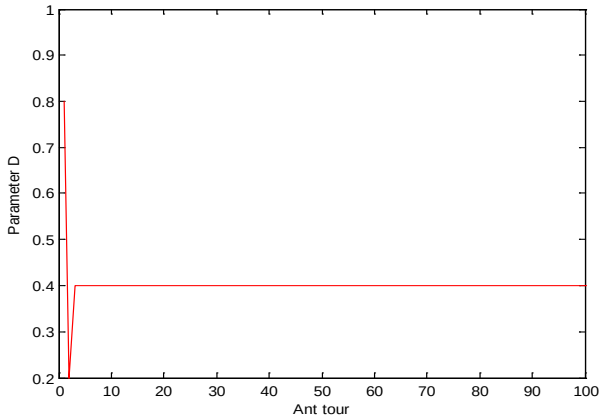


Figure 6.6 : Evolution du paramètre D.

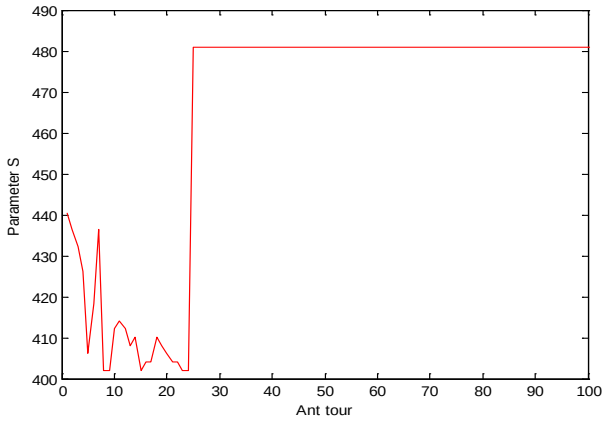


Figure 6.7 : Evolution du paramètre S.

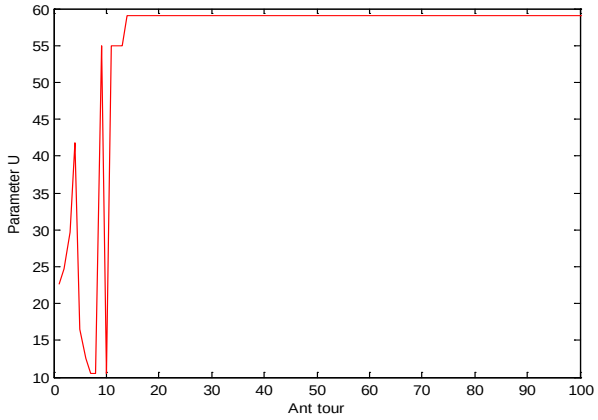


Figure 6.8 : Evolution du paramètre U

6.5.2 Exemple 2: *Stabilisation d'un simulateur d'hélicoptère par deux PI flous décentralisés*

On reprend le problème de stabilisation du simulateur d'hélicoptère par deux contrôleurs PI flous décentralisés présentés dans la section 5.4.2. Le vecteur des paramètres à régler est composée des valeurs maximales des univers de discours des variables d'entrée et les positions des singletons de la variable de sortie, soit donc :

$$P = [E_1\ D_1\ \Delta U_{11}\ \Delta U_{12}\ E_2\ D_2\ \Delta U_{21}\ \Delta U_{22}\ ]^T$$

L'algorithme de colonies de fourmis présenté dans la section précédente, est utilisé pour le réglage des paramètres des deux contrôleurs PIF dans le but de minimiser la fonction de performance (2.11) composée par la somme des carrés des erreurs à la fin des réponses à deux échelons unitaires des deux sous systèmes d'élévation et d'azimut.

L'algorithme est exécuté 5 fois avec 500 itérations par exécution. Le meilleur résultat des cinq exécutions est retenu. Le vecteur des paramètres à la fin de l'apprentissage est :

$$P = [E_1\ D_1\ \Delta U_{11}\ \Delta U_{12}\ E_2\ D_2\ \Delta U_{21}\ \Delta U_{22}\ ]^T$$
$$= [0.5\ 0.1\ 0.04\ O.0078\ 0.89\ 0.023\ 0.04\ 0.0055]^T$$

Et les bases des règles finales correspondantes aux deux contrôleurs PIF des sous systèmes d'élévation et d'azimut sont données respectivement par les tableaux 6.3 et 6.4. La figure 6.9 montre l'évolution de la fonction de performance. Les figures 6.10 et 6.11 montrent respectivement l'angle d'élévation et l'angle d'azimut, et les figures 6.12 et 6.13 montrent les signaux de commande. On remarque qu'après apprentissage, les sorties du système suivent parfaitement leurs références.

	Δe		
e	N	*Z*	P
N	-0.04	-0.0078	0.00
Z	-0.0078	0.00	+0.0078
P	0.00	+0.0078	+0.04

Tableau 6.3 : Base de règles finale du PIF 1

	Δe		
e	N	Z	P
N	- 0.04	- 0.0033	0.00
Z	- 0.0033	0.00	+0.0055
P	0.00	+0.0055	+0.04

Tableau 6.4: Base de règles finale du PIF 2

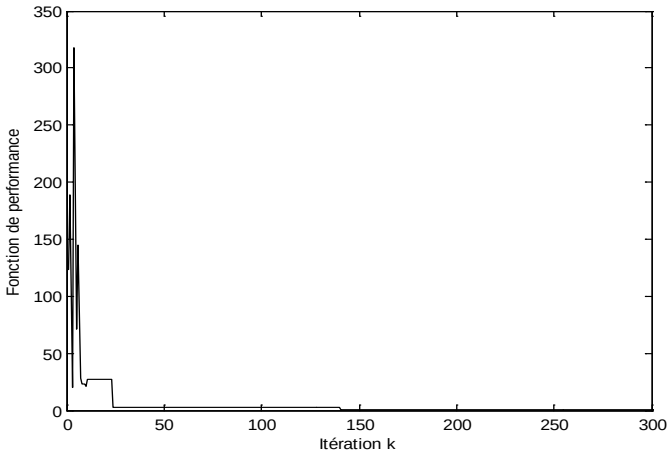


Figure 6.9: Evolution de la fonction de performance

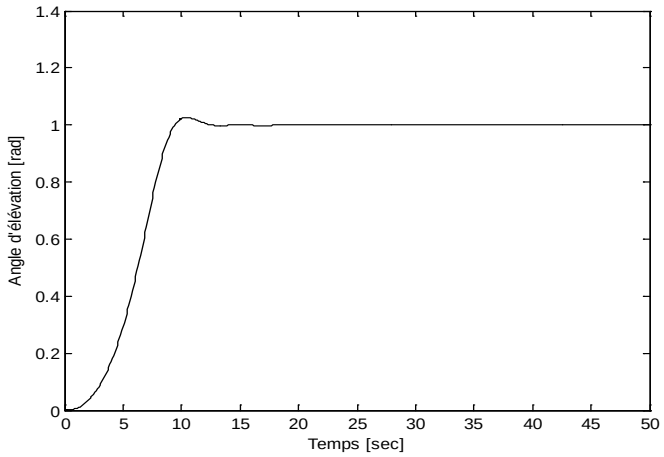


Figure 6.10 : Angle d'élévation

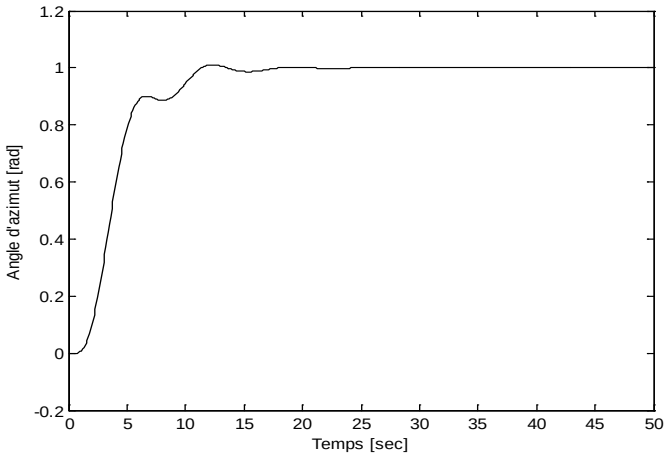


Figure 6.12 : Angle d'azimut

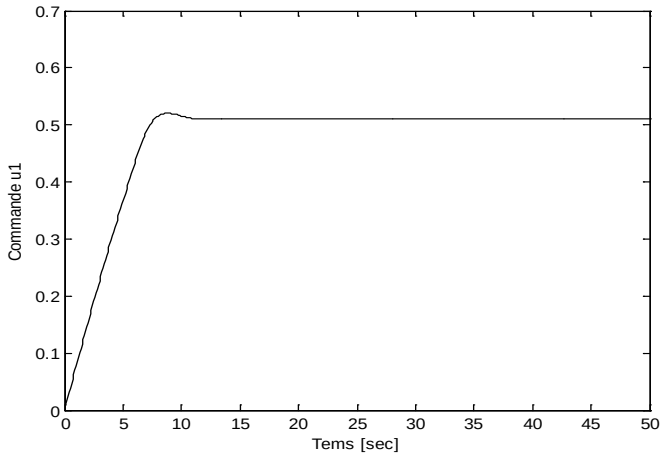


Figure 6.12 : Commande u_1

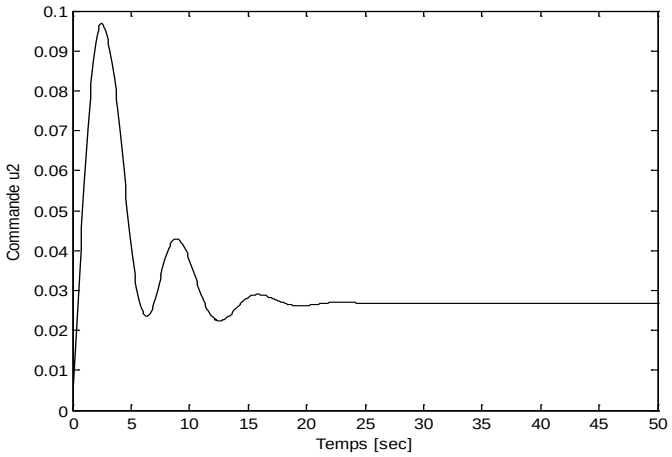


Figure 6.13 : Commande u_2

6.6 Conclusion

Dans ce chapitre, nous avons proposé une méthode de réglage des paramètres des contrôleurs flous par l'utilisation d'un algorithme d'optimisation par colonies de fourmis.

Dans cet algorithme, nous avons proposé des nouvelles règles pour la mise à jour de la phéromone et pour la sélection heuristique des paramètres par les fourmis.

La méthode proposée est utilisée en particulier pour l'optimisation des PID flous typiques présentés au quatrième chapitre. Deux exemples de simulations ont été fournis. Le premier consiste en le réglage des paramètres d'un PID flou linéaire placé dans une chaine de commande monovariable. Dans le deuxième exemple, la méthode est utilisée pour le réglage simultané des paramètres de deux PI flous non linéaires placés dans une chaine de commande multivariable décentralisée. Dans les deux cas, la méthode a montré de bonnes performances.

Conclusion générale

Le travail présenté dans ce mémoire contribue essentiellement à la résolution des problèmes de réglage des contrôleurs flous. Nous nous sommes concentrés sur les contrôleurs flous de type PID. Le travail présenté se divise en deux parties :

Dans une première partie, nous avons présenté les concepts des mots clés de ce travail à savoir; la commande floue, l'apprentissage par renforcement et l'optimisation par colonies de fourmis. Au début, la théorie de la commande par logique floue a été exposée et l'architecture de base d'un contrôleur flou est présentée. Le fonctionnement d'un contrôleur flou dépend d'un nombre important de paramètres qu'il faut déterminer afin d'optimiser ses performances. Les contrôleurs flous présentent la possibilité d'incorporer des connaissances expertes dans leurs structures, ce qui peut faciliter le bon choix des paramètres.

Par la suite, nous avons exposé les principes de l'apprentissage par renforcement et l'optimisation par colonies de fourmis, ces deux méthodes permettent d'explorer de façon très efficace l'espace des solutions possibles d'un problème d'optimisation. En faisant l'état de l'art des ces méthodes, on a constaté que plusieurs variantes de ces algorithme on été proposés et appliqués pour la résolution de plusieurs problèmes d'optimisation

Dans la deuxième partie, les contributions de ce travail sont présentées. Dans un premier lieu, nous nous sommes intéressés par l'étude de la structure analytique d'une classe typique de contrôleurs flous. Nous avons utilisé un modèle de Takagi-Sugeno d'ordre zéro avec une méthode de conception spécifique : pour la fuzzification des entrées, on a utilisé des univers de discours symétriques avec des fonctions d'appartenances équidistantes, pour la sortie, on a utilisé un univers de discours symétrique avec des singletons équidistants, le produit algébrique comme opérateur d'implication flou

et la méthode de la moyenne pondérée pour l'opération de défuzzification. La méthode de conception est présentée et la structure analytique de ces contrôleurs est développée. On a démontré mathématiquement et de façon claire que la structure analytique de ces contrôleurs PID flous ainsi construits est équivalente à celle des PID classiques ce qui nous a permis d'adapter la méthode de la réponse en fréquence de Ziegler-Nichols pour le réglage de ces contrôleurs PID flous. La méthode est validée par un exemple de simulation. Ces contrôleurs PID flous présentent aussi des avantages considérables par rapport à leur conception de base ; à savoir leurs paramètres sont interprétables physiquement et leurs structures présentent une faculté à introduire des connaissances expertes. Par conséquent, de tels contrôleurs sont plus facilement ajustables que leurs homologues classiques.

En second lieu, notre travail s'est orienté vers la recherche de nouvelles méthodes de réglage des paramètres des contrôleurs flous. En particulier, les PID flous. Nous nous sommes penchés sur les méthodes d'optimisation inspirées de la nature qui ont été gâtées par un intérêt considérable de la communauté des sciences techniques pendant ces dernières années. En particulier, les algorithmes d'apprentissage par renforcement et les algorithmes de colonies de fourmis ont eu un grand succès à travers plusieurs problèmes d'optimisation. Cependant, malgré leur succès, ces algorithmes n'ont pas été utilisés auparavant pour le réglage des paramètres des contrôleurs flous. Nous avons proposé des méthodes sur la base ces approches pour le réglage des paramètres des contrôleurs flous.

Dans la première méthode, on a proposé l'utilisation de l'apprentissage par renforcement pour le réglage des contrôleurs flous. Pour cette méthode deux algorithmes basés sur le Q-learning ont été proposées. Le premier peut être appliqué à tous les paramètres d'un contrôleur flou. En particulier, les contrôleurs PID flous linéaires et non linéaires. Dans le cas des contrôleurs « linéaires », seuls les valeurs maximales des univers de discours sont ajustées en ligne afin de minimiser un critère de performance. Dans le cas des contrôleurs flous non linéaires, on peut procéder de deux façons ; soit commencer le réglage avec un contrôleur initialement « linéaire » et puis le comportement non linéaire est obtenu par ajustement des positions de toutes les fonctions d'appartenance sur l'espace d'entrée et/ou les positions des singletons sur l'espace de sortie ou soit par la considération dès le début, que tous les paramètres sont soumis à l'apprentissage. La phase d'apprentissage peut être accélérée par l'introduction des connaissances dans le contrôleur initial. L'efficacité des algorithmes proposés est montrée par trois exemples de simulation. Dans le premier exemple de simulation, le Q-learning est utilisé pour le réglage des paramètres d'un PID flou linéaire pour stabilisation d'un pendule inversé afin de minimiser

la somme des erreurs quadratiques d'une réponse à un échelon. Dans le deuxième exemple, le même algorithme est utilisé pour le réglage simultané des paramètres de deux PI flous « non linéaires » placés dans une chaine de commande multivariable décentralisée pour la stabilisation d'un simulateur d'hélicoptère. La fonction de performance est composé par la somme des erreurs quadratiques de la réponse à un échelon des deux sous-systèmes du modèle de l'hélicoptère. Dans le troisième exemple de simulation, une version modifiée de l'algorithme Q-learning est utilisée pour le réglage des conclusions des règles d'un contrôleur flou pour la navigation réactive d'un robot mobile.

Dans la deuxième méthode, on a proposé l'utilisation de l'optimisation par colonies de fourmis pour le réglage des contrôleurs flous. Dans cette méthode, la recherche des paramètres optimaux par les fourmis est guidée par une nouvelle règle de mise à jour de la phéromone et une nouvelle loi d'exploration pseudo-stochastique. La recherche peut être accélérée par l'introduction des connaissances dans le contrôleur initial. Deux exemples de simulation sont fournis pour montrer les performances de la méthode proposée. Le premier consiste en le réglage des paramètres d'un PID flou linéaire pour la stabilisation d'un pendule inversé. Dans le deuxième exemple, la méthode est utilisée pour le réglage simultané des paramètres de deux PI flous non linéaires placés dans une chaine de commande multivariable décentralisée pour la stabilisation d'un simulateur d'hélicoptère.

Enfin, nous pensons que le travail présenté dans cette thèse ouvre de nouvelles perspectives selon les principales directions suivantes :

- La comparaison des méthodes développées entre elles et avec d'autres algorithmes stochastiques tels que les algorithmes génétiques.
- L'extension des méthodes d'apprentissage par renforcement et l'optimisation par colonies de fourmis à d'autres types de commande.
- L'extension de ces méthodes pour le réglage continu des paramètres des contrôleurs.
- L'hybridation de ces méthodes entre elles ou avec d'autres méthodes d'optimisation dans le but d'accélérer la recherche et d'améliorer les performances.
- La mise en œuvre pratique des algorithmes proposés.

Bibliographie

[ADA01] J. M. Adams and K. S. Rattan," A genetic multi-stage fuzzy PID controller with a fuzzy switch," in *Proc. IEEE Int. Conf.* ***on*** *Systems, Man, and Cybernetics*, Tucson, AZ, USA, 4, pp. 2239-2244, 2001.

[AST95] K. J. Åström and T. Hägglund,*" PID controllers, theory, design and tuning,"* Instrument Society of America, Research Triangle Park, North Carolina, 1995.

[BEK92] Beckers, R., Deneubourg, J. L., and Goss, S., "Trails and U-Turns in the Selection of a Path by the Ant Lasius Niger," *J. Theor. Biol.*, 159, 397-415, 1992.

[BEL57] R. E. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, New Jersey, USA, 1957.

[BHA04] S. Bhattacharya, A. Chatterjee, and S. Munshi, "A new self-tuned PID-type fuzzy controller as a combination of two-term controllers,"*ISA Transactions*, Vol. 43, pp.413–426, September 2004.

[BOU09a] **H. Boubertakh**, M. Tadjine, P.Y. Glorennec and S. Labiod," Tuning Fuzzy PID controllers using Ant Colony Optimization", in *Proc. Mediterranean Conference of Control and Automation* (MED'09), *Thessaloniki*, pp.13-18, 2009.

[**BOU09b]** **H. Boubertakh**, M. Tadjine, P.Y. Glorennec and S. Labiod," A New Mobile Robot Navigation Method using Fuzzy Logic and a Modified Q-Learning Algorithm ," Journal Intelligent and Fuzzy Systèmes, *Vol. 20, 2009.*

[BOU08a] **H. Boubertakh**, M. Tadjine, P.Y. Glorennec and S. Labiod," Tuning Fuzzy PID controllers using Q-learning Algorithm," *Archives of Control Sciences*, Vol. 18 No. 4, 2008.

[BOU08b] **H. Boubertakh**, M. Tadjine and P. Y. Glorennec, " A simple Goal Seeking Navigation Method for a Mobile Robot using Human Sense, Fuzzy Logic and Reinforcement Learning," *Lecture Notes in computer Science*, vol. 5177, pp.666-673, 2008.

[BOU08c] **H. Boubertakh**, M. Tadjine, P.Y. Glorennec and S. Labiod, " Comparison between Fuzzy PI, PD and PID controllers and Classical PI, PD and PID controllers," *International Review of Automatic Control*,Vol.1 No. 4, 2008.

[BOU06a] **H. Boubertakh**, P. Y. Glorennec,"Optimization of a Fuzzy PI Controller Using Reinforcement Learning," in *Proc. IEEE Int. Conf. on Information and Communication Technologies: From Theory to Applications*, Damascus, pp. 1657-1662, 2006.

[BOU06b] **H. Boubertakh**, S. Labiod et M. Tadjine, "Un contrôleur flou optimise par renforcement pour la navigation d'un robot mobile", *la 5ème Conférence sur le Génie Electrique (CGE'05)*, Ecole Militaire Polytechnique, Bordj El Bahri, Alger, avril 2006.

[BUH94] H. Bühler, *Réglages par logique floue*, Presses polytechniques et Universitaires Romandes, 1994.

[CHA06] W. Chatrattanawuth, N. Suksariwattanagul, T. Benjanarasuth and J. Ngamwiwit, "Fuzzy I-PD Controller for Level Control," *Proc. of the Int. Joint Conference*, Busan, Korea, pp. 5649-5652, 2006.

[CHU98] H.-Y. Chung, B. –C. Chen, J.-J. Lin,"A PI-type fuzzy controller with self-tuning scaling factors," *Fuzzy Sets and Systems*, 93, pp. 23-28, 1998.

[CLE02] M. Clerc, and J. Kennedy,"The particle swarm - explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. Evolutionary Computation*, vol. 6, no. 1, pp.58-73, 2002.

[DAH04] Y. Dahmani and A.Benyettou, " Fuzzy Reinforcement Rectilinear Trajectory Learning," *Journal of Applied Science*, Vol. 4, No. 3, pp.388-392, 2004.

[DEN90] J.-L. Deneubourg, S. Aron, S. Goss, and J.-M. Pasteels, "The self-organizing exploratory pattern of the Argentine ant," *Journal of Insect Behavior*, 3, pp.159–168, 1990.

[DIN03] Y. Ding , H. Ying and S. Shao," Typical Takagi–Sugeno PI and PD fuzzy controllers: analytical structures and stability analysis," *Fuzzy Sets and Systems*, 151, pp.245–262, 2003.

[DOR97] M. Dorigo and L.M. Gambardella,"Ant colony system: A cooperative learning approach to the travelling salesman problem," *IEEE Trans. on Evolutionary Computation*, vol. 1, no. 1, pp. 53-66, 1997.

[DOR96a] M. Dorigo, V. Maniezzo and A. Colomi,"Ant system: Optimization by a colony of cooperating agents," *IEEE Trans.* on *Systems, Man, and Cybernetics-Part B*, 26(1), 29-41, 1996.

[DOR96b] M. Dorigo, L. M. Gambardella, "Ant Colonies for the Traveling Salesman Problem," *Biosystems*, Vol. 43, pp. 73-81, 1996.

[DOR04] M. Dorigo and T. Stützle, *Ant Colony Optimization*, MIT Press, Cambridge, MA, USA, 2004.

[DOR06] M. Dorigo, M. Birattari, and T. Stützle, "Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique," *IEEE Computational Intelligence Magazine*, Vol.1, No. 4, pp.28–39, 2006.

[DOR99] M. Dorigo and G. Di Caro, “The Ant Colony Optimization metaheuristic,” in *New Ideas in Optimization*, D. Corne *et al.*, Eds. McGraw Hill, London, UK, pp.11–32, 1999.

[DRE03] J. Dréo, A. Pétrowski, P. Searry, E. Taillard, *Méthaheuristiques pour l’optimisation difficile*, Edition Eyrolles, 2003.

[DUA06] H.-B. Duan, D.-B Wang and X.-F. Yu,” Novel Approach to Nonlinear PID parameter optimization Using Ant Colony optimization Algorithm,” *J. of Bionic Engineering*, 3, pp.73-78, 2006.

[GAM95] L.M. Gambardella, M. Dorigo , “Ant-Q : A Reinforcement Learning Approach to the Traveling Salesman Problem,” *Proc. of the 12th International Conference of machine Learning*, pp. 252-260, Palo Alto, Morgan Kaufmann, 1995.

[GAM99] L. M. Gambardella, E. D. Taillard, and G. Agazzi, “MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows,” *In D. Corne, M. Dorigo, and F. Glover, editors, New Ideas in Optimization*, pages 63–76. Mc-Graw Hill, London, UK, 1999.

[GOS89] S. Goss, S. Aron, J. L. Deneubourg, and J. M. Pasteels, “Self-Organized Shortcuts in the Argentine Ant,” *Naturwissenchaften*, 76, pp.579-581, 1989.

[GLO94] P.Y. Glorennec, “Fuzzy Q-learning and dynamical fuzzy Q-learning,” *Proc. Third IEEE Inter.Conf. on Fuzzy Systems*, IEEE Press, Piscataway, NJ. , Orlando, FL, USA, pp. 474-479, 1994.

[GLO99] P. Y. Gloronnec, *Algorithmes d’apprentissage pour les systèmes d’inférence floue*, Edition Hermes, 1999.

[GU04] Gu, D., Hu H., “Accuracy Based Fuzzy Q-Learning for Robot Behaviors,” *Proc. of the IEEE International Conference on Fuzzy Systems (IEEE-FUZZY2004)*, Budapest, pp. 25-29, July 2004.

[HUM02] Humusoft, *CE 150 helicopter model: User's manual*, Humusoft, Prague, 2002.

[JAN07] J. Jantzen, *Foundations of Fuzzy Control*, West Sussex (England), John Wiley & Sons Ltd, 2007.

[KAE96] L.P. Kaelbling, M. L. Littman, A. W. Moore, “Reinforcement Learning: A Survey,” *J. of artificial Intelligence Research*, 4, pp.237-285, 1996.

[KEN01] J. Kennedy, R. C. Eberhart and Y. Shi, *Swarm Intelligence*, San Francisco, Morgan Kaufmann, 2001.

[KER06] S. Kermiche, M. L. Saidi and H.A. Abbassi, “Gradient Descent Adjusting Takagi-Sugeno Controller for a Navigation of Robot Manipulator,” *Journal of Engineering and Applied Science*, Vol. 1, No 1, pp.24-29, 2006.

[KIM02] B. J. Kim and C. C. Chung, “Design of Fuzzy PD+I Controller for Tracking Control,” *Proc. American Control Conference*, Anchorage, USA, pp. 2124-2129, 2002.

[KIS85] J.B. Kiszka, M.M. Gupta, and P.N. Nikiforuk. “Energetic stability of fuzzy dynamic systems,” *IEEE Trans. on Syst. Man Cybern.*, SMC-15(5), pp.783–792, 1985.

[KO06] C.-N. Ko, T.-L. Lee, H.-T. Fan, and C.-J. Wu," Genetic Auto-Tuning and Rule Reduction of Fuzzy PID Controllers," *Proc. IEEE Int. Conf. on Systems. Man, and Cybernetics*, Taipei, Taiwan, pp. 1096-1101, 2006.

[LI01] W. Li, X.G. Chang, J. Farrell, F. M. Wahl, "Design of an enhanced hybrid fuzzy P+ID controller for a mechanical manipulator," *IEEE Trans. on Systems, Man, and Cybernetics*, Part B, 31, pp. 938– 945, 2001.

[LU04] H. Lu, T. Mei, R. Wang, and M. Q. -H. Meng," Application of Self-tuning Fuzzy PI+PD Controller in Joint DC Servo Motors of a Four-leg Robot," *Proc. Inter. Conf. on Information Acquisition*, Beijing, China, pp. 520 – 523, 2004.

[MAM74] E.H. Mamdani, "Application of Fuzzy Algorithms for Control of Simple Dynamic Plant," *Proc. of IEEE*, Vol. 121, No. 12, pp. 1585-1588, 1974.

[MAM77] E. H. Mamdani, "Application of fuzzy logic to approximate reasoning using linguistic synthesis," *IEEE Trans. Computers*, vol. 26, no. 12, pp. 1182-1191, Dec. 1977.

[MAM75] E. H. Mamdani and S. Assilian," An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," *Int. J. of Man-Machine Studie*s, 7 (1) pp. 1-13, 1975.

[MEN02] R. Mendes, P. Cortez, M. Rocha and J. Neves,"Particle swarms for feedforward neural network training," in *Proc. of Int. Joint Conf. on Neural Networks*, pp. 1895-1899, 2002.

[MOH02] B. M. Mohan and Ambalal V. Patel, "Analytical Structures and Analysis of the Simplest Fuzzy PD Controllers," *IEEE Trans. on Systems, Man, and Cybernetics*, Part B: Cybernetics, 32, pp.239-248, 2002.

[MOH08] B. M. Mohan and A. Sinha, "Analytical structure and stability analysis of a fuzzy PID controller," *Applied Soft Computing*, 8, pp.749-758, 2008.

[MOO95] B. S. Moon," Equivalence between fuzzy logic controllers and PI controllers for single input systems," *Fuzzy Sets and Systems*, 69, pp. 105-113, 1995.

[MUZ95] M. Muzimuto,"Realization of PID controls by fuzzy control methods," *Fuzzy Sets and Systems*, 105, pp. 171-182, 1995.

[PED93] W. Pedrycz, *Fuzzy control and fuzzy systems*, Wiley, New york, 1993.

[PIV98] P. Pivoňka, "Fuzzy PI+D Controller with Normalized Universe," in *Proc. 6th European Congress on Intelligent Techniques and Soft Computing (EUFIT '98)*, Aachen, Germany, pp. 890-894, 1998.

[ROY04] A. Roy, K. Iqbal, and D.P. Atherton," Optimum Tuning of PI-PD Controllers for Unstable Sampled-Data Control Systems," *Proc. 5th Asian Control Conference*, Melbourne, Australia, pp. 478- 485, 2004.

[SAN02] E. N. Sanchez, V. Flores, "Real Time Fuzzy PI+PD Control for an Underactuated Robot," *Proc. IEEE Int. Symp. of Intelligent Control*, Vancover, Canada, pp.137-147, 2002.

[SIM03] K. M. Sim and W. H. Sun, "Ant colony optimization for routing and load-balancing: survey and new directions," *IEEE Trans. on Systems, Man, and Cybernetics*, Part A: Systems and Humans, Vol. 33, No. 5, pp. 560-572, 2003.

[STU00] T. Stützle and H. H. Hoos, “MAX–MIN Ant System,” *Future Generation Computer Systems*, Vol.16, No.8, pp.889–914, 2000.

[SUG85] M. Sugeno and G. T. Kang, “Fuzzy identification of systems and its applications to modeling and control,” *IEEE Trans. Syst., Man, Cybern.*, vol. 15, pp. 116-132, 1985.

[SUG84] M. Sugeno and K. Murakami, “Fuzzy parking control of model car,” *23 rd IEEE Conf. on Decision and Control*, 1984.

[SUT98] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Bradford Books. MIT Press, Cambridge, Mass., 1998.

[SUT88] R. S. Sutton,” Learning to predict by the method of temporal differences,” *Machine Learning*, 3, pp. 9-44, 1988.

[STU97] T. Stützle and H. H. Hoos, “The MAX–MIN Ant System and local search for the traveling salesman problem”, *T. Bäck, Z. Michalewicz, and X. Yao, editors, Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC’97)*, pages 309–314. IEEE Press, Piscataway, NJ, 1997.

[TAM86] T. Tamakawa, “High speed fuzzy controller hardware system,” *Proc. 2nd Fuzzy System Symp.*, pages 122–130, 1986.

[TAN01a] K. S. Tang, K. M. Man and G. Chen, “A G-A optimized Fuzzy PD+I Controller for Nonlinear Systems,” *Proc. of Int. Conf. on Industrial Electronics, Control, and Instrumentation* (IECON’01), Denver, USA, pp. 718-723, 2001.

[TAN01b] K. S. Tang, K. M. Man and G. Chen, “An Optimal Fuzzy PID Controller,” *IEEE Trans. on Industrial Electronics*, 48, pp.757-765, 2001.

[TOG86] M. Togai and H. Watanabe,” Expert system on a chip : An engine for real-time approximate reasoning,” *IEEE Expert Syst. Mag.*, 1 ,pp.55–62, 1986.

[VAR04] H. A. Varol and Z. Bingul, “A new PID Tuning Technique Using Ant Algorithm,” *Proc. American Control Conf.*, Boston, Massachusetts, USA, pp.2154-2159, 2004.

[VEE04] M. P. Veeraiah, S . Majhi and C. Mahanta, “Fuzzy Proportional Integral - Proportional Derivative (PI-PD) Controller,” *Proc. of the American Control Conf.*, Boston, Massachusetts, pp.4028-4033, 2004.

[WAT89] C. J. C. H. Watkins, Learning from delayed rewards. PhD Thesis, University of Cambridge, England, 1989.

[WAT92] C. J. C. H. Watkins and P. Dayan, ” Technical note, Q-learning, “*Machine Learning*, Vo. 8, pp. 279-292, 1992.

[WU07] C. –J. Wu, T.-L. Lee, Y.-Y. Fu and L.-C. Lai, “Auto-Tuning Fuzzy PID Control of a Pendubot System,” in *Proc. 4th IEEE Int. Conf. on Mechatronics*, Kumamoto Japan, pp.1-6, 2007.

[YAG94] R. R. Yager and D. P. Filev, *Essential of fuzzy modeling and control*, John Wily & Sons Inc., 1994.

[YIN98] H.Ying, “Constructing Nonlinear Variable Gain Controllers via the Takagi–Sugeno Fuzzy Control”, *IEEE Trans. on Fuzzy Systems*, Vol. 6, No. 2, pp.226-234, may 1998.

[ZAD65] L. A. Zadeh, “Fuzzy sets,” *Information and Control*, vol. 8, pp.338-353, 1965.

[ZAV03] P. G. Zavlangas and S. G. Tzafestas, “Motion control for Mobile Robot Obstacle Avoidance and Navigation: A Fuzzy Logic-Based Approach,” *Systems Analysis Modelling Simulation*. Vol. 43, No. 12, pp. 1625-1637 , 2003.

[ZHA06] J. Zhang, D. Yu, and S. Qi, “Structural research of fuzzy PID controllers,” *Proc. IEEE Int. Conf. on Control and Automation (ICCA)*, Budapest, Hungary, pp.1248-1253, June 2005.

[ZHE92] L. Zheng, Co. Yamatake-Honeywell, “A practical guide to tune of proportional and integral (PI) like fuzzy controllers,” *Proc. IEEE Int. Conf. on Fuzzy System*s, San Diego, CA, USA, pp.633-640, 1992.

ملخص:
العمل المقدم في هذه الأطروحة يساهم بصفة خاصة في حل مشاكل تعديل وسائط الضابطات المبهمة. ينقسم هذا العمل إلي جزأين:
في الجزء الأول، وضحنا طريقة نموذجية لإنشاء الضابطات المبهمة من نوع تناسبي-تكاملي-تفاضلي(PID). ثم برهنا رياضيا أن هذه الضابطات تكافئ نظيراتها الكلاسيكية. كما أن لها ايجابيات متعلقة ببنيتها الأساسية؛ وسائطها يمكن وصفها فيزيائيا وبنيتها تسمح باستغلال معارف الخبراء حول نظام التحكم وسهلة التحول إلى دالة لا خطية، وهذا مفيد عند القيام بتعديل الوسائط.
في الجزء الثاني لهذا العمل، اقترحنا طريقتين للبحث عن القيم المثلى لوسائط الضابطات المبهمة باستعمال الطرق التطورية. الطريقة الأولى تعتمد على خوارزمية التعليم بالتقوية:التعليم بالقيم Q (Q-learning)، حيث يتم البحث تكراريا عن قيم الوسائط المثلى بهدف تصغير دالة لكفاءة أداء نظام التحكم. الطريقة الثانية تعتمد أساسا على خوارزميات البحث المثالي باستعمال مستعمرات النمل. لكل طريقة، وضحنا بعض نتائج المحاكاة العددية لإبراز فعاليتها.

كلمات مفتاحية : تحكم غامض، ضابطات تناسبية-تكاملية-تفاضلية مبهمة، ضابطات تناسبية-تكاملية-تفاضلية كلاسيكية، تعليم بالتقوية، بحث مثالي باستعمال مستعمرات النمل.

Résumé :
Le travail présenté dans cette thèse contribue essentiellement à la résolution des problèmes de réglage des contrôleurs flous. Il est divisé en deux parties :
Dans la première partie, On a présenté une méthode typique de conception des contrôleurs PID flous. Puis, on a démontré mathématiquement que ces contrôleurs sont équivalents à leurs homologues classiques. Cependant, ils présentent des avantages liés à leur conception de base ; leurs paramètres sont interprétables physiquement, et leur structure présente la faculté d'introduire des connaissances expertes et une flexibilité de se transformer en une fonction non linéaire, ce qui est utile pour le réglage de leurs paramètres.
Dans la deuxième partie de ce travail, on a proposé deux algorithmes d'optimisation des contrôleurs flous par des méthodes sthochastiques. Le premier algorithme est basé sur un algorithme d'apprentissage par renforcement : le Q-learning, il est utilisé pour l'optimisation des paramètres d'un contrôleur flou afin de minimiser itérativement une fonction de performance du système de commande. Le deuxième algorithme est basé sur les méthodes d'optimisation par colonies de fourmis. Pour chaque algorithme, des exemples de simulation sont présentés pour montrer son efficacité.

Mots-clés : Commande floue, Contrôleurs PID flous, Contrôleurs PID classiques, Apprentissage par renforcement, Optimisation par colonies de fourmis.

Abstract:
The work presented in this thesis contributes essentially to solve the problem of tuning the parameters of fuzzy controllers. It is divided on two parts:
In the first part, we show a typical design method of fuzzy PID controllers. After, we demonstrate mathematically that these controllers are equivalent to their classical counterparts. However, they have other advantages related to their basic structure; their parameters are physically interpretable and their structure presents the faculty of knowledge incorporation ant has the flexibility to be nonlinear, which is useful in parameters tuning.
In the second part, we propose two algorithms for tuning parameters of fuzzy controllers. The first one is based on the reinforcement learning algorithm; Q-learning, it searches iteratively the optimal parameters so that a performance function is less than a satisfaction threshold. The second algorithm use ant colony optimization methods to search the optimal parameters of the fuzzy controllers. For each algorithm, some simulation examples are given to show its effectiveness.

Key words: Fuzzy control, Fuzzy PID controllers, Classical PID controllers, Reinforcement learning, Ant colony optimization.